Vehicle Sideslip Angle Estimation using a Hybrid Approach

Vehicle Sideslip Angle Estimation using the combination of Uncertainty Neural Networks and Kalman Filter variations

Dennis de Mol

Vehicle Sideslip Angle Estimation using a Hybrid Approach

Vehicle Sideslip Angle Estimation using the combination of Uncertainty Neural Networks and Kalman Filter variations

by

Dennis de Mol

For the degree of Master of Science in Vehicle Engineering at Delft University of Technology

Student number:	4437780		
Comittee:	Dr.ir. B. Shyrokau,	TU Delft	Supervisor
	A. Bertipaglia, MSc	TU Delft	Supervisor
	Dr.ir. R. Happee	TU Delft	Chair
	Dr.ir. M. Alirezaei	TU Eindhoven	
Graduation date:	16-03-2022		
Under embargo until:	16-03-2024		

TUDelft

Abstract

Accurate and robust vehicle state estimation is important for proper operation of vehicle control systems. For lateral stability control accurate estimation of the Vehicle Sideslip Angle (VSA) is of utmost importance. This thesis aims to develop an accurate and highly robust model to estimate the VSA. Common vehicle sensors such as the Inertial Measurement Unit (IMU), Global Positioning System (GPS) and steering angle sensor are used to provide measurements of vehicle states that are relatively easy to directly measure. Additionally, wheel bearing strains measurements are collected by the Load-Sensing Bearing (LSB), which can be mapped to the tire forces that are measured by wheel force transducers. Insight regarding the tire forces could be beneficial to VSA estimation. The focus lies upon neural network estimators and hybrid structures, which combine the neural network estimator with an observer model.
A large-scale experimental dataset composed of standardised vehicle manoeuvres is used to develop and evaluate different estimation architectures. The dataset consists of 216 manoeuvres, which corresponds to approximately two hours of driving time.

Neural network estimators are data-driven and therefore rely on high quality data. Various neural network architectures exist for the purpose of time series estimation. In this thesis the Feedforward Neural Network (FFNN), Recurrent Neural Network (RNN) and Transformer are examined in detail. The FFNN and RNN yield similar performance in terms of Root Mean Squared Error (RMSE) and Maximum Error (ME) on the test set. The Transformer is unable to match this performance level due to the absence of a proper positional encoding of the measurements. Due to the simpler structure and lower data requirement, the FFNN is selected for application in the hybrid structures.
To create a hybrid estimator, the uncertainty level of the VSA estimate from the neural network is required. To obtain the uncertainty using a neural network, various methods exist such as Monte Carlo Dropout (MCDO), Monte Carlo Batch Normalisation (MCBN) and the Uncertainty Deep Ensemble (UDE). These methods are compared using three metrics, the RMSE, Predictive Loglikelihood (PLL) and Continuous Ranked Probability Score (CRPS). A combination of these metrics does not only evaluate the estimation accuracy, but also the quality of the corresponding estimated uncertainty level. The UDE consisting of FFNNs provides the best performance and even outperforms the single FFNN in terms of RMSE by a decrease of 8.6%. Therefore, the UDE is used to develop the different hybrid structures.

The Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) are built on a nonlinear bicycle model. Different sets of inputs for the observers, as well as constant or adaptive covariance matrices create different variations. These observer variations are combined with the UDE to form hybrid estimation models. The different variations show equal performance due to the high accuracy of the UDE on the test set. The high performance of the UDE causes the measurement noise to be tuned to a very low level, which results in the observer 'trusting' the UDE estimations. Therefore, the performance of the different hybrid structures is almost equal to that of the original UDE.

To investigate if the observer is actually able to correct neural network estimations, a suboptimal neural network is created. This network is trained only on measurements with an absolute lateral acceleration lower than 5 m/s^2. This mimics sparse data of high sideslip angles, a common problem for data-driven approaches in this setting. This causes the uncertainty of the neural network estimations to be higher in these operating regions. A linear scaling of the uncertainty level to match the variance of the noise on the other measurements does not yield satisfactory results. However, an exponential scaling of the uncertainty level causes a higher differentiation between low and high confidence levels. This exponential scaling decreases the RMSE of the hybrid structure with 11.4 %. Furthermore, a method for adapting the process noise based on the quality of the observer model estimation is implemented. This equates to similar performance in terms of RMSE, but introduces a number of additional parameters to tune.

Acronyms

AD	Adaptive.
ADAS	Advanced Driver Assistance Systems.
CoG	Centre of Gravity.
CRPS	Continuous Ranked Probability Score.
DE	Deep Ensemble.
DoF	Degrees of Freedom.
EKF	Extended Kalman Filter.
ESC	Electronic Stability Control.
FFNN	Feedforward Neural Network.
GPS	Global Positioning System.
GRU	Gated Recurrent Unit.
IMU	Inertial Measurement Unit.
KF	Kalman Filter.
LSB	Load-Sensing Bearing.
LSTM	Long Short-Term Memory.
MCBN	Monte Carlo Batch Normalisation.
MCDO	Monte Carlo Dropout.
ME	Maximum Error.
NLP	Natural Language Processing.
PLL	Predictive Loglikelihood.
RMSE	Root Mean Squared Error.
RNN	Recurrent Neural Network.
ST	Standard.
TF	Tire forces.
TSBO	Two-Stage Bayesian Optimisation.
UDE	Uncertainty Deep Ensemble.
UKF	Unscented Kalman Filter.
VSA	Vehicle Sideslip Angle.
XM	Extra measurements.

Contents

1 Introduction

1.1. Motivation

Safety is a fundamental objective of the automotive industry, unfortunately complete safety will never be reached with human drivers. Already, driving has become a lot safer over time due to many innovations in physical systems, as well as in control systems, but the factor of human error will always be present. Autonomous driving has the potential to guarantee a higher level of safety. Commercially sold automated vehicles are coming closer every day, but a key factor pushing the technology back still is how to ensure complete safety while driving. To achieve this Advanced Driver Assistance Systems (ADAS) have to be designed that are very robust such that full automation can be reached. A requirement for proper operation of these is accurate and robust state estimation of the vehicle, only if the system is aware of the state of the vehicle, it can be controlled properly.

A well-known assistance system is Electronic Stability Control (ESC), ensuring lateral stability by assisting the driver when he or she threatens to lose control over the vehicle. For proper performance of ESC, accurate and robust information about various vehicle states is required. For example, these vehicle states include the longitudinal speed, yaw rate and lateral acceleration of the vehicle. The increase of reliance on ADAS due to automated driving raises the demand for reliable state estimation. Over time when there is no driver to intervene anymore, it is essential that the state of the vehicle is estimated robustly and accurately.
A vehicle state that is of particular interest for ESC is the Vehicle Sideslip Angle (VSA). The VSA along with the yaw rate are important variables for verification of vehicle stability as well as its handling characteristics such as understeer or oversteer (Keller et al., 2015; Rajamani, 2011). Also, the VSA can be controlled directly to extend the limit of stable vehicle cornering (Lu et al., 2016). Accurate and robust estimation of the VSA is required when controlling the state directly.

The VSA is defined as the angle between the longitudinal axis of the vehicle and the direction of the velocity vector of the vehicle measured at the Centre of Gravity (CoG) of the vehicle. In other words, it is the angle between the heading and moving direction of the vehicle. The VSA β is mathematically expressed in Equation 1.1 and can be visually interpreted from Figure 1.1. Here V_y and V_x denote the respective lateral and longitudinal component of the vehicle velocity vector at the CoG.

$$\beta = atan\left(\frac{V_y}{V_x}\right) \tag{1.1}$$

Tire force estimates available by the presence of the Load-Sensing Bearing (LSB) may be very useful for the task of VSA estimation (Kerst et al., 2016). However, it remains difficult due to the highly nonlinear relation between sideslip angle and tire forces. Unfortunately, it is also very difficult to measure this VSA directly using sensors. Direct measurement is only possible via an optical sensor or advanced GPS-inertial sensors combinations. These sensors are very expensive and not robust enough to apply in commercially used vehicles. For example, the quality of the optical sensor estimate depends on good lighting and a clear view. Additionally, operations such as calibration can further complicate the use of these sensors.

Differential braking via ESC is very useful when performing evasive manoeuvres or driving on low friction surfaces (Chatrath et al., 2020; Chowdhri et al., 2021). During these manoeuvres the VSA can become very large and the tires will start to operate in the saturation region. If so, it may be too late to restore control of the vehicle (Chakraborty et al., 2011).

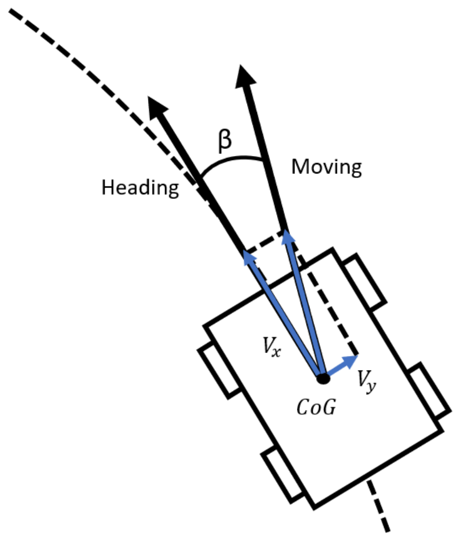

Figure 1.1: Vehicle sideslip angle

1.2. Objective

Increasing the performance and reliability of active safety systems is an open challenge that becomes more significant with the introduction of automated vehicles. The objective of the thesis report is to design a data-driven and hybrid model that is able to estimate the VSA with high accuracy and robustness using tire force information. To achieve this several subgoals, which are highlighted in this report, are identified.

- Neural network selection: Investigate what neural network architecture is best suited for the task of VSA estimation.
- Uncertainty estimation: Identify the optimal method for approximating the confidence level of the VSA estimations from the neural networks.
- Hybrid model selection: Find the optimal architecture for the hybrid model for the task of VSA estimation.
- Robustness analysis: Develop a method to investigate how the hybrid architecture deals with inaccurate predictions of the neural network.

This thesis is unique in current literature since it uses the measurements of a LSB as an input to the data-driven model. These bearing strains can be used to compute the tire forces, which can be very beneficial for the estimation. Additionally, more widely available sensors are used to measure other vehicle states. The availability of the bearing strain measurements eliminates the need for complex tire models. If the performance of the sideslip angle estimation is greatly improved by the availability of tire forces, this would indicate the value of load sensing bearings. What neural network architecture is best for estimation using tire force information needs to be evaluated.

Within current literature the proposed estimation architectures are only evaluated on a small selection of manoeuvres. The used datasets are small and less than ten tests are used to compare the performance of the different models. In this thesis the goal is to develop and compare the performance of different models on a large experimental dataset where numerous different driving dynamics are present.

In addition to the use of purely data-driven models, different observer models have been developed to contribute in the task of VSA estimation. Hybrid models, constructed from the combination of observers and data-driven estimators, are developed in an attempt to achieve accurate and robust estimation. Hybrid architectures require uncertainty estimates from the data-driven model. Different methods based on neural networks are available to compute such estimates. It needs to be evaluated which of those methods achieves the highest performance. Furthermore, various hybrid model structures exist in literature, the structure that is optimal for the task at hand needs to be determined. Also, the added robustness of the hybrid models must be evaluated to make conclusions regarding the added value of these structures, such results have not been published yet.

1.3. Outline

The thesis is structured in the following manner, in Chapter 2 the related work in the field of VSA estimation is summarized. In Chapter 3 the data collection and methodology of the research are discussed. Following, Chapter 4 introduces the used VSA estimation models along with their characteristics. Subsequently, these models are tuned and their performance is analysed in Chapter 5. At last, the conclusion of the research and future work follows in Chapter 6.

2 Related work

This chapter discusses the related work regarding VSA estimation. Within literature there are three main approaches, observer-based estimation, data-driven estimation and a combination of the two called hybrid estimation. These different approaches are divided into respective sections.

2.1. Observer-based approach

In some applications it is not possible to directly measure states of interest, if so the state observer may be a solution. The general structure of an observer is shown in Figure 2.1. The input to the plant, in this case the vehicle, is also presented to an observer model, here a vehicle model. The output of the plant can be measured directly using sensors, but not the hidden states, which may be of interest. However, the hidden states can be observed from the observer model. To make sure that the hidden states of the plant and the observer are approximately equal, the output is measured. When the plant and observer model diverge, the output error grows and there is corrected for this by an observer specific error weighting via a feedback loop.
There are different observers with slight variations from this structure. Additionally, the error weighting is different between observers. To achieve high performance of an observer, the used vehicle model must be accurate. This section discusses what vehicle models and types of observers are used in literature.

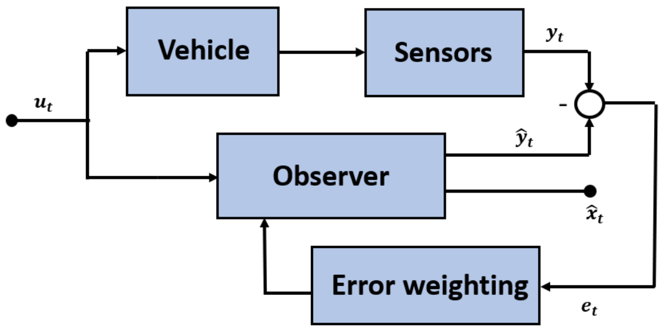

Figure 2.1: General observer structure

2.1.1. Vehicle model

Vehicle models can be divided into kinematic and dynamic models. When modelling in a kinematic manner there is relied on the geometry of the system. The motions are described without the consideration of the forces or torques. In the current application the vehicle motions are the motions in the system.
Dynamic modelling requires knowledge of the forces in the system. In these models, the vehicle dynamics are based on the equilibrium equations. Different models rely on different assumptions, which in turn affect the estimation accuracy of the model. To be able to use the equilibrium equations for the vehicle, tire forces must be accurately estimated. Due to tire forces being highly nonlinear, complex tire models need to be introduced. Also, the tire model must accurately represent the actual conditions, which may be difficult to achieve.

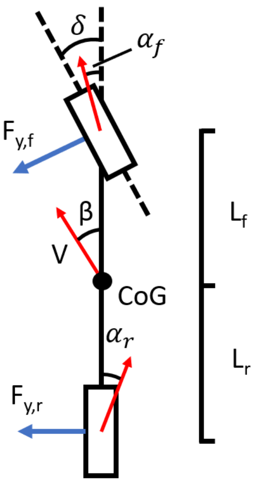

Figure 2.2: Planar single-track (bicycle) model

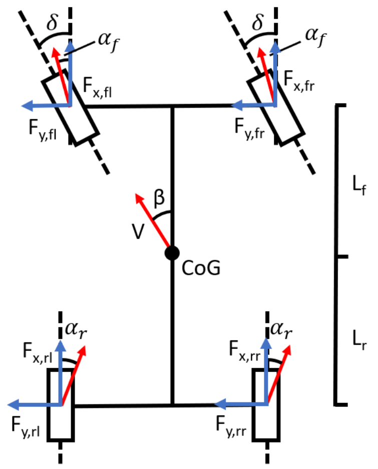

Figure 2.3: Planar two-track vehicle model

Vehicle models are clearly simplifications of reality, the respective disadvantage is that certain assumptions are introduced. When these assumptions are not fulfilled, the quality of the estimation can drop severely. Figure 2.2 and 2.3 show two different vehicle models, the bicycle model, developed by Segel (1956), and the two track model. In these figures the slip angles are denoted by α, the steering angle by δ, the tire forces of the different wheels by F and the vehicle velocity vector by V. Additionally, the distance from the CoG to both axles is defined by L_f and L_r. These are planar vehicle models to describe the dynamics of the chassis in the horizontal plane. These models are described in three Degrees of Freedom (DoF), namely longitudinal velocity, lateral velocity and yaw rate.

The bicycle model is clearly simpler, but also less accurate due to its simplification of only having two wheels. Thus, it is assumed that both wheels on an axle and the axle characteristics itself can be described by a single point located on the centre of the axles. There are also other assumptions that need to hold when working with the bicycle model, these include: constant forward velocity, no load transfer, linear tire ranges, no roll, pitch or vertical motion, 'ideal' steering dynamics, no suspension and no compliance effects. The two track vehicle model is also a simplification of the true system and several assumptions still need to holds such as no pitch and roll and that there is no suspension or compliance effects. However, the presence of 4 wheels can be very useful for dynamic modelling purposes.

The much simpler geometry of the kinematic vehicle model allows a simple analysis of how the motions in the model are related. It is substantially more difficult to derive these motions for a model with four wheels, as is the case for the two track model.
For dynamic modelling an analysis is made based on the forces present on the vehicle. The bicycle model makes a rigorous assumption that all front wheel forces can be described by a single point force. The same holds for the rear wheels. In contrast, the two track model can achieve a higher accuracy since the forces are applied at the true distance of the centre of the vehicle at the locations of the tires. This allows a more accurate calculation of the moments acting on the vehicle.

2.1.2. Kalman filter variations

The Kalman Filter (KF) is a well-known stochastic observer, it is an optimal filter that minimizes the variance around the estimate (Durbin and Koopman, 2012). It computes the state estimated via the modelled dynamic evolution of the system and the state inverted from the actual measurement. Via the reliability of both estimates a weighted average is computed, which is a compromise between the model and measurement accuracy. The regular KF is based on the assumption that the system is linear. Unfortunately, many vehicle models rely on nonlinear tire models since linear tire models are very inaccurate at higher slip angles. When high sideslip angles are reached, tire slip angles are also high and therefore linear tire models cannot be used. This also implies that the linear KF cannot be used in this case.

Extensions of the KF exist that are able to deal with nonlinear systems, namely the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF). The EKF uses a first-order Taylor expansion to linearize around the estimate of the current mean and covariance. Unfortunately, the first order approximation can cause significant linearisation errors. Another drawback of the EKF is that computation of the Jacobian matrix can be computationally expensive.

In contrast, the UKF uses the unscented transform which shows less linearisation error. The unscented transform works via representing the state distribution by a Gaussian random variable via a minimal set of sample points (Wan and Van Der Merwe, 2000). These sample points capture the true mean and covariance of the Gaussian random variable. When propagated through the true non-linear system, the posterior mean and covariance are captured accurately. This accuracy is up to that of a third-order Taylor expansion. The disadvantage of the UKF is that when the state dimension of system is high, a large number of sample points have to be propagated through the system. Additionally, all Kalman filter variations rely on the assumption that the distribution of the state to estimate is Gaussian.

In literature different implementations of the EKF for sideslip angle estimation have been presented (Naets et al., 2017; Sen et al., 2015). Chen and Hsieh use the EKF for sideslip angle estimation based on a kinematic model (B.-C. Chen and Hsieh, 2008). They evaluate the approach on different surfaces with different friction coefficients. A disadvantage of the approach is that the system becomes nearly unobservable when the yaw rate is low. When this occurs it is not possible to compute the estimation gain anymore.

Doumiati et al. (2010) apply the EKF and the UKF observer with a four-wheel vehicle model. The tire forces are estimated by the Dugoff tire model. The authors compare the performance of both observers and find that the UKF is better performing when the demands on the vehicle are high. They argue this is caused by the fact that tire behaviour and vehicle dynamics become extremely nonlinear under high loads. The EKF is based on first-order linearisation and thus the linearisation errors grow for highly nonlinear relations. In contrast, the UKF is able to deal better with this since it is able to capture higher order nonlinearities. They state that another advantage of the UKF is that is does not rely on the availability of the Jacobian matrix. The derivation of this matrix is nontrivial, especially when implementing complex tire models. Other UKF implementations for the goal of sideslip angle estimation are evaluated by Chen et al. and Van Aalst (J. Chen et al., 2016; van Aalst, 2020).

In this study bearing strain measurements are added for the purpose of estimation. There is already literature where the tire forces are used directly for the purpose of VSA estimation. Cheli et al. (2015) implement the smart tire measurements in an EKF based on a bicycle model. They conclude that no improvements are made by the addition of tire forces when the surface friction coefficient stays constant. However, when there is a jump of the surface friction coefficient, the addition of the tire forces is valuable since this can be recognized earlier.
Mazzilli et al. (2021) use the measurements from a smart tire system to determine the tire contact forces. These measurements are used as inputs to an UKF based on a nonlinear dynamic (7-DOF) double-track model. The availability of tire contact forces reduced the Normalised RMSE with 49% and 26% with different initialisations of the surface friction coefficient.
A limitation of both papers is that only a small number of tests is evaluated, for the case of Cheli and Mazzilli respectively three and six tests are evaluated. It is interesting to evaluate the performance of the estimation methods over a larger number of tests.

Next to the Kalman filter variations there is also literature available about other observers being used. These include the Luenberger observer (Cherouat et al., 2005; Ding et al., 2014) and the sliding mode observer (Y. Chen et al., 2014; Liu et al., 2012; Stephant et al., 2007), these will not be further discussed since they lie outside the scope of this thesis.

2.2. Data-driven approach

Since observer-based methods rely on vehicle models, they are restricted by the model assumptions. The neural network estimators are purely data-driven and are therefore model free. The most straightforward neural network architecture is the feedforward network with an input layer, a couple of hidden layers and an output layer. Xu et al. (2021) use that type of an architecture that consists of three hidden layers with respectively 12, 5 and 3 neurons. They use the logistic activation function in the hidden layers in combination with the backpropagation algorithm for learning the weights. A disadvantage of the use of a feedforward network is that transient effects cannot be modelled. To counter this problem the authors proposed to transform the inputs of the neural network to the frequency domain. The inputs consisted of the vertical accelerations of the accelerometers placed on the inner line of the tire. It is not clear how well this method is at dealing with transient behaviour due to the low number of evaluation tests.

Since previous measurements can possess predictive power, these earlier measurements can be used for prediction by RNNs. For VSA estimation multiple past measurements may have an influence on the current sideslip angle. Sieberg et al. (2021) make use of a network with recurrent layers containing the

Gated Recurrent Unit (GRU). The number of layers and the size of the layers is determined via sequential-based global optimization. The output layer of the network is a fully connected dense layer. The inputs of the neural network include the counter roll torques of both axes as well as the damping factors of the semi-active dampers. Unwanted oscillations that could not be explained appeared during estimation with the proposed model. This indicates the low robustness of the data-driven approach when low quality training data or no data is available. Ghosh et al. propose a RNN that uses the Long Short-Term Memory cell (2018). They use a network with 8 hidden layers consisting each of between 40 and 100 LSTM cells. A time window of 0.5 seconds for the input measurements provides the best results. The wheel speeds of all four wheels are included as inputs to the neural network estimator. This model needs to be trained for a full day before use and may be severely overfitting to the training data because of the large size of the estimator.

Bonfitto et al. (2020) propose the use of four different neural networks, three regression networks and a classification network. To deal with different road surfaces they use a parallel classification network for identifying the road conditions. The classification network tries to classify if the vehicle is located on a dry, wet or icy surface. Namely, the surface friction coefficient is very important for calculation of the VSA. The three regression networks are specifically trained for these different surface friction levels and use equal inputs. The final estimate of the network is the outcome of the regression network that corresponds to the classified road condition. The architecture, displayed in Figure 2.4, does require the availability of a large amount of data points in all surface friction conditions. The measurements of the wheel speeds are converted to longitudinal slips before use as input to the different neural network estimators. The regression networks are based on a Nonlinear Auto-Regressive with Exogenous input (NARX) architecture. These models use current and past values of input data along with past estimates of the output.

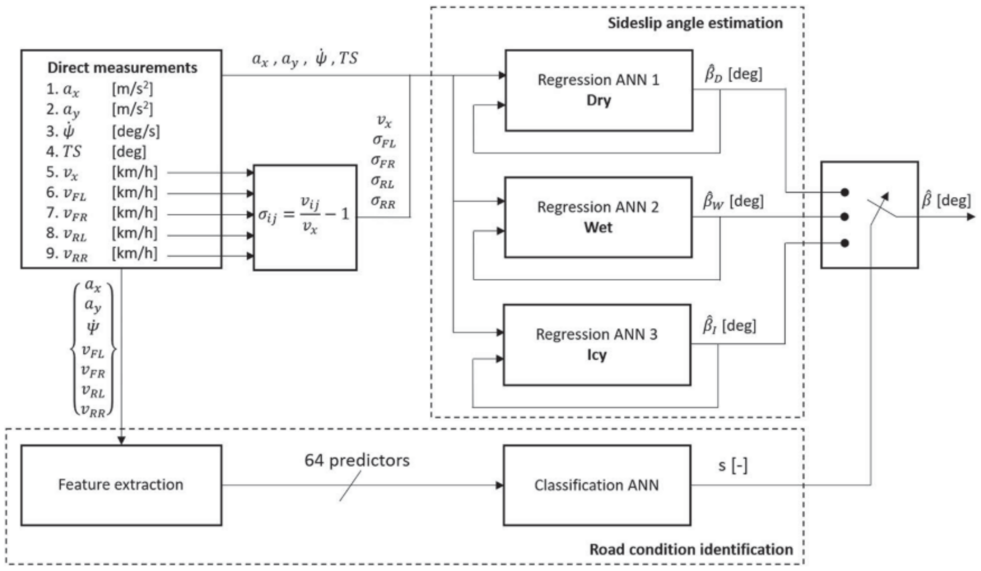

Figure 2.4: Estimation architecture proposed by Bonfitto et al. (2020)

Data-driven approaches are very useful since they do not rely on models and are therefore free from assumptions. However, for a data-driven model to behave correctly a large number of high quality data points has to be available. Furthermore, it is almost impossible to gain insight into the dynamics of the trained estimator.

2.3. Hybrid approach

Hybrid approaches aim to combine the advantages of both model-based and learning-based approaches. Using a neural network in combination with an observer can help with the robustness of the prediction task. Most hybrid approaches are constructed in a way such that the neural network estimates the tire forces or sideslip angle. These estimates are then fed in to the observer that provides a smoothed and corrected estimate of the VSA.

Acosta and Kanarachos (2018) propose a hybrid structure where different regression neural networks are combined with an EKF to estimate the lateral tire forces, longitudinal speed, lateral speed and yaw rate. The hybrid observer structure consists of three different neural networks for different levels of the surface friction coefficient. Again, the construction of three neural network estimators for different modes requires the data availability for all modes. These network estimators estimate the lateral tire forces. The neural network estimator consist of an ensemble of small neural networks with only 10 neurons in the single hidden layer. To estimate the friction coefficient a recursive least squares regression is applied. The final estimate for the lateral tire force is an interpolation between the outcomes of these networks weighted by the estimated friction coefficient.

Boada et al. (2016) adopt the ANFIS structure. This allows the implementation of fuzzy logic rules to make the VSA estimation more robust. The ANFIS network aims to combine the advantages of the neural network and fuzzy logic estimator. In the network if-then rules are learned that are able to capture nonlinear relations in an efficient manner. Using these if-then rules an estimator is created that is better adapted to variable environments and presents better generalisation capabilities. However, training the network properly requires a large amount of training data. An UKF is added for filtering and correcting the resulting VSA estimate.

Kim et al. (2020) use a deep ensemble based estimator where a time window of measurements is fed to two fully connected layers and a layer of LSTM cells. Next, two fully connected layers are in place, one to calculate an estimate of the VSA and the other to estimate the corresponding standard deviation. An ensemble estimator is constructed using 5 networks. Using such an ensemble estimator has advantages and disadvantages. An ensemble estimator can decrease the variance on the estimate. However, increasing the number of networks increases training time. The final estimate of the ensemble estimator is the mean of the 5 different predictions of sideslip angle. The respective uncertainty level of the estimation is calculated using the estimated standard deviations. The architecture of the hybrid model is presented in Figure 2.5.

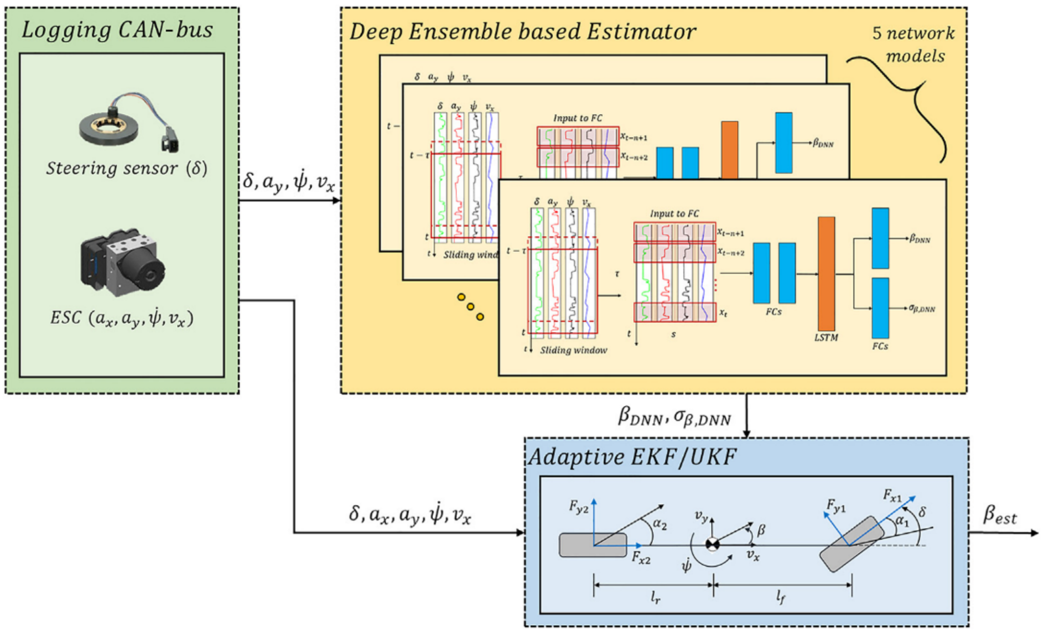

Figure 2.5: Estimation architecture proposed by Kim et al. (2020)

The paper of Ayyad et al. (2021) uses a Compression Recurrent Neural Network (C-RNN) for providing uncertainty estimates. The model is trained such that it compresses the training data distribution the best via the minimum description length principle. The aim of the estimation is to accurately identify out of distribution inputs. However, the model is unable to make accurate predictions for these inputs. Therefore, this structure is only useful for identification purposes and not for making predictions.

2.4. Summary

In this chapter several methods to estimate the VSA are discussed. This can be subdivided into observer-based, data-driven and hybrid approaches. Observer-based methods rely on an underlying vehicle model, which introduce corresponding assumptions. The UKF seems to be the preferred type of observer since it is able to capture high nonlinear relations more effectively. In literature the RNN is the preferred data-driven model architecture for estimating the VSA, since it is able to model temporal relations. The disadvantage of data-driven approaches is that they rely on a large amount of high quality data to prevent unwanted excitations. Various hybrid architectures are already evaluated in literature, using the neural network estimator as a sensor to the observer is the most common structure. A shortcoming of current literature is that the estimation models are trained and evaluated on either simulated data or a small experimental dataset.

The next chapter introduces the large-scale experimental dataset used in this study. Furthermore, it is discussed how this dataset is processed before it is used to train the data-driven models.

3 Experimental setup

Chapter 3 discusses how the experimental data is collected and processed. The data is collected during different standardised vehicle handling manoeuvres performed on testing grounds. These manoeuvres can be used to evaluate the vehicle dynamics in various conditions. An overview of the performed manoeuvres is presented, just as what data is being recorded. Naturally, experimental data has to be preprocessed before using it to develop data-driven models. It is discussed how the preproccesing procedure is constructed and what corresponding design choices are made.

3.1. Data collection

3.1.1. Testing manoeuvres

The different testing manoeuvres aim to provide insight regarding the VSA at different vehicle states. For example, manoeuvres are executed where the vehicle is driven in a circle with different radii and different velocities. This corresponds to different lateral accelerations at various longitudinal velocities in a steady-state. In contrast, using the lane change manoeuvre the transient behaviour of the vehicle can be examined. Using different sorts of testing manoeuvres can increase the robustness of the estimator. The different manoeuvres in the experimental dataset are discussed below.

Braking in turn (*ISO 7975*): During the braking in turn manoeuvre the vehicle is turning and braking at the same time. This allows evaluation of the VSA during high negative longitudinal accelerations. The manoeuvre is performed with various initial speeds and with ESC turned on and off.

Steady state cornering (*ISO 4138*): This manoeuvre consists of the vehicle driving in a circle at a certain longitudinal velocity. This can provide information regarding the steady-state behaviour of the vehicle at different levels of lateral acceleration. These tests are performed in both directions. The initial speeds differed as well as the radius of the circle the vehicle is following.

Hockenheim: These tests are not standardised handling manoeuvres, but various laps around a reproduced Hockenheim circuit at the Automotive Testing Papenburg facility. It can be very useful to look at the behaviour of the vehicle when it is driving at its limits on a track. The circuit is driven in both directions with ESC turned on and off.

J-turn (*ISO 7401*): During the J-turn manoeuvre the vehicle starts by driving straight with an initial velocity, at a moment the steering wheel angle is immediately changed to a set angle. This mimics a sudden steering manoeuvre where the aim is to quickly reach a high lateral acceleration. This manoeuvre is also performed with various initial velocities.

Double lane change (*ISO 3888-1*): The double lane change manoeuvre is a well-known manoeuvre to test the handling performance of a vehicle. The test consists of driving the vehicle from an entry lane to a side lane and then returning to the entry lane. The distance between the end of the entry lane and the beginning of the side lane is specified, just as the width of the lanes that depends on the width of the vehicle. The manoeuvre is performed at increasing initial velocities until the vehicle is not able to pass the test anymore.

Slalom: During the slalom manoeuvre the vehicle needs to slalom around pylons. This manoeuvre is used to evaluate the transient behaviour of the vehicle. Again, the initial speeds are increased until the vehicle is not able to pass through the pylons anymore.

Spiral: The beginning of the spiral manoeuvre is equal to the J-turn manoeuvre. However, after that a second turn is added at a moment when the vehicle is not yet stable. The spiral is executed in both directions at velocities of 50 and 100 km/h with the ESC turned on and off.

Random steering (*ISO 7401 - ISO 8726*): The random steering manoeuvre consists of applying sinusoidal steering inputs where the vehicle is expected to behave linearly. Various frequencies of steering inputs are evaluated during these tests.

Roll: During the roll manoeuvre the axial rotation of a vehicle towards the opposite side of the steering is mimicked. This is achieved by making short sudden steering to let the vehicle body roll over the chassis. It is expected that this manoeuvre does not create high sideslip angles.

3.1.2. Data logging

Within the testing vehicle a large number of states are recorded by a number of different sensors. These sensors included an Inertial Measurement Unit (IMU), Global Positioning System (GPS), steering angle sensor, four load sensing bearings, four wheel force transducers and an optical VSA sensor. The different sensors are discussed below.

GPS: The GPS is being used to determine the location of the vehicle. Since it is able to measure its position over time, it can also be used to determine the velocity and acceleration of the vehicle in the x- and y-direction.

IMU: The IMU can measure the accelerations and angular rate of the vehicle it is located in. These accelerations can be integrated to find the velocities, angle and the covered distance.

Steering angle sensor: The steering angle sensor is used to measure the steering angle at the steering wheel.

Optical sideslip angle sensor: The optical sideslip angle sensor is being used to find the 'true' VSA via an optical analysis. The Corrsys Datron is used for this task, which is a very expensive sensor that is not always robust. This makes it impossible to use commercially. The variance of the sensor is around 0.5 degrees.

Load sensing bearing: The load sensing bearing measures the bearing strains of the wheel bearing. These bearing strains can be used to reconstruct the wheel forces for the different wheels. There are four load sensing bearings available, one for every wheel.

Wheel force transducer: The wheel forces and moments are measured directly using Kistler wheel force transducers. These measurements do not require a mapping to determine the wheel forces and are expected to be more accurate than the mapping of the bearing strains. A wheel force transducer is also located at all wheels of the vehicle.

3.1.3. Data preprocessing

The used dataset consists of experimental data that needs to be preprocessed before use. The measurements to use as features for the sideslip angle estimation need to be selected. Furthermore, within the data measurement errors and measurement noise is present. This needs to be eliminated before using it to develop a model to achieve higher estimation performance.

Feature selection

To present an overview of the frequency each input variable is used in the literature, a bar plot is shown. In Figure 3.1 it is illustrated how many times each variable is used as an input to a data-driven model. In this analysis 13 research papers that focused on learning-based approaches have been analysed.

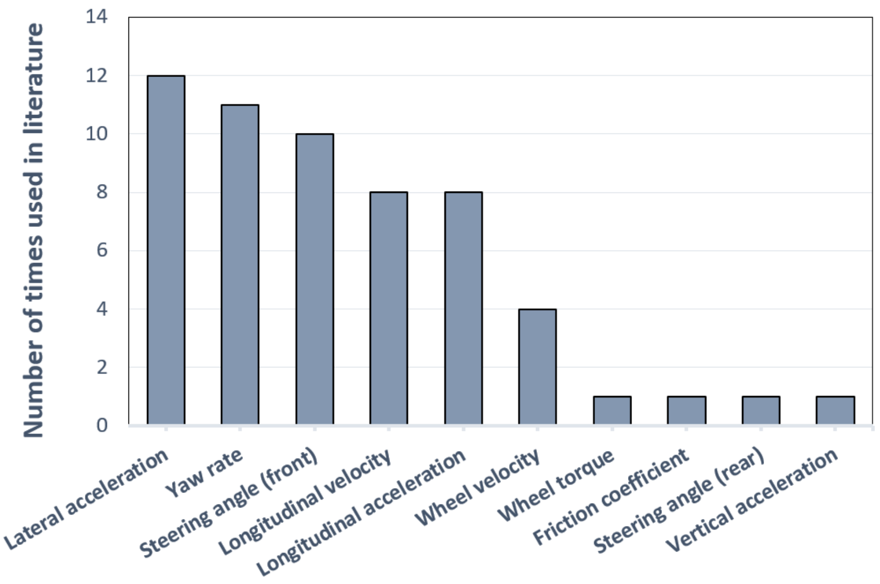

Figure 3.1: Feature frequency across 13 analysed research papers

Lateral acceleration is an input variable that is used consistently across the literature. This is to be expected since this variables is closely related to the VSA. However, there is a clear disadvantage to using the lateral acceleration as a predictor for the sideslip angle. A road bank angle would generate a component of lateral acceleration due to gravity. Melzi and Sabbioni (2011) indicate that when the bank angle is not included as a predictor, the value of lateral acceleration gets misunderstood by the network, compromising the quality of the estimate. They argue that adding manoeuvres with a bank angle should improve the robustness of the estimator, but the estimation quality may decrease. It is important to note that the same problem holds for a road slope, this angle would generate a component of longitudinal acceleration. This could degrade the performance of the neural network in a similar manner for the longitudinal acceleration feature.

The longitudinal velocity is another frequently used feature for VSA estimation. This state is assumed to be known, but this may not always be the case. When using a GPS sensor the velocity can be determined accurately, provided that a high quality receiver is used and that there is no signal blockage. However, in literature the longitudinal velocity is sometimes estimated using the combination of wheel velocities and longitudinal slip estimations from an observer. This estimation method can severely decrease the quality of the velocity estimate. In this study measurements from a GPS sensor are used.

In preliminary research there is investigated if adding the wheel forces or bearing strains has any effect on the performance of the data-driven estimator. To test this, three datasets with different features are created. In the first dataset only features that can be relatively easy be measured are included. In the second dataset also the wheel forces measured by the wheel force transducer in all three directions for all four wheels are included. The third dataset included six bearing strains for every wheel. An overview of the features in each dataset is presented in Table 3.1.

Dataset 1	Dataset 2	Dataset 3
Lateral acceleration	Lateral acceleration	Lateral acceleration
Longitudinal acceleration	Longitudinal acceleration	Longitudinal acceleration
Longitudinal velocity	Longitudinal velocity	Longitudinal velocity
Yaw rate	Yaw rate	Yaw rate
Steering angle	Steering angle	Steering angle
	Longitudinal wheel force (x 4)	Strain gauge #1 (x 4)
	Lateral wheel force (x 4)	Strain gauge #2 (x 4)
	Vertical wheel force (x 4)	Strain gauge #3 (x 4)
		Strain gauge #4 (x 4)
		Strain gauge #5 (x 4)
		Strain gauge #6 (x 4)

Table 3.1: Features different datasets

A Feedforward Neural Network (FFNN) is trained on the different datasets to evaluate the difference in performance. The results showed that adding the wheel forces or bearing strains increase the performance on the test set by decreasing the Root Mean Squared Error (RMSE) with respectively 9.3% and 13.2%. The precise results are presented in Table 3.2. Based on these preliminary results there is decided to use the bearing strains as input features for development of the different estimation models. In Appendix A the results can be found where the dataset containing the wheel forces is used to develop the hybrid model.

		Dataset 1	Dataset 2	Dataset 3
Training	RMSE	0.367	0.355	**0.322**
Validation	RMSE	0.354	0.300	**0.277**
Test	RMSE	0.354	0.321	**0.307**

Table 3.2: Performance of FFNN trained on different features evaluated by RMSE [deg]

Next to the bearing strains the five most frequently used input variables have been selected as features for the sideslip angle estimation. The bearing strains and additionally selected features are discussed individually below.

Lateral acceleration: The lateral acceleration in m/s^2 is measured by the GPS and IMU sensor. There is chosen to select the measurement of the IMU as the true acceleration. This choice is made since the GPS measurements must be differentiated twice to obtain accelerations, making it more error prone.

Longitudinal acceleration: The longitudinal acceleration in m/s^2 is measured by the GPS and IMU. The case is equal to that of the lateral acceleration and thus the measurement of the IMU is selected.

Longitudinal velocity: The longitudinal velocity in m/s is measured by the GPS and IMU. The IMU measurement must be integrated while the GPS measurement must be differentiated. Integration is more error prone and therefore the differentiated signal of the GPS is being used as the velocity measurement.

Steering wheel angle: The steering wheel angle is measured by the steering angle sensor in degrees. This angle is measured at the steering wheel and converted to the angles of the wheel using the known steering wheel ratio.

Yaw rate: The yaw rate in deg/s is measured by the IMU sensor. This quantity is measured directly by the IMU such that no integral or derivative has to be used.

Bearing strains: The wheel bearing strains are measured by load sensing bearings. These load sensing bearings are placed at all four wheels as to obtain all wheel forces in the vehicle. For each wheel there are six strains measured, these strains can be mapped to obtain the forces and moments present on the wheel. In literature these forces are reconstructed using a mapping from the bearing strains. The mapping coefficients are determined by a linear least squares regression between transformed bearing strains and wheel forces (Kerst et al., 2019). It is interesting to analyse if the neural network is able to identify the wheel forces in an equivalent manner.

Outlier removal and normalisation

Statistical outlier removal is used to remove extreme outliers. Note that removal is done cautiously to avoid deleting valuable scarce measurements of edge cases. Only data is used where the longitudinal velocity of the vehicle is higher than 5 m/s. Namely, below this velocity the sideslip angle measurements from the optical sensor are inaccurate. Aside from deleting single measurement points, the complete manoeuvre should be evaluated. During some tests control of the vehicle was lost and the vehicle spun. These manoeuvres are deleted from the testing set since the presence of such high sideslip angles in the dataset can have a negative effect on the performance of the estimator. Additionally, the aim of the study is not to estimate the VSA during spins of the vehicle.
Before the data is used for the development of the estimator, it must be normalised. Normalisation is required since all variables have a different physical interpretation and a different order of magnitude. Unscaled input variables can result in a slow or unstable learning process of machine learning models (Brownlee, 2018). There are two common methods of normalising the data, the first is to map the data onto the interval of [0,1] as displayed in Equation 3.1. Here $x_{min,i}$ denotes the minimal and $x_{max,i}$ the

maximal value of the respective feature.

$$x_{norm,i} = \frac{x_i - x_{min,i}}{x_{max,i} - x_{min,i}} \tag{3.1}$$

The second method of normalisation is to scale the data such that it has a mean of zero and a standard deviation of one. Every input variable is demeaned by subtracting the mean and divided by the standard deviation of the respective variable. The mean of each feature is denoted by $\bar{x}_i$ and the respective standard deviation by $\sigma_{x,i}$.

$$x_{norm,i} = \frac{x_i - \bar{x}_i}{\sigma_{x,i}} \tag{3.2}$$

There are no set rules for when to choose which normalisation mapping, selection is based upon trial and error. The normalisation mapping that results in better performance of the model is selected.

Data filtering

A visual inspection of the data revealed that there is a significant amount of noise present in the measured data. Therefore, a filter needs to be applied to remove most of this measurement noise. Especially the IMU measurements contain a large amount of noise. It is important to filter out the noise such that the network is trained with accurate measurements. The data is filtered using a lowpass filter with a cut-off frequency of 5 Hz. For the purpose of VSA estimation there is focused upon vehicle behaviour below these frequencies. For active stability functions the signals are typically filtered with a lowpass filter with a cut-off frequency around 5 Hz (Acosta and Kanarachos, 2018; Viehweger et al., 2021).
An Infinite Impulse Response (IIR) filter is used to design the lowpass filter. However, although the Finite Impulse Response (FIR) filter has a higher computational load, it is a better alternative. By implementing the IIR filter the first few measurements must be deleted from each manoeuvres. However, this small change has no significant effect on the performance. In contrast to the FIR, the IIR does not have a constant phase for all frequencies.

Offset removal

Another preprocessing step for the data is to eliminate the offset in the bearing strain measurements. The bearing strain values at an equal state differed between manoeuvres, this variation consisted of a certain offset that needs to be eliminated before accurate estimation can be performed. This offset is different for each day of data collection and even varies slightly between each manoeuvre. The presence of this variation in the data could therefore significantly impact the performance of the neural network estimator. To determine this offset the measurements of the LSB are averaged over the periods in the manoeuvre where the longitudinal and lateral acceleration of the vehicle are close to zero ($a_x \wedge a_y < 0.05 m/s^2$). At these time points the bearing strains are considered to be close to the respective offset. However, for some manoeuvres the vehicle is never in a non-accelerating state and thus the offset could not be determined. For these manoeuvres the average offset of that testing day is selected. These calculated offsets are then subtracted from the data of the respective manoeuvre to create a dataset without offsets. In this manner the bearing strains are suited to train an estimator on. It must be noted that this method is an approximation, but it produced the most promising results without knowledge of the exact offsets before each manoeuvre. It is also a method that is used in real applications for calibration purposes, which shows the validity of the approach.

Dataset characteristics

Next, an analysis of the available data is presented. When all the manoeuvres are combined there is a total of approximately 630.000 measurements available. The number of tests per manoeuvre is shown in Table 3.3. The length of the various manoeuvres differs, therefore the average length of each manoeuvre is presented.

It is interesting to analyse the distribution of the VSA measurements, which is shown in Figure 3.2. It must be noted that this is a log density plot and to provide intuition the percentiles of the distribution are highlighted. This provides insight regarding the operating region where the estimator will have a large amount of training data and where not. From this histogram there can be concluded that the data availability around a sideslip angle of zero is much more extensive than for sideslip angles with high absolute values, which is to be expected. With normal driving, only low sideslip angles are reached. The data is approximately symmetric with respect to positive and negative sideslip angles. High VSA measurements are really scarce in the dataset, this causes worse performance of the estimator in this operating region. Based on the number of available measurements there is expected that the performance of the data-driven estimator will approximately drop when the absolute value of the sideslip angle is larger

Manoeuvre	Number of tests	Average duration [s]
Total	190	27.7
Braking in turn	42	10.0
Circle	28	46.2
Hockenheim	7	146.2
J-turn	33	9.5
Double lane change	33	13.5
Slalom	15	23.2
Spiral	14	21.8
Random steering	8	115.4
Roll	10	18.9

Table 3.3: Size of dataset

than 6 degrees. Each bin in the histogram represents 1.5 degrees and an analysis showed that when the sideslip angle is between 6 and 7.5 degrees only 750 measurements are available. In contrast, when the sideslip angle is between 4.5 and 6 degrees there are still 4500 measurements available. Of course, experimental validation is requires to validate this statement.

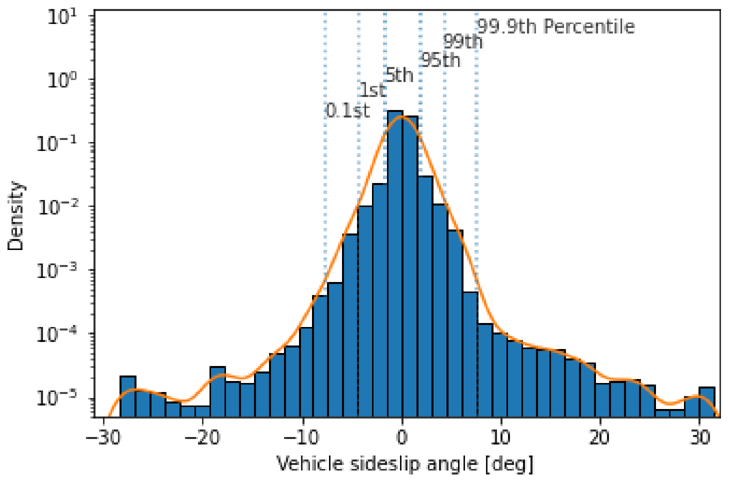

Figure 3.2: Density distribution of VSA measurements on a logarithmic scale

Data division

When modelling a system using neural networks, a labelled dataset is required. This dataset must be divided into a set to train and evaluate the model on. This is necessary to properly evaluate the performance of the model. If one would use the same data to train the models as well as to evaluate their performance, overfitting would occur. This means the model is only suited for the training data and has no ability to generalize. If so, the model will not be able to perform equally well outside the training set. Furthermore, the hyperparameters of the model must be tuned for optimal performance. The solution to this problem is to divide the available data up into three different sets: a training, validation, and testing set. Various methods to split the data into these different sets exist (Y. Xu and Goodacre, 2018).

The training set of the data is usually the largest subset of data and is used for model training. In other words, on this part of the data all the models with different hyperparameters are trained.
The validation set is the part of the data that is used for model selection and hyperparameter tuning. For example, to get the optimal performance from a neural network, one must determine the number of neurons in each hidden layer. This number of neurons is a data dependent hyperparameter. Models with a different number of neurons can be built on the training set to subsequently be evaluated on the validation set. The performance on the validation set is then pivotal in deciding the number of neurons.
The testing set is third set of data and should exclusively be used to evaluate the performance of the tuned model. It is important not to evaluate the model halfway through the tuning process on the testing data since this will lead to overfitting on the data. Also, the testing set must be independent from the training and validation set. In this way, the performance capability of the estimator outside the training data can be properly evaluated.

Manoeuvre	Train & Validation	Test
Braking in turn	40	2
Circle	26	3
Hockenheim	4	3
J-turn	28	5
Double lane change	29	4
Slalom	11	4
Spiral	13	1
Random steering	6	2
Roll	10	0

Table 3.4: Data division manoeuvres

For this study, the testing data set is handpicked with the aim to mimic the highest number of driving conditions. This is achieved by selecting manoeuvres where different handling characteristics are evaluated and the corresponding reached sideslip angles have a large range. This allows a proper evaluation of the generalization capabilities of the estimator. The remaining manoeuvres are divided into the training and validation set. The number of tests in the respective datasets are shown in Table 3.4. There is chosen for a 70%/15%/15% division between the training, validation and testing datasets. The data division resulted in a combined training and validation data set of 535.000 data points and a test set of 95.000 data points.

3.2. Evaluation metrics

To evaluate the performance of the different models, evaluation metrics must be selected. These metrics should be chosen in a way such that they are able to judge certain performance characteristics of the estimators. In the current setting it is useful to evaluate the overall variance of the estimation error, but also look at the maximal error that is reached. Namely, it is important to have an idea of the upper bound on the error. Therefore, the two used metrics are the RMSE and the Maximum Error (ME). These two metrics are commonly used in the literature and provide the opportunity to evaluate the variance as well as the maximum of the estimation error. The RMSE, a measure of the variance of the error deviation, is defined in Equation 3.3. The ME deviation is defined in Equation 3.4. The true value is denoted by y_i and the respective estimation is denoted by $\hat{y}_i$. The standard deviation of the error is denoted by σ_i.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{y}_i - y_i}{\sigma_i}\right)^2} \tag{3.3}$$

$$ME = \max(|\hat{y}_i - y_i|) \tag{3.4}$$

Next to these two well-known measures, another metric is introduced. The RMSE in the nonlinear region of the tire dynamics. It is known that the tires dynamics become nonlinearly at lateral accelerations of around 4 m/s^2. This nonlinear RMSE will compute the RMSE of the predictions where the absolute value of the lateral acceleration is larger than 4 m/s^2. Here, A denotes the set of measurements where the tires are behaving in the nonlinear region. The RMSE is then computed over this set of measurements.

$$A = \left\{ \left(\frac{\hat{y}_i - y}{\sigma_i}\right)^2 \middle| |a_{y,i}| > 4 \right\} \tag{3.5}$$

$$RMSE_{nl} = \sqrt{\frac{1}{|A|}\sum_{i=1}^{|A|} A} \tag{3.6}$$

3.3. Summary

This chapter describes how the experimental data is collected and processed. Various sensors are measuring different vehicle states when it is performing vehicle handling manoeuvres. These manoeuvres evaluate the vehicle behaviour in different operating regions. Wheel force transducers and LSBs are used to provide insight regarding the tire forces of the vehicle. This results in a large-scale experimental dataset where the outliers are filtered out via statistical outlier removal. It is apparent that a small number of high sideslip angle measurements are present in the dataset. The performance of the different methods is compared via the RMSE, the RMSE in the nonlinear tire region and the ME.
The used estimation architectures are discussed in the next chapter. The FFNN, Recurrent Neural Network (RNN) and Transformer are introduced as purely data-driven models. Also, current state of the art methods to estimate the uncertainty of the neural networks are highlighted. Finally the different observer variations and the hybrid model architecture is discussed.

4 Estimation architectures

This chapter introduces the different neural network architectures. Additionally, how these networks are trained and regularized is discussed. Next, the hybrid approach combining the neural network estimator with the observer is elucidated. Various observer variations are introduced accompanied by an analysis regarding their differences.

4.1. Feedforward neural network

Neural networks are very versatile models that can be applied to a large variety of tasks. The key to successfully applying such a modelling method is to design the network and tune the hyperparameters in a way that it is able to properly represent the actual system. More formally speaking, the goal is to create a neural network of which the effective capacity is equal to the complexity of the task.

One of the simplest neural network architectures is that of a FFNN. In such a network the information only moves forward and no memory is present in the network. Neurons are elementary units in a feedforward neural network. A neuron is a mathematical function with the form as presented in Equation 4.1, a visual representation is shown in Figure 4.1.

$$g(x_t; w) = \Phi\left(w_0 + \sum_{j=1}^{n} w_j x_{t,j}\right) \tag{4.1}$$

Here, w_0 denotes the bias and w_j denotes the weights corresponding to an input x_j. Within the function $\Phi(\cdot)$ only linear operations are performed. The function $\Phi(\cdot)$ is an activation function that maps the resulting value from the linear operations. One should note that the non-linearity within the neural networks solely originates from the activation function. Thus, when only linear activation functions are used, the resulting model is linear.

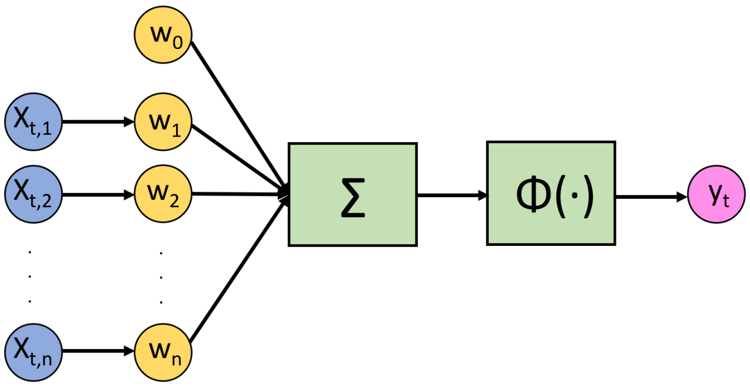

Figure 4.1: Artificial neuron visualised

Activation function

Examples of activation functions are the linear function, Rectified Linear Unit (ReLU) function, Sigmoid function, hyperbolic tangent function, Softsign function, and Softmax function. These functions can be divided into non-saturated and saturated functions. A saturated function is a function that flattens as it

approaches infinity, this implies that the derivative will approach zero. A drawback of using a saturated function as activation function is that when a neuron becomes saturated, in other words the input value of the neuron becomes very large in absolute value, it becomes very difficult to change the weights within this neuron. This is the case since the change of the weights is dependent on the derivative of the activation function at that point. Naturally, this change is slow when the derivative is close to zero.
The linear, ReLU and sigmoid activation function along with the derivatives are shown in Figures 4.2-4.4. It can be observed how the derivatives differ between the activation functions. The sigmoid function is an example of a saturated activation function.

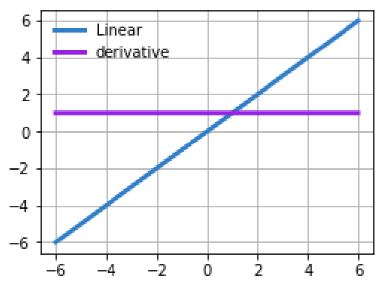

Figure 4.2: Linear Activation

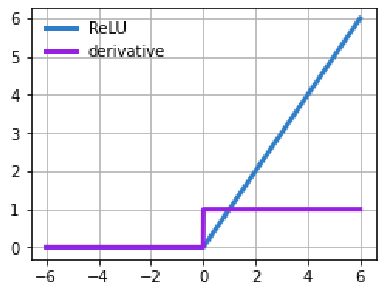

Figure 4.3: ReLU Activation

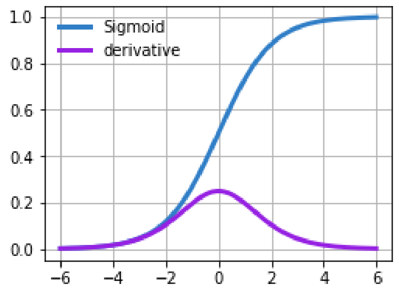

Figure 4.4: Sigmoid Activation

The choice for a certain activation function is very important when designing a neural network. There does not exist one superior activation function since the choice is problem dependent. Also, within a network different activation functions may be used in different layers. For example, the activation functions used in the hidden layers can differ from those in the output layer. This can especially be useful when the output must represent a classification, probability, or fraction. For a binary classification task the sign function can be used for the output layer. This will solely result in values of -1 and 1, which can be values corresponding to a certain class.

A typical feedforward neural network consists of three or more layers. The first layer is the input layer where the independent variables of the data point are filled in. Therefore, the number of neurons in the input layer will correspond to the number of input variables. The final layer is called the output layer, which presents the resulting dependent variables. Again, the number of neurons in the output layer will correspond to the number of outputs of the model. The layers between the input and the output layer are called hidden layers. Between subsequent layers there exist connections, the output of the previous layer is the input to the next layer. Within a feedforward network there do not exist connections between neurons in the same hidden layer. Since the neurons in the input layer do not receive values from previous layers, no computations are made within these neurons, these neurons simple indicate the distribution of the input variables over the neurons in the hidden layer. A shallow neural network is characterized by consisting of only one or two hidden layer(s) of neurons. These layers can consist of an arbitrary number of neurons. When the network consists of three or more hidden layers, one speaks of a deep neural network.

The goal of a neural network is to train the model such that it predicts the output values at a certain time point correctly based on the corresponding inputs. Thus, the weights must be adjusted in such a way that the model minimises the loss function. The loss function is used to score predictions to determine the performance of the model. These predictions should be close to the true measured value.

Weight learning

In single-layer neural networks the weight adjustment is straightforward since the loss is a direct function of the weights. In a multi-layer network the problem is more complicated since the loss is a composition of weights in earlier layers (Aggarwal, 2018).
The weights in the neural network can be updated via different methods, one well-known method is gradient descent. Gradient descent is a simple method that computes the updated weights via Equation 4.2. Here θ_t is the set of weights at time step t, α denotes the learning rate and $\nabla J(\theta_t)$ represents the gradient of the loss function with respect to the weights at the current weight values.
The backpropagation algorithm provides a way to compute the gradient of the cost function with respect to the weights and biases. These gradients can be used for stochastic gradient optimisation. The backpropagation algorithm is introduced by Rumelhart et al. (1986) and relies on two assumptions: the cost function can be written as an average over individual training examples and the cost function can be written as a function from the outputs of the neural network.

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t) \tag{4.2}$$

To explain the backpropagation algorithm the notation of Nielsen (2015) is followed. For a simple feedforward network with an arbitrary number of L layers, the backpropagation algorithm consists of the following steps. Note that in these steps the effect of a single training sample is examined.
The first step is to fill in the input data and compute the prediction of the estimator. Next, the error in the output layer is calculated (δ^L), the difference between the prediction and the true value. This output error has to be backpropagated through earlier layers in the network via Equation 4.3. The first term $(w^{l+1})^T$ 'moves' the error δ^{l+1} backwards through the linear part of the neuron. The term $\sigma'(z^l)$ measures how fast the activation function σ is changing at the evaluated point. This multiplication is performed using the Hadamard product denoted by $\odot$.

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{4.3}$$

When the backpropagated error is known for each layer, the gradient of the cost function can be found. The goal of the backpropagation algorithm is to obtain these gradients.

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{4.4}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \sigma(z_k^{l-1})\delta_j^l \tag{4.5}$$

Using these gradients it is possible to simply apply gradient descent to learn the parameters. A faster and more efficient method is to use batches of training data to calculate the gradients instead of using the complete dataset. With stochastic gradient descent the parameters are updated using the gradient of these batches. Optimization algorithms exist that can be used instead of stochastic gradient descent that aim to improve the training speed. Well known optimisation algorithms include RMSprop, Adagrad and Adam (Duchi et al., 2011; Hinton et al., 2012). The optimal optimisation algorithm is data dependent.

For 'learning' the weights in the model there has been made use of the Adam optimizer, developed by Kingma and Ba (2014). When training a network with a relatively simple structure this optimizer performs well. The name Adam is derived from 'adaptive momentum estimation', the method the optimiser uses to update the network weights. The first and second moment of the gradients are being used to speed up training using the momentum principle. The Adam optimiser contains several hyperparameters that need to be tuned such as the learning rate and various decay rates.

Batch normalisation

The training of deeper neural networks is complicated by the fact that the distribution of the layer inputs is affected by the parameter values of the preceding layers. Changes in the parameter values of preceding layers can cause significant changes of the input distribution. Therefore, the parameter values in this layer must adapt for the change in distribution, in turn this causes difficulties for the next layer. This effect propagates through the layers and becomes a considerable problem in models that contain many layers, in other words, deep networks. It is especially problematic for deep networks when saturated activation functions are being used. This constant adaptation, known as internal covariate shift, causes more training steps to be required.

Batch normalisation is a method introduced by Ioffe and Szegedy (2015) to decrease the required training time for deep neural networks. The batch normalisation layer is used to normalise the output between the hidden layers, in this manner the internal covariate shift is reduced. It is most effective if the normalisation takes place between the sum of the neuron and the activation function. Batch normalisation uses, as the name suggests, batches for the normalisation of the layer inputs. It is computationally expensive to use the complete data set for normalisation, since all data must be passed through the neural network at every backpropagation step. In stochastic optimisation the use of batches decreases the computational load and also adds regularisation to the network. Every weight updating step the mean and variance of the respective batch are calculated using the equations below.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{4.6}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{4.7}$$

The inputs are normalised by using these values for the mean and variance. However, normalisation of the inputs to a layer is very constraining and could restrict the inputs to the linear region of a nonlinear activation function. In such cases the neural network model is unable to model nonlinear relations. To avoid this, a scaling and shift are in place. This is achieved using learned γ and β parameters. Every time step the values of γ and β are learned using backpropagation of the gradient of the loss. Since batches are used, it can be suboptimal to use these values if a very skewed batch is selected. Therefore, the selected parameters γ and β are computed by an exponential moving average over its previous values.

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{4.8}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{4.9}$$

Recent literature has argued that batch normalisation does not solve the problem of internal covariate shift, but rather significantly smooths the optimisation landscape (Santurkar et al., 2018). This smoothness also improves training due to better performance of the optimisers on smoother surfaces, but exemplifies the lack of knowledge in the field of training neural networks.

Weight initialisation

When the size of a neural network grows, the initialisation of the weights becomes more important. The initial point can determine if the algorithm converges or not, unfortunately a lot is unknown about initialisation (Goodfellow et al., 2016). There are various methods of initialisation that aim to improve the training speed. Glorot and Bengio (2010) propose an initialiser that aims to satisfy the objectives of maintaining activation variances and back-propagated gradients variance as one moves up or down the network. The variance of the uniform distribution then depends on the number of nodes (n) in the current and next hidden layer. The exact specification of the initialisation distribution is presented in Equation 4.10.

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] \tag{4.10}$$

When the initialisation of the model is not close to a good solution, the optimisation algorithm can get stuck in a local minimum in this vicinity, which is called premature convergence. In this case, the model has a high error on the training and validation set. The choice of initialisation leads to a form of stochasticity in the training of the model since each initialisation can result in a different solution.

Hyperparameter tuning

Hyperparameter tuning is essential for the performance of the neural network. By adjusting the hyperparameters the effective capacity of the model is changed with the aim to match it to the complexity of the task. There are two approaches to tune the hyperparameters in a neural network, manual tuning by the user or automatic tuning using an algorithm.

Manual search: Manual search is the most straightforward approach and consists of simple trial and error. Different hyperparameters sets are chosen and the estimator is trained using the selected set. To evaluate the choice a certain performance metric is evaluated such as the RMSE on the validation set. The lower the RMSE on the test set, the better the selected set of hyperparameters.

Grid search: All different combinations of hyperparameters in the selected ranges can also be evaluated, this is called grid search. It must be noted that this is very inefficient. The required number of evaluations scales exponentially with the number of hyperparameters to tune.

Random search: Grid search quickly becomes infeasible due to the high number of required evaluations. Bergstra and Bengio (2012) show that for tuning neural networks random search performs much better than pure grid search over the same domain. Random search is able to find models that are as good as or better within a fraction of the computational time. To decide which hyperparameters should be selected, again the RMSE on the validation set is evaluated. To perform the random search the user must indicate what hyperparameters to vary and over what range.

Sequential model-based optimization: Another method for hyperparameter tuning is sequential model-based optimization. Hutter et al. (2011) introduced this technique for problems where evaluation fitness function, in this case the calculation of the RMSE on the validation set, is computationally expensive. The methods employs Bayesian optimization for finding the next set of hyperparameters to evaluate. The concept of Bayesian optimization is to reduce the number of times the function has to be evaluated by only evaluating the most promising sets of hyperparameters. This method has the advantage over grid search and random search that it does not evaluate hyperparameters in the vicinity of already evaluated hyperparameter sets with low performance.
The Bayesian optimization uses a surrogate function, which is a probability representation of the objective function built from previous evaluations. The next set of hyperparameters that needs to be evaluated is selected by maximizing the expected improvement with respect to the set of hyperparameters. In other words, this means finding the best set of hyperparameters under the surrogate function.

4.2. Recurrent neural network

A Recurrent Neural Network (RNN) is a special type of network architecture that allows the modelling of temporal dependence. This special architecture allows the input at a certain time step to interact with the hidden state of previous time steps. This way of being able to model temporal dependence over data points can be very useful for modelling the VSA. Due to the transient response of the vehicle, the accelerations and yaw moment of a few timesteps ago will still have an effect on the current sideslip angle (Rajamani, 2011).

RNNs are a class of neural networks that have been extensively used in the areas of speech recognition and natural language processing (Graves et al., 2013; Peng et al., 2015). These network architectures are useful for these tasks since they have a 'memory', namely current inputs influence the outcome a few steps ahead in time. For example, this is useful in speech recognition tasks where words can only be recognized if a sequence of tones is entered into the model.
By adjusting the weights of the connections between neurons of consecutive time points, one can adjust the influence of the temporal dependence. When these weights are set to zero, one obtains a feedforward neural network. If high weights are used, previous states have a large effect on the current outcome. A visual representation of the architecture is presented in Figure 4.5.

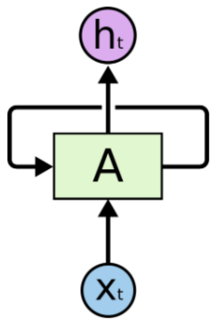

Figure 4.5: Representation of a simple recurrent neural network. The state of the current time step influences the output of the next time step. Reprinted from Colah's blog, by C. Olah, 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanishing/Exploding gradients

The primary challenge when training RNNs is the vanishing/exploding gradient problem. The RNN model is trained by updating the coefficients using the backpropagation through time algorithm. This algorithm is very similar to the backpropagation algorithm, but suited to networks with a recurrent structure. The algorithm calculates the gradient of the weights in the same way as the backpropagation algorithm as shown in Equation 4.5. Rewriting the backpropagation formulas, Equation 4.11 is obtained. Suppose a RNN is trained to estimate a certain parameter that heavily depends on a certain input of a large number of timesteps ago, in other words there is a delay in the response of a few timesteps. When the heavily correlated input is a large number of time steps apart from the output, this dependence is difficult to capture. This is caused by the multiplicative nature of the weight updating equation over time.

$$\frac{\partial C}{\partial h^l} = c\sigma'(h^{l+1})\frac{\partial C}{\partial h^{l+1}} \tag{4.11}$$

Namely, upon inspection of Equation 4.11 one can observe that the partial derivative used for updating the weights is dependent on the derivative of the activation function. For most activation functions this derivative is lower than one for all values of h^{l+1}, which denotes the loss in a certain hidden layer. Furthermore, the shown equation is iterative over the time steps. Suppose that $\sigma'(h^{l+1}) = 0.3$, when trying to update the weights 10 time steps ago the magnitude of the gradient update will drop to $0.3^{10} \approx 6 * 10^{-6}$ of the original values. Therefore, many updates are required to significantly change this parameter.
When the computed gradient is smaller than one, over time the update gradient will converge to zero, this is called the vanishing gradient problem. When the computed gradient is larger than one, the gradient will grow to infinity, respectively the exploding gradient problem. There are methods to decrease the impact of this problem by using modified backpropagation algorithms or gradient clipping (Pascanu et al., 2013). However, often other recurrent cells are selected for its greater capability of modelling longer temporal dependencies.
It is important to note that the vanishing/exploding gradient problem also occurs when trying to train deep networks. The same multiplicative nature of gradients can be found when trying to propagate the error through different layers of the network.

Long Short-Term Memory cell

As discussed, RNNs are useful when one wants to use past information for a present prediction. When this information is recent (few time steps ago), the discussed RNN structure is useful. However, this structure is not able to model long-term dependencies. To learn long-term dependencies a new kind of cell is developed, the Long Short-Term Memory (LSTM) cell (Hochreiter and Schmidhuber, 1997).

The LSTM cell is capable of learning long-term dependencies due to the presence of a cell state that is passed through time. The cell state is represented by the top horizontal line in Figure 4.6. It allows information to pass through time freely.

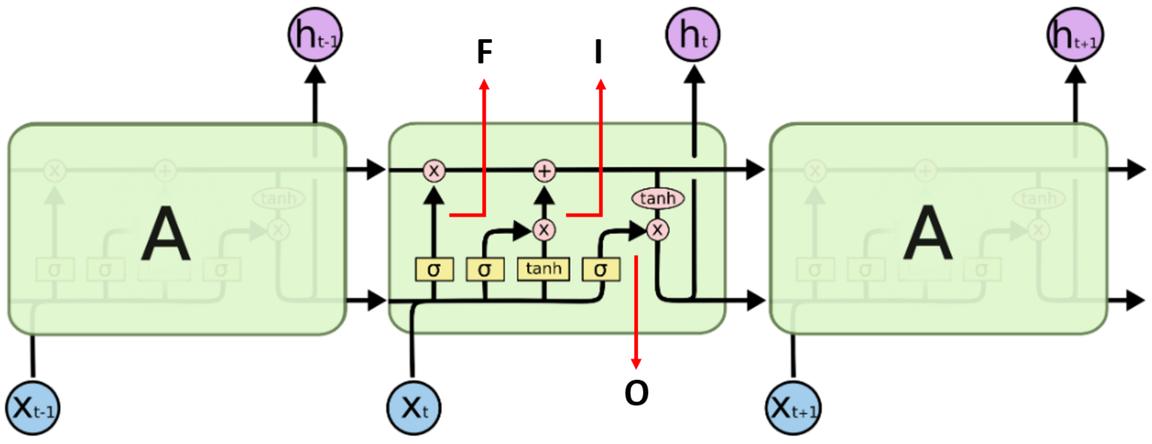

Figure 4.6: Structure of the LSTM cell. Note that σ represents the sigmoid function and tanh represents the hyperbolic tangent function. Reprinted from Colah's blog, by C. Olah, 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Within the LSTM cell several gates exist, these are discussed from left to right as they can be observed in Figure 4.6. The first gate is the 'forget gate' indicated by F, this gate looks at the previous hidden state (h_{t-1}) and current data point (x_t) and determines what data to remove from the cell state via Equation 4.12.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{4.12}$$

The next gate is called the 'input gate', indicated by I in Figure 4.6. This gate decides what data gets added to the cell state. This gate again makes use of the previous hidden state (h_{t-1}) and current data point (x_t). The variable i_t is a scaling factor that indicates how much each part of the state must be updated. The new cell state (C_t) is composed of the old cell state (C_{t-1}) multiplied by the forget state (f_t) to 'delete' this invaluable data. Additionally, the newly computed cell state ($\tilde{C}_t$) times the scaling factor (i_t) is added to the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4.13}$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{4.14}$$
$$C_t = f_t C_{t-1} + i_t \tilde{C}_t \tag{4.15}$$

The last step is to determine the output signal of the LSTM cell. The respective section is indicated by O in Figure 4.6. This output signal is composed of a combination of information retrieved from the updated cell state (C_t) and information from the data point (o_t).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{4.16}$$
$$h_t = o_t \tanh(C_t) \tag{4.17}$$

There also exist RNN structures that are bidirectional. In the current study there is chosen not to evaluate such architectures as the model is developed for online estimation. Thus, the data of future time steps is unknown and a bidirectional structure is not useful.

Gated Recurrent Unit

There exist multiple variants of the LSTM cell such as the Gated Recurrent Unit (GRU). This unit combines the 'forget gate' and 'input gate' to a single 'update gate' (Cho et al., 2014). Therefore, the GRU is less complex than the LSTM cell and less weights need to be 'learned'. When the dataset is small the choice can be made to use the GRU instead of the LSTM cell. This simplification of the GRU has the consequence that the memory content (hidden state) is exposed at each step without any control. In contrast, the LSTM has the ability to only show a part of the memory content using the output gate.
Another difference is the location of the input gate or respectively the reset gate. This different location changes the dynamics of how information is added to the memory. In the LSTM the amount of information added to the memory is independent of the forget gate. However, the GRU controls the flow of the memory in the previous step, but does not control the amount of added information to the memory. An overview of the GRU is presented in Figure 4.7.

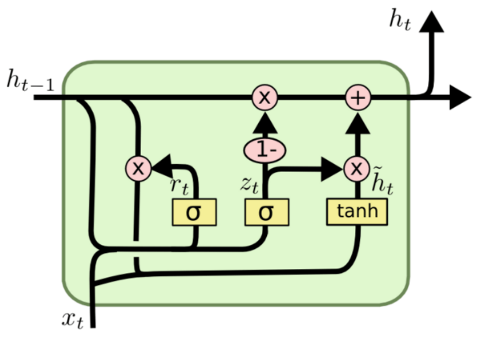

Figure 4.7: Structure of the GRU. Reprinted from Primo.ai, 2020, https://primo.ai/index.php?title=Gated_Recurrent_Unit_(GRU)

Chung et al. (2014) evaluated the performance of the GRU versus the LSTM cell on different tasks and could not make a concrete conclusion about which of the gating units performed better. In their research they found that it is very dependent on the task.

4.3. Transformer

RNNs are very popular for dealing with sequential data, but are accompanied by a few problems. Firstly, information is lost when the RNN is trying to process long data sequences. This is especially true for vanilla RNN structures due to the vanishing/exploding gradient problem, but also holds in a lesser degree for LSTM and GRU cells. Secondly, RNNs must process the data sequentially and since the next cell depends on the state information of the previous cell, RNNs are unsuited for parallel computing. This causes longer training and prediction times.

Transformers are a relatively new approach in deep learning that aim to solve these problems. Transformers are introduced by Vashwani et al. (2017) for the task of Natural Language Processing (NLP) and are based on the principle of attention. The Transformer makes it possible to achieve sequence to sequence modelling without the use of recurrent network units. If use of the Transformer for the task of VSA estimation is advantageous needs to be evaluated. Also, due to computational complexity it has to be investigated if real time application of the architecture is feasible.

Attention

Attention is a principle that humans use every day without being aware of it. To introduce the concept it is useful to present an example. Imagine that you lost your red water bottle in a room and are searching for it by looking across the room. When searching you explicitly focus on objects that appear to be red, in other words you pay attention to red objects. This is natural since this is a feature of your water bottle. This attention is very useful since you do not have to pay attention to objects of a different colour since this cannot be your water bottle. Machine learning tries to exploit this attention mechanism to increase the performance of the neural network.

The principle of attention is introduced in deep learning for the purpose of translating large texts. The main goal of attention in this context is to establish connections between related parts of the text. When related parts of the text are further apart, information can get lost when using RNNs due to their sequential structure. However, due to the attention principle Transformers do not suffer from this distance due to the made connections.

Transformers make use of self-attention, a mechanism using keys, values and queries. It is useful to explain these concepts using the process of language translation. In this situation the keys and values are the memory of what words the model has seen before and the query is the current word that needs to be processed. The query is compared against the keys and based on the similarity a score is given. These scores are normalized and multiplied by the values to obtain an attention value. The exact mathematical formulation is presented in Equation 4.18 where Q, K and V indicate respectively the queries, keys and values. Here, the Softmax function creates a probability function over the keys with peaks at similar keys. This distribution is multiplied by the values for the final score.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{n}}\right)V \tag{4.18}$$

The Transformer in the paper of Vashwani et al. uses multi-head self-attention, essentially an ensemble method. The attention function is not computed once, but multiple times in parallel. Thus, the attention score is calculated multiple times using what the authors call 'different representation subspaces at different positions'. The attention scores are averaged out to form a final score.

Architecture

The Transformer is constructed in an encoder-decoder structure. This essentially means that the input is first encoded into a feature subspace, after which it is decoded to the output. For example, with language translation the encoder converts the input sentence into a language independent feature space. Subsequently, the decoder translates this to the new language. Each language will have a specific encoder and decoder. The encoder must be specific for the input language, the decoder determines the output language. When a decoder for the French language is used, the output is French, when a decoder for the Dutch language is used, the output is Dutch.

A full overview of the Transformer model architecture is presented in Figure 4.8. The part of the Transformer that acts as the encoder in outlined by a red box, where the decoder is outlined by the blue box. The multi-head attention block, which is discussed in the attention section, is part of the encoder and of the decoder. For the task of NLP the authors use a total of 6 encoders and 6 decoders, these are placed in a sequential manner.

Positional encoding

Before the input reaches the encoder it is processed by learned embedding layers such that the dimension of the data is suited for the encoder. Furthermore, to preserve positional information of the sequence, a sinusoid-wave-based positional encoding is applied and summed with the embedding output. At last, after the final decoder layer a learned linear and Softmax layer are applied.
The Transformer is trained by feeding a sentence in one language to the Transformer via the input and the translation of the sentence to the output. It must be noted that the translation of the sentence is shifted

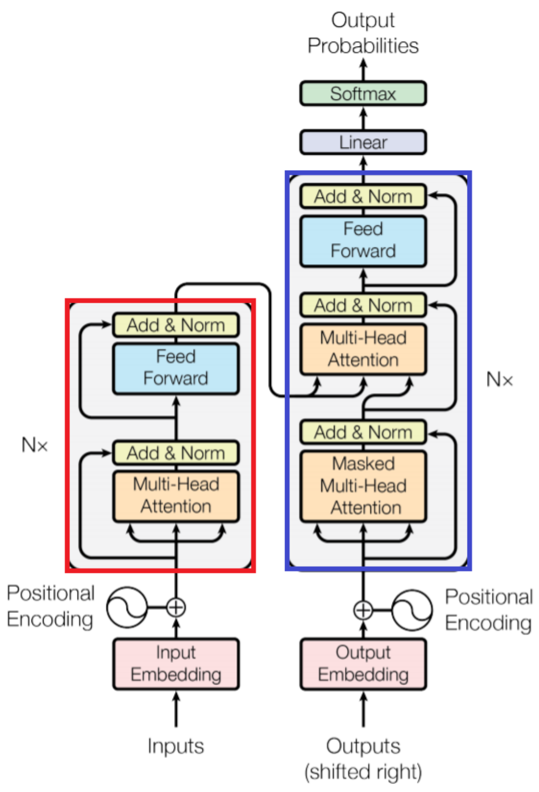

Figure 4.8: Transformer architecture proposed by Vashwani et al. (2017)

right by one position, this is to avoid the decoder from simply copying the decoder input. By shifting the decoder input one position to the right it is asked of the model to predict the next word, when the prediction is wrong the model immediately gets corrected using the shifted decoder input. This method of training a model is also known as teacher forcing, which helps to train the model faster. When the sentence is entered into the decoder input it must start with a start-of-sentence token and it must end with an end-of-sentence token.

Inferring using Transformers is different from that of more classical predictors. To translate a sentence it needs to be entered into the encoder input. In the decoder a start-of-sentence token has to be filled in. The Transformer will then output a single element which needs to be added to the decoder input. The Transformer then runs again predicting the second element, which again must be added to the decoder input. These steps have to be repeated until the Transformer predicts an end-of-sequence token, at this point the full encoder sequence is translated. It can be seen that multiple runs through the encoder are required to translate a sentence.

From the architecture one can gather that the Transformer has a complex structure with a large amount of weights that have to be ’learned’. In general, when more weights need to be ’learned’, more data has to be available and training of the network will take longer. For NLP tasks a lot of training data is available and the network can be pretrained. If the weights of the same model for a different application are available one can make use of transfer learning, which can severely shorten the required training time.

Time series application

The Transformer architecture is originally designed to be used for NLP, a task that is completely different from time series estimation. Thus, it is unclear if the architecture can be used for this application. To start, when sentences are used as an input to the Transformer they are tokenized. This means they are translated into integers according to a given vocabulary V. Of course, floats as inputs of the time series data cannot be tokenized in such a way. To solve this problem a new input and output embedding has to be designed that is able to deal with float inputs.
Next, in the NLP case a collection of superimposed sinusoidal functions are added to each input embedding. When processing time series this is not viable since the inputs are floats instead of distinct tokens. Kazemi et al. (2019) offer a solution to this problem by introducing the Time2Vec representation for time. This method encodes measurements taken close to each other in time with similar values, which allows the Transformer to spot that the measurements happened close after one another. Also, measurements

from the same hour as previous days or one the same weekday get encoded with a higher similarity. This allows the Transformer to also model daily, weekly and seasonal fluctuations. It must be noted that the measurements must come from the same timeline to make it possible to model such fluctuations. When the system is independent of time and not all measurements are consecutive, this encoding is not useful. If so, the independent sections of measurements must be positionally encoded separately.

Researchers have already evaluated the performance of the Transformer when used for forecasting. Wu et al. (2020) used the Transformer for forecasting the influenca-like illness activity. These time series are highly influenced by seasonal effects. The Transformer shows great promise by outperforming ARIMA, LSTM and Seq2Seq models. As discussed, it is important that enough data is available to train the Transformer. The track record of the Transformer is very good and evaluation of the performance can be very interesting.

If the size of the available training data is low or a decoder is not necessary, the decoder can be discarded. A decoder may not be necessary when the complexity of the task is low or with a small number of output parameters. It is also possible to replace the decoder with a fully connected layer. When discarding the decoder, inference does not have to be performed using teacher forcing and the model can be trained in a traditional manner. Additionally, when training a model with many weights it is always useful to start with pretrained weights. It does not matter for what task these weights are used since pretraining will almost always reduce training time significantly.

4.4. Hybrid approach

In literature hybrid structure models achieve the highest performance. These structures combine data-driven with observer-based models to provide a higher quality sideslip angle estimation. Different structures exist with the aim of achieving the highest performance. The most frequently used structure consists of a data-driven model estimating a sideslip angle along with the respective uncertainty level. The observer uses these two estimates as a sensor measurement and a corresponding amount of noise. There are many design choices to be made when constructing the hybrid model. The building blocks of the hybrid structure are discussed in this section. Choices must be made regarding the type of observer model to use, the method of calculating uncertainty of a data-driven model, and the overall structure of the model.

4.4.1. Extended Kalman filter

The first observer to discuss is the EKF, an extension of the well-known Kalman filter. The Kalman filter is an optimal estimator, but can only be applied to linear systems. Since the dynamics that need to be modelled here are nonlinear, this observer cannot be applied.

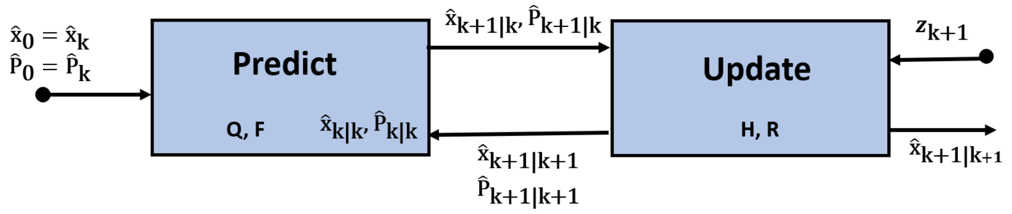

Figure 4.9: Schematic representation EKF

To understand the EKF observer the large similarities with the linear Kalman filter are used. The variables that one attempts to track are located in the state vector x. The covariance matrix P denotes the (co)variances of the different states in x, a measure of the uncertainty of the estimation. The Kalman filter consists of two phases, the prediction and update phase. During the prediction phase the next conditional state is predicted using the transition matrix F. Additionally, the covariance matrix for the next time step is computed using the transition matrix and the process noise. The discrete linear Kalman filter described here operates in discrete time steps. The subscripts in the equations denotes the time step of the respective state or covariance matrix and conditional on what time step this state or covariance matrix is estimated. For example $\hat{P}_{k+1|k}$ denotes that the covariance matrix at time $k + 1$ is estimated conditionally on the information present at timestep k. During the predict phase only information about time step k is present, no measurements from time step $k + 1$ are available.

$$\hat{x}_{k+1|k} = F\hat{x}_{k|k} \tag{4.19}$$
$$\hat{P}_{k+1|k} = F\hat{P}_{k|k}F^T + Q \tag{4.20}$$

After the measurements at time step $k+1$ are known, the update phase follows. The measurements are contained in the observation vector z. This observation vector is used to determine how accurate the conditional state $\hat{x}_{k+1|k}$ is by computing the residual vector y, the error between the observation and the conditional observation. The conditional observation is calculated by multiplication of the measurement function H with the conditional state.

$$y_{k+1|k+1} = z_{k+1} - Hx_{k+1|k} \tag{4.21}$$

Next, the Kalman gain K has to be calculated, which is dependent on the system uncertainty S. The system uncertainty is calculated using the covariance matrix P and the measurement noise R. The measurement noise R is an indication of the noise level on the measurements in the observation vector z.

$$S = H\hat{P}_{k+1|k}H^T + R \tag{4.22}$$
$$K = \hat{P}_{k+1|k}H^TS^{-1} \tag{4.23}$$

Then, the final prediction of the state and covariance matrix conditional on the most recent information can be made. A visual representation of the EKF is presented in Figure 4.9.

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + Ky_{k+1|k+1} \tag{4.24}$$
$$\hat{P}_{k+1|k+1} = (I - KH)\hat{P}_{k+1|k} \tag{4.25}$$

As mentioned, the EKF is a nonlinear extension to the Kalman filter that linearizes around the current estimate. The dynamics in the discussed Kalman filter are linear, which allows a linear transformation between the state and next conditional state. However, in nonlinear systems this linear transformation does not exist and the EKF compensates for this by using the first order derivatives for these transformations. To achieve this, the matrices F and H are replaced by the partial derivative of these functions evaluated at the current state.

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k|k}} \tag{4.26}$$
$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k+1|k}} \tag{4.27}$$

To use the EKF the matrices F and H have to be derived from the system dynamics. The matrix R can be determined via evaluation of the measurement noise and can otherwise be tuned. Also, the matrix Q must be tuned for optimal performance.

4.4.2. Unscented Kalman filter

The EKF has a flaw that decreases the performance significantly. For a linear system the Kalman filter is able to propagate the Gaussian random variable through the system dynamics analytically. However, for nonlinear system dynamics the state distribution is approximated by a Gaussian random variable that is propagated through the first order linearization of the nonlinear system. Higher order terms are ignored, which can lead to suboptimal performance and divergence of the filter.

In the UKF, proposed by Julier and Uhlmann (1997), the state distribution is also represented by a Gaussian random variable, but is identified using a minimal set of sample points. These carefully chosen sample points are named 'sigma-points'. These sigma-points are selected from the source Gaussian and are mapped using the nonlinear dynamics on the target Gaussian. These points are then being used to calculate the actual mean and variance of the transformed Gaussian. The sigma-points are located around the mean in directions based on the covariance matrix of the system.

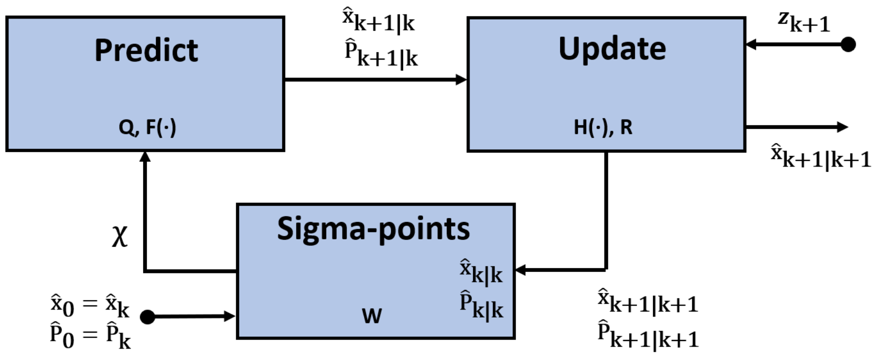

Figure 4.10: Schematic representation UKF

A number of $2N$ sigma-points is required in a system with N dimensions. The exact locations of the sigma-points are computed using the following formulas, here μ denotes the mean of the source Gaussian, λ is a scaling parameter for how far the sigma points need to be located from the mean and Σ is the covariance matrix of the source Gaussian.

$$\mathcal{X}^{[0]} = \mu \tag{4.28}$$

$$\mathcal{X}^{[i]} = \mu + (\sqrt{(N+\lambda)\Sigma})_i \qquad \text{for i = 1,...N} \tag{4.29}$$

$$\mathcal{X}^{[i]} = \mu - (\sqrt{(N+\lambda)\Sigma})_{i-n} \qquad \text{for i = N+1,...2N} \tag{4.30}$$

The different sigma points also have weights assigned to them. These weights are based on the scaling factor and sum to 1. The mean has a weight that grows when the scaling factor becomes larger, thus when the sigma-points are located further from the mean in the source Gaussian, the mean is higher weighted. The weights are placed in such a way that results in approximations up to the third order for Gaussian inputs (Wan and Van Der Merwe, 2000). When non-Gaussian inputs are used, the approximations are accurate to the second order. The accuracy can be up to the third and fourth order, but this is determined by the choice for α and β.

$$w^{[0]} = \frac{\lambda}{n+\lambda} \tag{4.31}$$

$$w^{[i]} = \frac{1}{2(n+\lambda)} \qquad \text{for i = 1,...,2N} \tag{4.32}$$

The next step is to propagate the sigma-points through the nonlinear system. The mean and variance of the target Gaussian can easily be computed using the weights and transformed sigma-points. In these equations $f(\cdot)$ denotes the nonlinear system. Furthermore, the process noise Q must be added to the covariance matrix to account for unmodeled noise in the model.

$$\mu' = \Sigma_{i=0}^{2N} w^{[i]} f(\mathcal{X}^{[i]}) \tag{4.33}$$

$$\Sigma' = \Sigma_{i=0}^{2N} w^{[i]} (f(\mathcal{X}^{[i]}) - \mu')(f(\mathcal{X}^{[i]}) - \mu')^T + Q \tag{4.34}$$

For the update step the calculated sigma-points are used to determine the expected outcome of the system using the nonlinear measurement function $h(\cdot)$. This expected outcome is used to calculate the covariance of the system in the measurement space. This covariance matrix contains the respective measurement noise R_t.

$$\mathcal{Z} = h(\mathcal{X}) \tag{4.35}$$

$$\hat{z} = \Sigma_{i=0}^{2N} w^{[i]} \mathcal{Z}^{[i]} \tag{4.36}$$

$$S = \Sigma_{i=0}^{2N} w^{[i]} (\mathcal{Z}^{[i]} - \hat{z})(\mathcal{Z}^{[i]} - \hat{z})^T + R_t \tag{4.37}$$

The last step is to calculate the Kalman gain, in contrary to the EKF the Jacobian is not used for this. Instead, the cross-correlation between the sigma-points in the state space and sigma-points in the measurement space is calculated. This term is very similar to the $\hat{P}H^T$ being used in the EKF. The Kalman gain is then calculated by multiplying this with the predicted covariance matrix.

$$T = \Sigma_{i=0}^{2N} w^{[i]}(\mathcal{X}^{[i]}) - \mu')(\mathcal{Z}^{[i]} - \hat{z})^T \quad (4.38)$$
$$K = TS^{-1} \quad (4.39)$$

The final states are then predicted using almost equivalent formulas as those used in the UKF. A schematic overview of the UKF is presented in Figure 4.10.

$$\mu = \mu' + K(z - \hat{z}) \quad (4.40)$$
$$\Sigma = (I - KT)\Sigma' \quad (4.41)$$

Using the UKF one is able to capture up to the third order Taylor series terms, a large improvement over the EKF. Additionally, the UKF does not depend upon derivatives of the system dynamics. This can be useful when these dynamics are unknown.

4.4.3. Confidence level estimation

For an observer to operate in an optimal way, estimations and the respective uncertainty level must be provided to estimate the state. In this study neural networks are being used for predictions, these networks do not present the uncertainty of an estimation by default. However, in recent literature more attention has been paid towards estimating the uncertainty of neural network estimations. Several methods that are viable in the current setup are discussed.

Deep ensemble estimation

The most straightforward manner of estimating the uncertainty level of a neural network estimation is to train multiple models and letting these models predict independently. The different predictions can then be combined to form a final estimate and an uncertainty level. The final estimate is the mean of the predictions of the different models and the uncertainty level is the variance of the estimations of the different models. Predicting using a combination of different models is a type of model averaging. The different models that are trained on the data can have the same network structure with equal hyperparameters for training in place. This is possible due to the high stochasticity of neural network training. The network weights are initialized in a different manner and gradient evaluation using batches will almost always cause the network to converge to a different set of network weights. The advantage of this method is that it is very easy to implement and that is it is possible to use this method of uncertainty level estimation for all different neural network structures. The disadvantage is that one must train a large number of neural networks and that the uncertainty level estimation may not be accurate at points where all models predict incorrectly.

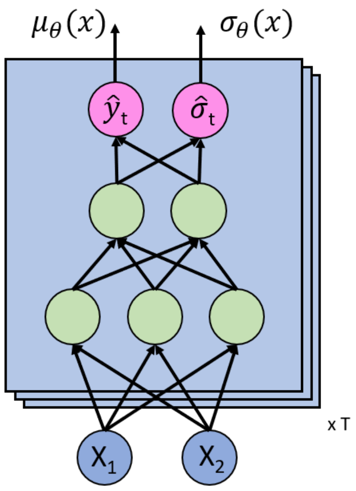

Figure 4.11: Structure of UDE

Lakshminarayanan et al. (2016) adapt this method by training the individual models in such a way that they predict the outcome and uncertainty level simultaneously. This is achieved by training the neural networks to output two values, the prediction and the respective uncertainty in terms of the variance, shown in Figure 4.11. To train a network in such a way, the negative log-likelihood criterion denoted in Equation 4.42 is minimized. This allows the two outputs to be trained synchronously. This model is called the Uncertainty Deep Ensemble (UDE) and can be applied to all types of neural networks discussed.

$$-\log p_\theta(y_n|x_n) = \frac{\log\sigma_\theta^2(x)}{2} + \frac{(y-\mu_\theta(x))^2}{2\sigma_\theta^2(x)} + c \tag{4.42}$$

The predictions of the ensemble are considered to be a mixture of Gaussian distributions. To compute predictive probabilities of the ensemble there is assumed that the ensemble predictions form a mixture of Gaussians which mean and variance are equal to the mean and variance of the true distribution. The variance of a Gaussian mixture cannot be calculated by simply taking the mean of all estimates. Instead, Equation 4.43 should be applied where $\mu_*(x)$ is the mean of all predictions. In this manner a Uncertainty Deep Ensemble (UDE) is constructed.

$$\sigma_*^2 = M^{-1}\sum_m(\sigma_{\theta_m}^2 + \mu_{\theta_m}^2(x)) - \mu_*^2(x) \tag{4.43}$$

The paper also proposes the use of adversarial training, a method of training neural networks proposed by Szegedy et al. (2013) and further improved by Goodfellow et al. (2014). Using this method adversarial examples are generated by applying small perturbations to the training data. These perturbations are chosen in a direction such that the loss is maximally increased. This direction is found by evaluation of the gradient of the loss function. These adversarial examples are generated via the fast gradient sign method, shown in Equation 4.44. Here, ϵ denotes the adversarial training multiplier.

$$x' = x + \epsilon \,\text{sign}(\nabla_x l(\theta, x, y)) \tag{4.44}$$

Adding these adversarial examples to the training data can increase the robustness of the estimator. The method smooths the predictive distribution of the estimator around the observed training examples in an efficient manner. Namely, only in the direction where the gradient is the largest.

Monte Carlo Dropout

There also exist methods to estimate the uncertainty of an estimation using a single model, such as Monte Carlo Dropout (MCDO) (Gal and Ghahramani, 2016). The use of a single neural network model clearly decreases training time over ensemble methods. However, the overall idea of the method is comparable to that of ensemble methods. Namely, the single model is used to create a large number of different models by the use of dropout. The dropout method deletes neurons randomly from the network with a certain tuned probability. This creates a submodel with slightly different behaviour. For MCDO to operate, a number of submodels are created by applying dropout to the network. An ensemble is formed from these submodels. The final estimation is the mean of the predictions of the submodels and the uncertainty is the variance of the predictions. A visual representation of this mechanism is presented in Figure 4.12. It is important to note that MCDO can only be applied to networks where a dropout layer is in place. However, it is possible to put a dropout layer in place where the nodes have a probability of zero of dropping out. This will not interfere during the training phase, but allows the use of MCDO during the interference phase.

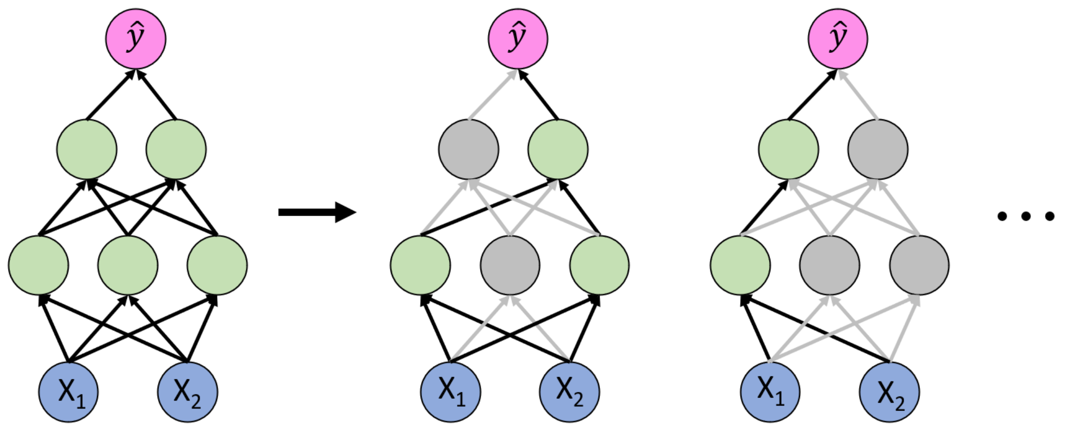

Figure 4.12: Submodel creation MCDO

The dropout rate during the interference phase needs to be tuned such that the confidence intervals overlap with the true error distribution. The assumption that the predictions of the submodels form a normal distribution has to be made. However, this can be verified by making a large number of predictions using the different submodels and performing a Jarque-Bera test. When the distribution is known, the dropout rate must be tuned such that the number of predictions in a certain probability interval is in accordance with the number of predictions of the normal distribution in that probability interval. To be more precise, the dropout rate is increased until 95 % of the measurements lie within the 95 % confidence interval of the estimation.

Monte Carlo Batch Normalisation

The last method to estimate the uncertainty of neural network predictions is Monte Carlo Batch Normalisation (MCBN) (Teye et al., 2018). The method to estimate the uncertainty is based around the batch normalisation layer, discussed in Section 4.1. To use a neural network model for MCBN, it needs to be trained in a normal manner with a batch normalisation layer in place. Only in the inference phase changes are made such that the uncertainty level can be estimated.

It is important to recall that the batch normalisation layer is positioned between the linear operations in the artificial neuron and the activation function. The output of the linear operation needs to be normalised to avoid internal covariance shift. Using the normal batch normalisation layer this is achieved by subtracting the mean of that batch and normalising using the variance of the respective batch, as shown in Equation 4.8. When using MCBN the batch statistics of the current batch are not used, but instead the statistics of a randomly sampled batch from the training data. A number of T times a different randomly sampled batch is drawn and the corresponding statistics are being used to normalise the output of the linear operation. After this normalisation step the output is scaled and shifted by learned parameters γ and β. The parameters are learned via an exponential moving average. This step allows the model to determine the optimal distribution for each hidden layer. An overview of MCBN and in what way the layer is implemented in the artificial neuron is shown in Figure 4.13.

The drawing of the randomly sampled batches from the training data introduces randomness to the inference phase. The number of times a randomly sampled batch of training data has to be drawn (T) and the size of the respective batch are hyperparameters that need to be tuned.

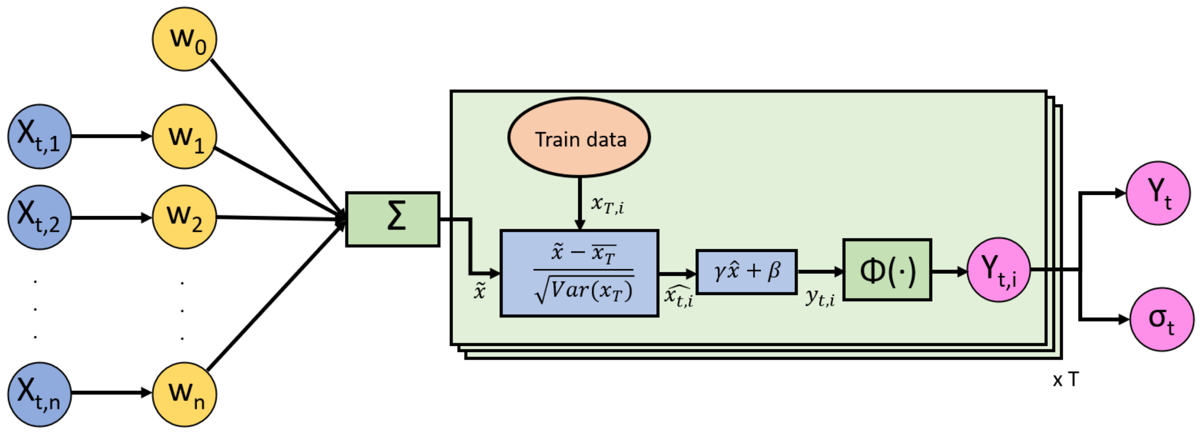

Figure 4.13: MCBN method visualised

A disadvantage of MCBN is that it can only be used in network architectures where a batch normalisation layer is present, which is not trivial. Namely, applying a batch normalisation layer to a RNN is deemed foul practise. This is caused by the fact that the estimated statistics of the incoming distribution are shared between all timesteps. This can cause batch normalisation to be less effective for RNN structures. Instead, for RNN structures one should implement layer normalisation introduced by Ba et al. (2016). Unfortunately, it is not possible to apply a procedure equivalent to MCBN to layer normalisation.

4.4.4. Observer model

In the hybrid structure an observer must be implemented to make the estimation more robust and smooth. The observer is constructed on an underlying model that describes the dynamics of the system. In this case the observer must be based on a vehicle model that describes the dynamics of the vehicle in an accurate manner such that accurate estimation of the VSA can be achieved. For analysis of the VSA, the nonlinear bicycle model is selected, a common choice in literature (Cheli et al., 2015; Kim et al., 2020). This simple model is able to provide accurate sideslip angle estimates.

The nonlinear bicycle model is used to compute several states that cannot be measured directly, including the VSA. The motions of the vehicle are derived via Newton's second law. Via the equilibrium equations of the system the lateral velocity (v) and yaw rate (r) states are updated in the observer model. Since the bicycle model assumes constant forward velocity (u), the forces in the forward direction (x) must sum to zero. In the equilibrium equations l_f and l_r respectively denote the distance from the CoG to the front and rear wheels. The mass of the vehicle is m and I_z denotes the moment of inertia of the vehicle around the vertical axis.

$$\sum F_y = F_{yf}\cos(\delta) + F_{yr} = m(\dot{v} + ru) \tag{4.45}$$

$$\sum M_z = l_f F_{yf} - l_r F_{yr} = I_z \dot{r} \tag{4.46}$$

To use the nonlinear bicycle model the tire forces have to be estimated, which requires a tire model. There is chosen to use the Dugoff tire model, introduced by Dugoff et al. (1969). The Dugoff model provides analytical relations of the longitudinal and lateral tire forces as a function of the slip angle and slip ratios. Therefore, it accounts for the coupling of the forces in both directions, indicated by the friction ellipse. The model ignores the effects of turn slip and camber.
The tire slip angle of the front and rear wheel are calculated using the lateral and longitudinal velocity of the vehicle, as well as the steering angle (δ). The distance from the CoG to the front and rear wheels is respectively indicated by l_f and l_r. The vertical force of the wheels (F_z) is calculated using the mass of the vehicle (m) and the geometry of the bicycle model. Here, L denotes the length of the wheelbase of the vehicle and CoG_z is the height of the CoG.

$$\alpha_f = \delta - \arctan\left(\frac{v + l_f r}{u}\right) \tag{4.47}$$

$$\alpha_r = \arctan\left(-\frac{v - l_r r}{u}\right) \tag{4.48}$$

$$F_{z,f} = \frac{mgl_r}{L} - \frac{mCoG_z\dot{u}}{L} \tag{4.49}$$

$$F_{z,r} = \frac{mgl_f}{L} + \frac{mCoG_z\dot{u}}{L} \tag{4.50}$$

Next, the road surface friction coefficient of both tires is calculated using the longitudinal velocity, a road surface constant (μ_0), the slip angle of the tire and the friction reduction coefficient of the tire (e_r).

$$\mu_f = \mu_0\left(1 - e_r u\sqrt{tan(\alpha_f)^2}\right) \tag{4.51}$$

$$\mu_r = \mu_0\left(1 - e_r u\sqrt{tan(\alpha_r)^2}\right) \tag{4.52}$$

Then, it is computed if the tires are behaving in the linear region by computing the λ variable. When λ is greater than 1, the tire is behaving linearly. The cornering stiffness of the tire, which is a tire dependent characteristic, is denoted by C_α.

$$\lambda_f = \frac{\mu_f F_{z,f}}{2\sqrt{(C_{\alpha,f}tan(\alpha_f))^2}} \tag{4.53}$$

$$\lambda_r = \frac{\mu_r F_{z,r}}{2\sqrt{(C_{\alpha,r}tan(\alpha_r))^2}} \tag{4.54}$$

$$f(\lambda) = \begin{cases} \lambda(2-\lambda) & \text{if } \lambda < 1 \\ 1 & \text{if } \lambda \geq 1 \end{cases} \tag{4.55}$$

Finally, the lateral and longitudinal tire forces can be calculated. However, here we are only interested in the lateral tire forces.

$$F_{y,f} = C_{\alpha,f}tan(\alpha_f)f(\lambda_f) \tag{4.56}$$

$$F_{y,r} = C_{\alpha,r}tan(\alpha_r)f(\lambda_r) \tag{4.57}$$

The advantage of the Dugoff tire model is that it contains a low number of parameters to tune. However, the shape of the tire curve is very limited where the peak in the tire curve cannot be modelled. This makes it a simplified representation of the real tire behaviour.

Various observer variations are implemented in an attempt to achieve optimal performance. All different variations are used in combination with the EKF and UKF. The only exception is the tire forces observer which is only being used in combination with the EKF.
An important note is that all observers are designed under the assumption that no road bank angles are present. In contrast, neural networks are able to deal with road bank angles, although the quality of the estimation might be slightly lower.

Standard (ST): The first observer variation to discuss is the standard observer. This observer uses the steering angle, longitudinal velocity and longitudinal acceleration as inputs. The observer states are the lateral velocity and yaw rate. The sideslip angle is then calculated using the estimated lateral velocity and measured longitudinal velocity.

Tire forces (TF): The tire forces observer variation also makes use of the longitudinal force on the front wheels. These forces, measured per wheel, add two inputs to the observer. Van Aalst (2020) proposed to use these forces to compute the yaw moment of the vehicle at the CoG. Also, the force component in the lateral direction created by the steering angle of the front wheels can be computed. This extra information is useful for the prediction phase as well as for the computation of the first derivative of the transition and measurement function.

Extra measurements (XM): In the extra measurement variation the lateral forces on the front and rear wheels are added as two measurements to the observer. This addition leads to an extension of the dimensions of the transition and measurement function. This extension allows a more precise modelling of the vehicle dynamics and aims to improve the quality of the transition and measurement function. It also allows an analysis of the error between the estimated and true lateral force.

Adaptive (AD): In the last observer variation the measurement noise matrix is adaptive to the error on the lateral tire forces estimation. When the lateral force error is higher than a certain threshold, the noise levels corresponding to the lateral forces states are changed.

The true tire forces are presented as measurements to the observer. The tire forces are also estimated by the observer using the Dugoff tire model. When the error between the estimated and measured lateral tire forces becomes too large, the tire is probably operating in the nonlinear region. At this point the measurement noise on the tire forces is increased. However, one should avoid constant switching between a reduced and normal value for the sensor noise. Therefore, the hysteresis principle is introduced by Tuononen (2009) for the situation where the lateral force error becomes too large. The noise levels in the covariance matrix increase (mode 2) when the sum of the lateral tire forces error is above the threshold τ_2 and are held constant (mode 1) when the sum of the lateral tire forces error is below τ_1. Here it is important to note that τ_2 is higher than τ_1, when the difference between these two thresholds is large enough, chattering is avoided. The hysteresis principle is visualised in Figure 4.14.

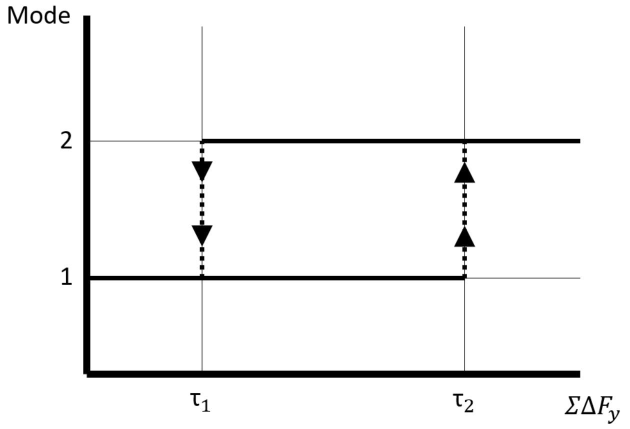

Figure 4.14: Hysteresis principle

Cheli et al. (2015) propose to increase the sensor noise values in a linear way where Van Aalst (2020) proposes to increase the R matrix values nonlinearly, as shown in Equation 4.58. Here, R_{ii} denotes the original measurement noise value, MR denotes the maximum reduction parameter, $F_{y,err}$ is the lateral tire forces error and σ is the sigma parameter. During the tuning process MR and σ have to be tuned in an attempt to achieve maximal performance. The nonlinear increment showed better performance and is therefore selected.

$$R_{ii,new} = R_{ii} - MR * (1 - exp(-0.5(F_{err}/\sigma)^2)) \tag{4.58}$$

4.4.5. Hybrid structure

The hybrid structure is created by combining the data-driven model with the observer. A visual representation of the hybrid structure is presented in Figure 4.15. The possible variations for the different components of the hybrid model are displayed in the respective blocks.

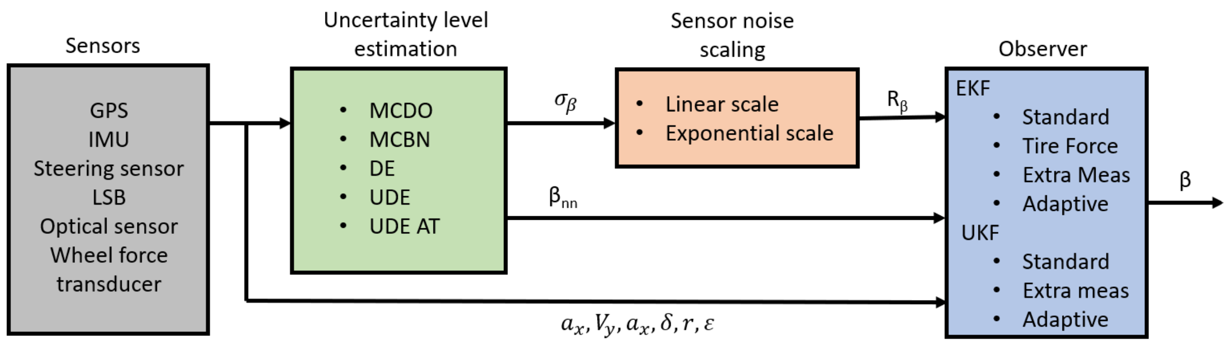

Figure 4.15: Proposed hybrid structure with suggested components

In Section 4.4.1 and 4.4.2 the mathematical formulations of the EKF and UKF are introduced. In these filters two types of uncertainty are introduced, process noise (Q) and measurement noise (R).
The transition matrix is used to compute the states of the next timestep based on the current state. This matrix is often an approximation of the true dynamics, which can cause a slight error in the state prediction. The process noise is therefore added to the covariance of this conditional state estimate to account for untracked influences. The process noise Q can therefore be interpreted as a level of confidence one has in the quality of the transition matrix in that operating region.
In the proposed structure there are also measurements available that can help to refine the state estimation. However, the measurements (z) can also contain a certain amount of noise, captured in the measurement noise matrix R. This covariance matrix is used to help compute the eventual Kalman gain matrix K. It is very well possible that the amount of measurement noise changes over time, dependent on an external variable.

The matrices Q and R need to be tuned for the specific task such that optimal performance can be achieved. Both Q and R are assumed to be diagonal matrices with on the diagonal the noise levels of the corresponding state or measurement. The proposed hybrid approach uses the uncertainty level of the sideslip angle estimate to compute the sensor noise. Namely, the estimation of the neural network is used as a measurement to the observer and a scaled version of the respective uncertainty level is used as the corresponding noise.

The measurement noise matrix R is not constant over all timesteps since the uncertainty level of the neural network estimator is different for each timestep. Therefore, each timestep the measurement noise corresponding to the sideslip angle estimate has to be adapted based on this uncertainty level. When the uncertainty of the neural network estimator is high, the corresponding measurement noise is high. When the measurement noise is high, the observer does not have a lot of trust into the corresponding measurement and determines it is unreliable. This causes the observer to trust the estimation of the internal model more and thus weights this value higher. Vice versa, when the measurement noise is low, the observer deems the estimation of the neural network reliable. The final outcome of the observer is a weighted average of the measurement from the neural network estimator and the estimation of the internal model. It must be noted that this weighted average is not linearly weighted based on the amount of measurement and process noise.

For proper operation of the observer, the uncertainty level estimate has to be scaled by a constant. This constant has to be tuned such that the scale of the measurement noise of the sideslip angle measurement is optimal for estimation. The sensor noise corresponding to the neural network estimate is calculated by Equation 4.59, where R_{scale} denotes the scaling constant.

$$R_\beta = R_{scale}\sigma_\beta \tag{4.59}$$

It is significantly more difficult to find a proper method of adjusting the Q matrix. This matrix must be adjusted based on how much confidence there is in the prediction of the transition matrix. When the prediction is inaccurate, the process noise should be increased and when the prediction is accurate it should be decreased. An ”innovation-based” method to adapt the process noise in such a way is proposed by Akhlagi et al. (2017). The difference between the posterior state estimate and the predicted state estimate is calculated each time step. This difference is called the ”innovation” and is computed via Equation 4.60 where x_t^+ denotes the posterior state estimate and $\hat{x}_t^-$ denotes the predicted state estimate.

$$\begin{aligned} \hat{w}_{t-1} &= x_t^+ - \hat{x}_t^- && (4.60)\\ &= K_t(z_t - H_t(x_t^-)) && (4.61)\\ &= K_t d_t && (4.62) \end{aligned}$$

The process noise Q can be approximated by calculating the covariance matrix corresponding to this ”innovation”. Calculating the covariance matrix is achieved via Equation 4.63. This multiplication ensures the covariance matrix is a positive semi-definite symmetric matrix.

$$Q_{k-1} = E[\hat{w}_{t-1}\hat{w}_{t-1}^T] = K_t E[d_t d_t^T] K_t^T \tag{4.63}$$

Unfortunately, it is not possible to estimate the expected value online in a sequential manner. Therefore, it is approximated by an exponential moving average in the following manner. Here α is a forgetting factor to tune that determines how much the process noise is influenced by the last estimate.

$$Q_k = \alpha Q_{k-1} + (1-\alpha)(K_t d_t d_t^T K_t^T) \tag{4.64}$$

This process noise estimate can be used in the next time step. This approach is an approximation, but allows adaptive feedback based on the available measurements. The method indicates how the quality of the transition matrix changes for different operating regions. For example, it is possible that for a certain range of lateral accelerations the estimation of the transition matrix is really accurate. This would result in a small ”innovation” and thus the process noise would decrease. Therefore, a higher weight is placed on the estimation from the transition matrix. The final estimation will therefore be closer to the estimation of the observer model.

4.5. Summary

In this chapter the different estimation models are introduced. The data-driven models consist of the FFNN, RNN and Transformer. The RNN and Transformer are able to model temporal behaviour since these structures can estimate based on the data measurements of multiple time steps. Additionally, the methods to compute the uncertainty of the neural network estimate are introduced. The uncertainty level estimate allows the proper adaptation for the covariance matrix of the observer. MCDO and MCBN can be used in combination with the original model while the UDE requires a deep ensemble to be trained. The different observer model variations are constructed using the EKF and UKF based on the nonlinear bicycle model. Linear and exponential scaling of the uncertainty estimate is proposed for adaptation of the covariance matrices.

In the next chapter the models introduced in this chapter are evaluated on a test set. Additionally, a method is introduced to evaluate the robustness of the used hybrid model. Finally, the computational complexity and required training time of the different models is analysed.

5 Results

In this chapter the performance of the different estimation structures is evaluated and analysed. Firstly, the performance of the data-driven neural networks is discussed and evaluated. Secondly, the performance of the neural networks that are able to predict uncertainty levels of their estimations are discussed. This is achieved using various performance metrics that are able to judge the quality of the predicted uncertainty level. Next, the hybrid structures are evaluated that combine the data-driven neural network estimator with a model-based observer. The robustness of these hybrid structures is put to the test by combining the observer with a suboptimal neural network. At last, the required computational time of the different estimation methods is discussed.

5.1. Neural network models

In this section first the model optimization procedure of the different neural networks is addressed. The hyperparameters that need to be tuned and corresponding network design choices are reasoned. Additionally, the optimal hyperparameters are presented. Finally, the performance of the different networks is presented and compared.
The neural networks in the current section are fitted on the features of Dataset 3 in Table 3.1. This is the feature set that included the six bearing strains per wheel.

5.1.1. Feedforward neural network

There is chosen to select a ’simple’ network structure for the FFNN estimator. A network with a maximum of two hidden layers and a limited amount of neurons in each layer. The tuning and training of such a network is fairly straightforward and it allows an evaluation of the capabilities of a ’simple’ network structure. Even within a simple neural network structure there are still many hyperparameters to tune. Examples are the learning rate, number of hidden layers, number of neurons in these layers, dropout rate of the neurons, and the activation function. As discussed, there are several methods of tuning these parameters. There is chosen to implement a Bayesian optimisation algorithm for hyperparameter tuning. This method, discussed in Section 4.1, allows evaluation of only the most promising combinations of hyperparameters to speed up the tuning process. The optimisation is implemented using the gp_minimize from the Scikit-learn optimisation toolbox (Pedregosa et al., 2011).

Hyperparameter optimisation requires the user to indicate the ranges of the different hyperparameters and the respective prior distribution. The exact ranges and distributions of the different parameters are presented in Table 5.1. For these ’simple’ estimators the size of the network is kept relatively small and limited to 300 and 200 neurons for the respective layers. It is possible for the network to consist of either one or two hidden layers. Furthermore, a linear, nonlinear non-saturated (ReLU) and nonlinear saturated (Sigmoid & Tanh) activation function are evaluated. The dropout rate for each neuron is restricted to being between 0 and 0.4. It is important to note that a uniform distribution over all hyperparameters is selected with exception of the learning rate parameter. For this parameter a log-uniform distribution is chosen due to the multiplicative effect of this parameter on the training dynamics. Namely, the error propagated back through the network is the learning rate times the loss of the function. For this task a log-uniform search for the parameters is more suited.

	Range	Optimal
Number of hidden layers	1-2	2
Number of nodes layer 1	50-300	250
Number of nodes layer 2	50-200	100
Learning rate	$10^{-4} - 10^{-1}$	0.001
Activation function	Linear - ReLU - Sigmoid - Tanh	ReLU
Dropout	0-0.4	0.2

Table 5.1: Optimized hyperparameters FFNN

Next to developing a shallow FFNN with only two layers, a Deep Neural Network (DNN) was trained. Such networks are constructed using a higher number of hidden layers to model more complex nonlinear relations between the input variables. However, these deeper networks are not able to match the performance and are quickly overfitting to the data. Therefore, no deeper feedforward neural networks are analysed.

5.1.2. Recurrent neural network

The hyperparameters that need to be tuned for the optimal RNN are mostly equal to those of the FFNN. However, there is one important additional hyperparameter, the length of the time window of measurements being used by the RNN. This time window needs to be tuned properly, when the time window is too short, not all available information is used. In contrast, when the time window is too long, unnecessary data is added as an input to the estimation, which can act as undesirable noise.
In literature the used time windows differ significantly. Ghosh et al. (2018) use a time window of 0.5 seconds, while Sieberg et al. (2021) use a window of only 0.1 seconds. Therefore, there is optimized over the range of 5 to 50 timesteps (0.05-0.5 seconds). Additionally, when analysing the vehicle behaviour during the J-turn manoeuvres, the transient response seems the have a maximal duration between 0.2 and 0.6 seconds. This validates the range of the time window.

The next design choice is to select the type of recurrent cell to use. The options include the normal recurrent cell, the LSTM cell and the GRU. Due to results in literature regarding the performance of the LSTM cell, the LSTM cell is selected as a starting point. The same priors hold over the hyperparameters introduced in the FFNN section, for the length of the time window a uniform prior is used.

	Range	Optimal
Length time window	5-50	20
Number of LSTM cells	50-200	100
Learning rate	$10^{-4} - 10^{-1}$	0.001
Activation function	ReLU - Sigmoid - tanh	tanh
Recurrent activation function	ReLU - Sigmoid - tanh	sigmoid
Dropout	0-0.4	0.2

Table 5.2: Optimized hyperparameters RNN

The optimal time window for the RNN is 20 time steps, equivalent to 0.2 seconds. This seems to be in the same range as described in literature. Furthermore, a single layer of 100 LSTM cells with the most commonly used activation and recurrent activation function proved to be optimal. It must be noted that RNNs with a larger number of hidden layers have been evaluated as well. These models did not result in any improvement, which may be caused by the high number of parameters in LSTM networks. The LSTM cells in the single hidden layer dropped out during training with a probability of 20%. Next to the LSTM cell the performance of the RNN using the GRU is evaluated. These estimators yielded similar performance and for further experiments the network consisting of the LSTM cells is considered.

5.1.3. Transformer

Tuning the Transformer is a difficult task due to the high number of hyperparameters. The required computational time for tuning hyperparameters using Bayesian optimisation scales exponentially with the number of hyperparameters. Additionally, the required training time of a single Transformer is very high, making the tuning process computationally expensive. To keep the required computational time feasible, there is chosen to tune only the hyperparameters that have the most significant effect on the model architecture.

To use the Transformer structure, the input data must be positionally encoded. For most time series applications this can be achieved by using the Time2Vec representation introduced by Kazemi et al. (2019).

However, in the current case the timed data is not continuous over all manoeuvres or even of the same duration. The different tests are all completely independent of each other with respect to time. There is no seasonality or time dependence to model. It is therefore difficult to positionally encode one of the J-turn manoeuvres with a slalom manoeuvre in the same vector format. A new method of positional encoding must be devised for the task.

The positional encoding must encapsulate the order of the different measurements. The first option is to put all measurements of the different tests after each other and encode these using the Time2Vec representation. The problem of this approach is that seasonal relations are encoded when these are not present. Thus, the Transformer would train on relations that are not truly present.
Another approach is grouping the different tests by manoeuvre, this would result in a manoeuvre specific encoding. Still the measurements must be encoded a second time for the order within these tests. There are several issues with this encoding method, the first of which is that there is not known beforehand which type of manoeuvre is encountered when driving normally. Therefore, it is not possible to apply this encoding in real driving situations. Secondly, there does not have to be a relation between the measurements at a certain time of different tests. Namely, differing initial velocities can cause the vehicle to approach the first turn at a different time point. Nevertheless, there is chosen for this type of encoding due to the lack of better alternatives.
It must be noted that when no proper positional encoding is in place, the Transformer will behave as a FFNN. Since the order of the measurements is unknown, no more information is available for the Transformer to act on. In other words, all measurements are independent of each other, just as is the case for FFNN training.

	Range	Optimal
Number of Transformer blocks	1-4	2
Number of heads	2-4	4
Head size	32-256	128
Number of neurons in layer	32-256	64
Learning rate	$10^{-4} - 10^{-1}$	0.001
Dropout	0-0.4	0.2

Table 5.3: Optimized hyperparameters Transformer

There is chosen to discard the decoder of the Transformer model. Since the aim is to solely estimate the VSA it is unnecessary to decode the hidden state of the encoder. Namely, the VSA estimate is a single scalar for every timestep. Thus, one can train the Transformer to let this hidden state between the encoder and decoder be the VSA estimate.
To reach optimal performance only 2 Transformer blocks are required. This is significantly less than for NLP tasks where six or more blocks are used (Vaswani et al., 2017). This can be explained since the complexity of the task is significantly lower. Also, the number of heads is lower with a number of four compared to eight in NLP applications. The feedforward layer in each Transformer block consisted of 64 neurons, which dropped out at a chance of 20%.

5.1.4. Performance analysis

Table 5.4 shows the performance in terms of RMSE and ME of the different neural network models on the training, validation and test set. Additionally, the performance on the different manoeuvres in the test set is broken down. When no more than three tests of the respective manoeuvre are present in the test data set, only the average is shown. If four or more tests are present, also the minimum and maximum of the performance metrics over the tests is shown. The best performances are made bold.

			FFNN	RNN	Transformer
Training	RMSE	Avg	0.067	**0.063**	0.068
Validation	RMSE	Avg	**0.045**	0.056	0.059
Test	RMSE	Avg	0.284	**0.276**	0.298
	ME	Max	**2.254**	2.570	2.481
Braking turn	RMSE	Avg	0.211	0.219	**0.209**
	ME	Avg	0.795	0.735	**0.717**
Circle	RMSE	Avg	0.376	**0.369**	0.410
	ME	Avg	1.210	**1.130**	1.508
Hockenheim	RMSE	Avg	0.271	**0.269**	0.301
	ME	Avg	2.254	**1.755**	1.847
J-turn	RMSE	Avg	0.258	0.258	**0.257**
		Min	**0.170**	0.201	0.180
		Max	0.417	**0.399**	0.437
	ME	Avg	0.867	**0.770**	0.871
		Min	**0.430**	0.545	0.463
		Max	1.265	**0.995**	1.339
Lane change	RMSE	Avg	0.226	0.229	**0.221**
		Min	0.216	0.217	**0.211**
		Max	0.236	0.245	**0.235**
	ME	Avg	**1.100**	1.138	1.194
		Min	0.767	**0.764**	0.967
		Max	1.553	**1.544**	1.617
Random steering	RMSE	Avg	**0.175**	0.179	0.182
	ME	Avg	1.356	1.074	**0.924**
Slalom	RMSE	Avg	0.384	**0.347**	0.402
		Min	0.287	**0.268**	0.355
		Max	0.438	**0.395**	0.446
	ME	Avg	2.182	1.697	**1.695**
		Min	**1.159**	1.305	1.410
		Max	**2.182**	2.570	2.481
Spiral	RMSE	Avg	0.464	**0.367**	0.371
	ME	Avg	1.399	1.353	**1.270**

Table 5.4: Performance neural network architectures in RMSE [deg] & ME [deg]

From Table 5.4 it can be observed that the RNN with LSTM cells shows the best performance in terms of RMSE on the test set. However, the FFNN estimator has the lowest ME over all the test manoeuvres and follows closely in second in terms of RMSE. The differences in terms of RMSE and ME are very small between the models on the various manoeuvres. During the slalom and lane change manoeuvre when the vehicle exhibits transient behaviour the RNN and Transformer should be able to outperform the FFNN. This is expected since the estimator has information of previous timesteps to its disposal. However, the performance of the FFNN on these manoeuvres is very similar and thus it appears that no gain is made from the availability of these data points.

One advantage of the FFNN is the simple structure of the network that allows it to be quickly trained and feasible for all different uncertainty level estimation methods. Due to the small difference in RMSE between the RNN and FFNN no exclusions can be made. The limited performance of the Transformer can be attributed to the implementation of the positional encoding. When the Transformer is not able to recognize the order of the measurements, it has equal capabilities as the FFNN. Thus, similar performance is expected at best. However, it is likely that the positional encoding simply adds noise to the estimation.

Next to analysing the performance metrics for the various manoeuvres it is useful to perform a visual analysis. Several manoeuvres are highlighted to show the estimation differences between the models.

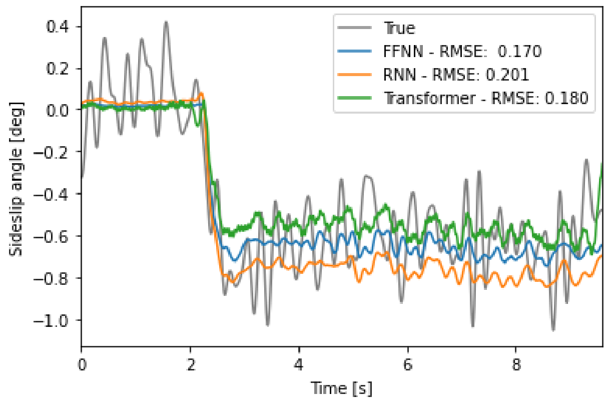

Figure 5.1: VSA estimation J-turn manoeuvre low velocity

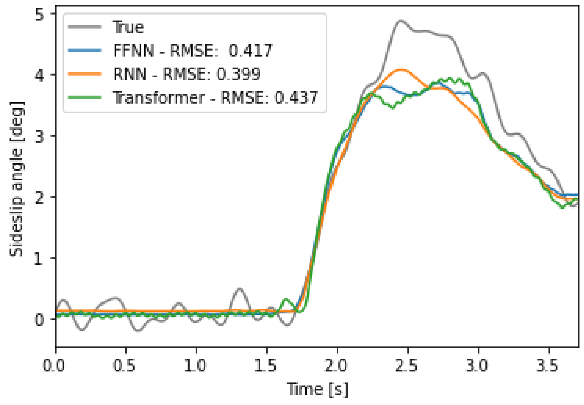

Figure 5.2: VSA estimation J-turn manoeuvre high velocity

Figure 5.1 shows that the estimated sideslip angle during the turn of the J-turn remains approximately constant a different level for the different estimators. Normally, the transient behaviour varies for different models, but the models converge to the same sideslip angle when the states remain approximately constant. This is what would be expected in the second part of the J-turn manoeuvre, but no convergence takes place.
Figure 5.2 shows that none of the estimations reach the peak of the actual sideslip angle. One explanation is that ESP is turned off during this particular test, which can result in a higher VSA. This information is not available for the neural network estimator. Evaluation of tests with ESP turned off can provide information regarding the generalisation capabilities of the estimator. In this case it appears that the estimator is not fully able to generalise for the ESP being turned off.

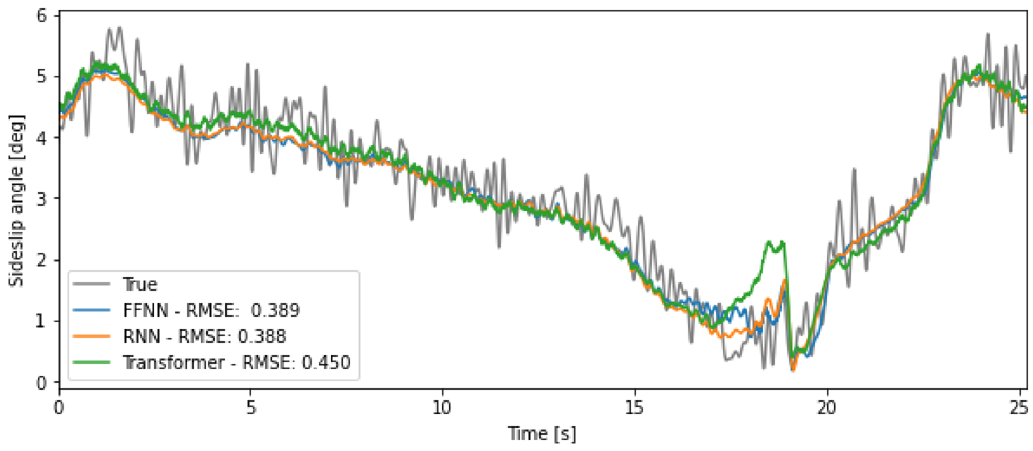

Figure 5.3: VSA estimation circle manoeuvre with diameter of 25 meter

During the circle manoeuvre shown in Figure 5.3 the Transformer shows unwanted behaviour. Between the 15 and 20 seconds mark the model estimates a sideslip angle that is too high. The other models also seem to be disrupted during this period, although significantly less. It is unclear what is the cause of this behaviour.

During the Slalom manoeuvre shown in Figure 5.4 very high sideslip angles are reached. All models show strange behaviour during the last peak where it is estimated that the VSA decreases slowly for some short time. In reality the sideslip angle is still decreasing fast which leads to a significant error. However, all estimation models are able to model the magnitude of the peak. No large differences between the different models appeared during this manoeuvre.

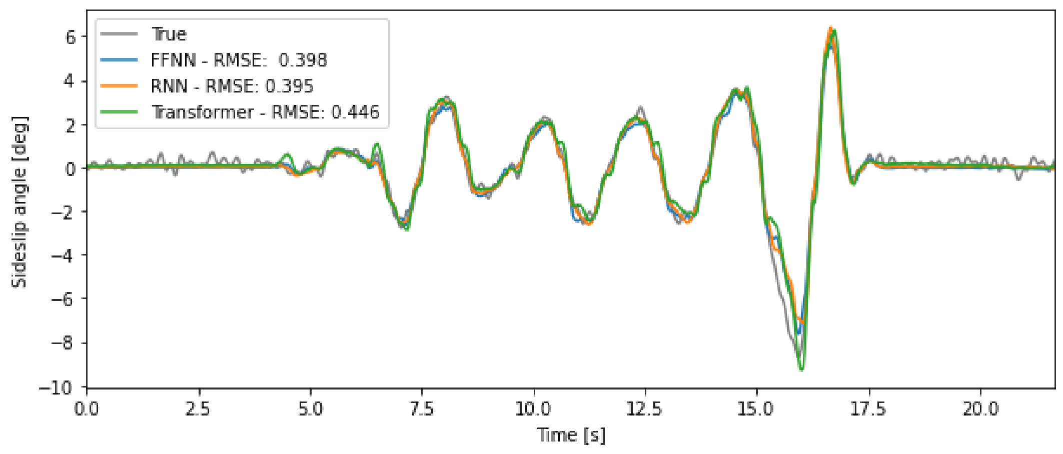

Figure 5.4: VSA estimation slalom manoeuvre

As discussed, the most problematic manoeuvres are highlighted and discussed in this section. Overall, the neural network estimators showed the ability to robustly estimate the sideslip angle in a large number of situations. The only issue is that there is not much data available of very high sideslip angle measurements. Therefore, the neural network estimator may not have trained sufficiently in this particular operating region.

5.2. Uncertainty level estimation

There are different methods of estimating the uncertainty level of a neural network estimation. To compare the performance of these methods MCDO, MCBN, a Deep Ensemble (DE), an UDE and an uncertainty deep ensemble using adversarial training (UDE AT) are implemented. The used estimation methods are trained and tuned upon the earlier defined training and validation set, the performance is again evaluated upon the testing set.

To compare the performance of the different methods additional metrics are introduced. Namely, using the RMSE the quality of the mean prediction can be evaluated, but not the quality of the respective uncertainty level. Teye et al. (2018) use two metrics to compare the performance, the Predictive Loglikelihood (PLL) and the Continuous Ranked Probability Score (CRPS). The PLL is widely accepted as a metric for ranking the quality for uncertainty estimation. An important feature is that it makes no assumptions regarding the form of the distribution. The metric is maximized by a perfect prediction.
The CRPS takes the full predicted probability density function into account and is less sensitive to outliers than the PLL metric. The definition of the CRPS is shown in Equation 5.1. The CRPS can be interpreted as the sum of the squared area between the cumulative distribution function and 0 where $y < y_i$ and between the cumulative distribution function and 1 where $y \geq y_i$. A perfect prediction with no variance results in a CRPS of zero, the lower the CRPS the higher the performance.

$$CRPS = \int_{-\infty}^{\infty} (F(y) - \mathbf{1}(y \geq y_i))^2 dy \tag{5.1}$$

These metrics should also be used to tune the different methods. Namely, there are still a few hyperparameters that need to be tuned on the training data set to obtain optimal performance. For MCDO, the dropout rate for the inference stage has to be tuned. Using MCBN the added noise variance and batch size need to be tuned. For the deep ensembles the different hyperparameters of the individual models have to be tuned. The deep ensemble with adversarial training also contains the adversarial training multiplier that requires tuning. It is very easy to tune MCDO and MCBN since there is only a single hyperparameter and the evaluation time of the uncertainty estimation methods is low. This means that in a short time window a large number of hyperparameter values can be evaluated. Thus, the optimal value can quickly be found. This is in stark contrast with the deep ensemble estimators that depend on a large number of hyperparameters with a high evaluation time. Training a single network approximately takes one minute and since the ensembles consist of around 10 to 20 models, training a single ensemble takes 10-20 minutes. Thus, it becomes computationally expensive to perform a hyperparameter optimisation over a high number of parameter dimensions. For such an optimisation a computer cluster is required, in this study such steps are not taken. Instead only a handful of hyperparameter combinations have been evaluated that are selected based on experience with the dataset and earlier hyperparameter evaluations.

The MCDO estimator is evaluated with the dropout rate varying between 0.1 and 0.6 with steps of 0.05. This is the rate at which the neurons drop out to create the submodels. The PLL is maximized and the CRPS is minimized for a dropout rate of 0.4.
For tuning the MCBN estimator the noise variance parameter is varied between 0 and 0.5 while the number of batches is varied between 50 and 250 in steps of 50. The noise variance should be set to zero and the number of batches to 100. Thus, no noise should be added since there is enough variation in the training data set. For both the MCDO and MCBN estimator a FFNN model is used as a base model. The MCDO method is also applied to a RNN model, but this did not lead to an improvement.

As discussed, tuning the deep ensemble models is slightly more complex due to the added computational complexity. To compare the performance in a fair manner there is decided to select equal hyperparameters for all three ensembles. Upon manual tuning there is decided to train 20 FFNN models with 2 hidden layers with respectively 50 and 20 neurons. These are trained for 10 epochs each with a batch size of 256. Tuning the ensemble with adversarial training does require the tuning of a few additional hyperparameters. These are related to adversarial training, the most prominent of which is the multiplier parameter, which is set to 0.1

	RMSE [deg]	PLL [-]	CPRS [-]
FFNN	0.284	-	-
RNN	0.276	-	-
MCDO	0.267	0.796	0.149
MCBN	0.266	**0.846**	0.151
DE	0.306	0.810	0.190
UDE	**0.235**	0.655	**0.124**
UDE AT	0.237	0.669	0.125

Table 5.5: Uncertainty estimation performance

Table 5.5 shows the performance of the different uncertainty level estimators. It becomes clear that the UDEs achieve the best performance in terms of RMSE. It is unexpected to observe that the normal ensemble model is not able to meet this accuracy since the same model structure is used. On top of that, the uncertainty models need to present an estimation as well as the corresponding uncertainty level. One would suspect that this task is significantly more difficult than solely predicting the VSA. The difference between the training of the two ensembles is that the used loss function differs. For the normal deep ensemble the mean-squared error function is minimized where for the UDE the negative log-likelihood is minimized. However, for a constant uncertainty level, the negative log-likelihood loss is equal to the mean-squared error loss plus a constant. Therefore, this should not have an effect.

The predictive log likelihood is maximized by the MCBN method while the CRPS is minimized by the UDEs. It appears that the UDEs estimate a wider confidence interval than the other methods. The PLL punishes a wider confidence interval more than the CRPS metric, which can explain the discrepancy between the two methods. The wider confidence interval is no issue in this study since it will be scaled regardless.

Next, a visual representation of the estimation of MCBN and the UDE is presented in Figures 5.5 and 5.6, here the shaded blue indicates the 95% confidence interval of the estimate. A slalom manoeuvre is highlighted and immediately various observations can be made. First of all, the estimated confidence level of the UDE estimator is much wider than that of the MCBN estimator. Next, it is useful to analyse the variability of the confidence interval. When combining the uncertainty estimator with the observer, the scale of the confidence interval is not very important since this is scaled either way. However, the variability of the confidence interval is important. When the confidence interval is much larger when the model is very uncertain, this provides valuable information for the observer. In contrast, when the confidence interval is always approximately of the same size, the observer will have problems differentiating between the cases.
In Figures 5.5 and 5.6 it can be seen that the variability of the confidence interval of the UDE is much higher. In combination with the high performance in terms of RMSE of this estimator, there is chosen to select this estimator for further evaluation.

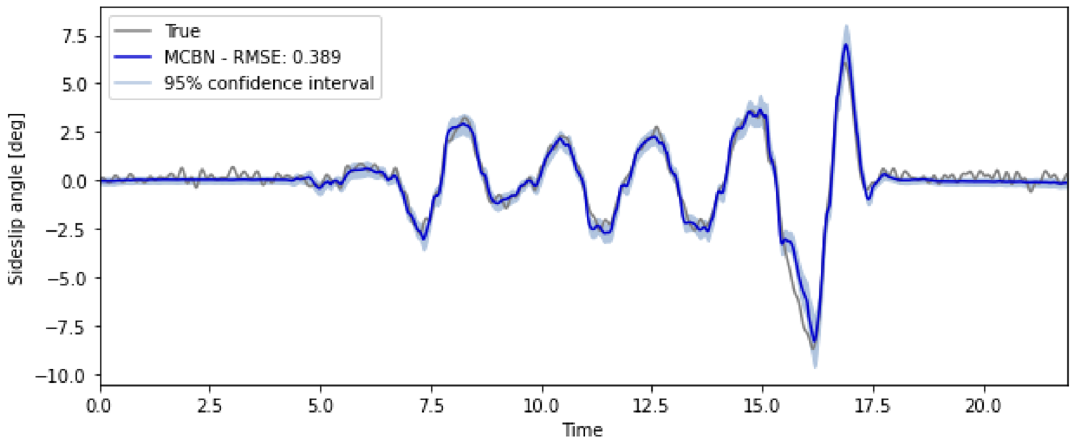

Figure 5.5: VSA estimation MCBN of slalom manoeuvre

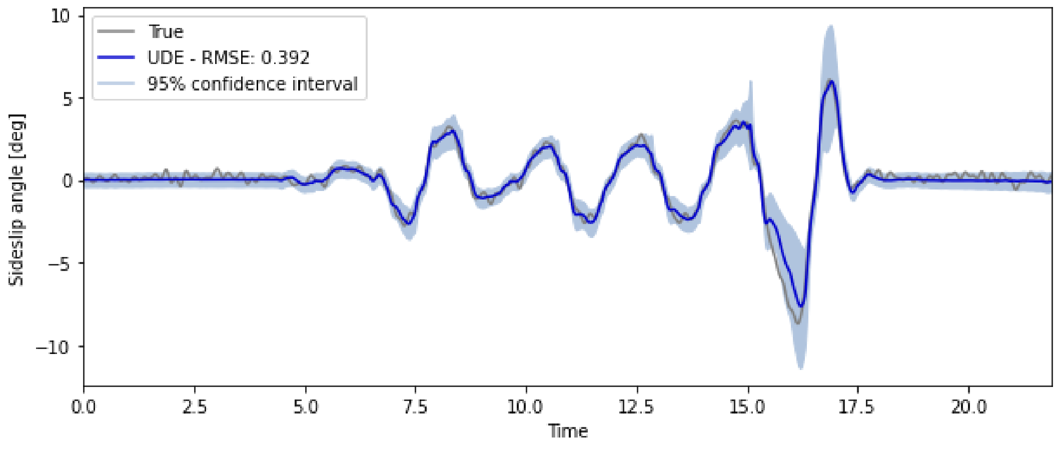

Figure 5.6: VSA estimation UDE of slalom manoeuvre

5.3. Hybrid architectures

In this section the tuning procedure for the hyperparameters of the various hybrid structures is discussed. The different hybrid estimators have a selection of hyperparameters that need to be tuned for optimal performance. Table 5.6 provides an overview of the hyperparameters that need to be tuned for the observers, which differs between the variations. The variations with an adaptive R matrix require the highest number of hyperparameters to be tuned.

Model	Q_{11}	Q_{22}	R_{scale}	$\tau_{1,2}$	MR
EKF ST	X	X	X		
EKF TF	X	X	X		
EKF XM	X	X	X		
EKF AD	X	X	X	X	X
UKF ST	X	X	X		
UKF XM	X	X	X		
UKF AD	X	X	X	X	X

Table 5.6: Parameters to tune for different hybrid structures

Torun et al. (2018) recently proposed Two-Stage Bayesian Optimisation (TSBO), an algorithm useful for optimizing parameters tied to non-convex black box functions. In these functions it is not possible to extract gradients with respect to the different parameters, not allowing gradient descent methods. TSBO is an active learning algorithm since it selects the next evaluation point based on the expectancy that it will maximize the reward, this is achieved by applying Bayes' theorem. For the respective distributions of the prior, posterior and likelihood, Gaussian distributions are assumed due to their theoretical and practical advantages. To optimize via Bayesian Optimisation an acquisition function is 'learned' and maximized to find the next point at which to evaluate.

The TSBO works, as the name suggests, in two stages, the fast exploration stage and the pure exploitation stage. The purpose of the fast exploration space is to quickly find the region in the sample space where the global optimum appears to be located. To do so, the sample space of the variables to optimize is divided into 2^d regions of hyperrectangles. The centre points of these hyperrectangles are evaluated to learn the acquisition function over the sample points. The new set of regions to evaluate are determined by querying the newly learned acquisition function. This is repeated until the Euclidean distance between the previous best and current best set of hyperparameters is lower than a set threshold.

The second stage is the pure exploitation stage and during this stage the best result from the fast exploration stage is finetuned. The found tight region is divided into three new regions along its longest dimension and candidate points are generated using the learned acquisition function. This is repeated a number of times to get closer to the optimum of the function.

The different hybrid structures are tuned using TSBO with the RMSE as evaluation metric. However, the RMSE is not evaluated on the complete validation data set, since this would severely slow the optimization process. Instead there is chosen to evaluate the RMSE on a couple handpicked manoeuvres from the validation set. These manoeuvres are chosen based on three factors, namely high sideslip angles are reached, high RMSE when evaluating the performance of the neural network estimator, and differing vehicle dynamics. First of all, it is useful to evaluate manoeuvres where high sideslip angles are reached since the tires are in the nonlinear region. Secondly, it is important to determine if adding the observer could improve the quality of the estimation where the neural network is performing relatively worse. At last, the observer must be able to deal with different types of vehicle dynamics, for example transient and steady-state behaviour. This should be reflected when tuning the hyperparameters.

5.3.1. Performance analysis

The next step is to analyse the results of the different hybrid structure variations. Table 5.7 shows the evaluation of the different performance metrics on the test set using the different observers. Note that this is the performance achieved in combination with the UDE without adversarial training.
It is important to note that for a fair comparison the Hockenheim manoeuvres that included bank angles are removed. One of the assumptions of the observer model is that no bank angles are present. Unfortunately, this causes only a single Hockenheim manoeuvre to remain for the analysis.

		EKF AD	EKF TF	EKF XM	EKF AD	UKF ST	UKF XM	UKF AD
RMSE	avg	0.282	0.259	0.262	0.267	0.263	0.264	**0.254**
RMSE NL	avg	0.341	0.303	0.311	0.315	0.333	0.334	**0.293**
ME	max	2.142	2.070	2.041	2.148	**1.841**	2.081	2.077

Table 5.7: Comparison performance hybrid structures in RMSE [deg] and ME [deg]

Optimising the hyperparameters of the model showed that a low amount of measurement noise resulted in the best performance. This is caused by the fact that the estimation performance of the UDE is high by itself. However, this causes all the different hybrid structures to reach approximately equal performance. The UKF using the adaptive R matrix (AD) presented the optimal performance in terms of RMSE. However, the ST UKF outperforms the other models significantly in terms of the ME. The difference compared to the other observer variations of 0.2 degrees is significant.

5.3.2. Robustness of hybrid structure

In the previous section the VSA estimation using the different hybrid structures is evaluated. In these structures, the observer has a lot of confidence in the predictions of the UDE. This confidence is established by a very low amount of measurement noise in the R matrix corresponding to the estimated sideslip angle. This is optimal since the UDE estimator provided accurate estimates for all manoeuvres in the test case. However, it could be the case that in a specific part of the operating range the UDE is less reliable, deteriorating the performance. This can for example be caused by the absence of enough training data points in a part of the operating region. In this case the observer must be able to correct the UDE estimate. It needs to be determined how the hybrid structures perform when there are inaccuracies in the UDE estimator. To test this, a non-optimal estimator is trained and coupled with an observer to see if the observer is able to correct the lower quality sideslip angle estimates.

Suboptimal neural network estimation

The first step is to create a suboptimal neural network estimator, which can be achieved in various ways. The estimator can be trained on only a small random subset of the training data, or only on data from a specific part of the operating range. There is chosen to train the UDE estimator on a subset of the training data. Two different subsets are created, for the first subset all the measurements are deleted where the lateral acceleration is higher than 7 m/s^2. The second subset is created in a similar manner, but this time the threshold is 5 m/s^2. The number of remaining data points after these operations and the performance of the respectively trained estimators is presented in Table 5.8. To provide some insight regarding the established thresholds, the maximal lateral acceleration the vehicle can sustain is around 10 m/s^2. The UDE is trained using the same hyperparameters as before. From the table it can be observed that the performance of the estimator drops when the measurements of high lateral accelerations are deleted. The difference in performance between the full set and the subset where only measurements with lateral accelerations lower than 7 m/s^2 ($|a_y| < 7$) are present is not that large. It appears that lowering the threshold to 5 m/s^2 ($|a_y| < 5$) significantly deteriorates the performance of the estimator.
The approach of deleting data mimics scarce to no data being available in certain operating regions. This can very well be the case in real applications and can normally be the cause of serious drops in performance of data-driven approaches. Therefore, correction of the observer model would be proof of model robustness. When an observer model is combined with a different neural network estimator, the R and Q matrices have to be tuned again.

	# data points	%	RMSE [deg]
Full	534.576	100	0.256
$\lvert a_y \rvert < 7$	479.566	89.7	0.295
$\lvert a_y \rvert < 5$	431.952	80.8	0.417

Table 5.8: Summary of data subsets

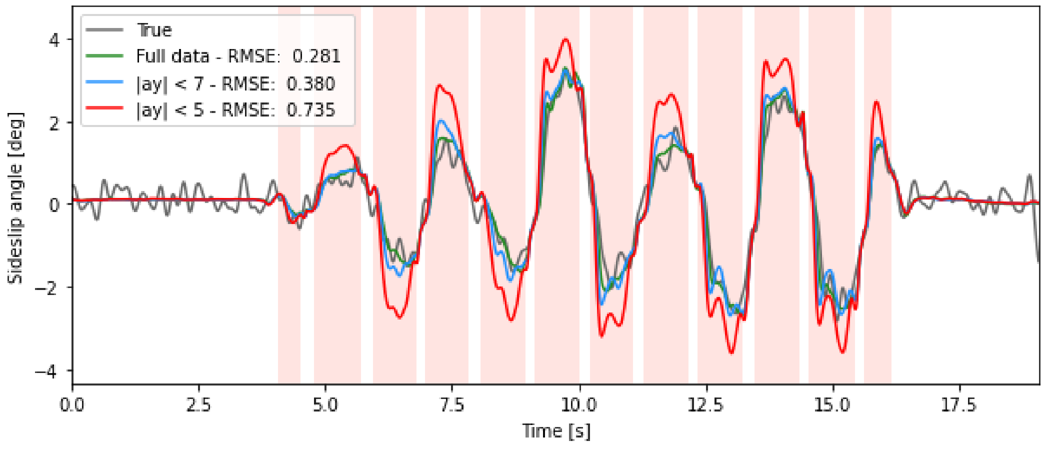

Figure 5.7: VSA estimation suboptimal UDE estimators for slalom manoeuvre

The difference between the different estimators is visually highlighted in Figure 5.7. Here, the red shaded background indicates that the absolute value of the lateral acceleration is above 5 m/s^2. It is clearly visible that the neural network that is not trained on data in this operating region is struggling to make accurate estimations.

Linear R scaling

The next step is to couple the suboptimal neural network estimators with the various observer variations and determine the performance. The structure of this hybrid model is described in Section 4.4.5 and the measurement noise is computed using Equation 4.59. It is also very useful to gain intuition about the performance of the observer variations and the uncertainty deep ensembles when these are not coupled. Therefore, the performance metrics of these cases are also presented.

	Metric	No obs	EKF ST	EKF TF	EKF XM	EKF AD	UKF ST	UKF XM	UKF AD
No NN	RMSE	-	0.407	0.406	0.401	**0.392**	0.443	0.452	0.456
	RMSE NL	-	0.600	0.596	0.594	0.582	0.567	**0.565**	**0.565**
	ME	-	2.265	**2.166**	2.384	2.270	2.534	2.383	2.381
UDE (full)	RMSE	0.256	0.282	0.259	0.262	0.267	0.263	0.264	**0.254**
	RMSE NL	0.310	0.341	0.303	0.311	0.315	0.333	0.334	**0.293**
	ME	2.205	2.142	1.964	1.985	2.148	2.679	2.663	**1.710**
UDE ($\|a_y\|$ < 7)	RMSE	**0.295**	0.319	0.307	0.312	0.364	0.316	0.344	0.368
	RMSE NL	0.442	0.430	0.409	0.425	0.493	**0.368**	0.540	0.613
	ME	2.596	2.078	2.353	2.163	2.268	**1.769**	2.928	3.787
UDE ($\|a_y\|$ < 5)	RMSE	0.417	0.400	0.397	0.404	0.392	**0.376**	0.406	0.476
	RMSE NL	0.786	0.593	0.584	0.608	0.579	0.490	0.452	**0.400**
	ME	3.314	2.268	2.211	2.270	2.270	2.024	**2.007**	2.593

Table 5.9: Non-optimal uncertainty estimation, linearly scaled R matrix, RMSE [deg] and ME [deg]

Table 5.9 shows the characteristics of the different estimators. First, the performance of the observers when these are not coupled to neural networks is discussed. It becomes apparent that the RMSE of the different variations is around 0.4 degrees and the maximal error is between 2.1 and 2.5 degrees. There is not a large difference in performance between the different hybrid variations.
For the UDEs that are trained on the full data or subsets it becomes apparent that the RMSE increases when the training set becomes smaller. This is of course expected since less training data, especially in a specific operating region, equates to worse performance for data-driven estimation methods. Furthermore, especially the RMSE in the nonlinear operating region increases, which is logical since training data from this region is deleted.

When the observer variations are coupled to the UDE that is trained on the full dataset, the hybrid approach also performed well. The presence of the observer did not deteriorate the performance in any way. In the case where the observer variations are coupled to the UDE trained on data with measurements where the lateral acceleration is lower than 7 m/s^2, the presence of the observer seems to deteriorate the performance slightly. None of the variations is able to lower the overall RMSE. However, the standard UKF variation does seem able to improve the estimation in the nonlinear operating region.
When the UDE is only trained on measurements with a lateral acceleration lower than 5 m/s^2, the presence of the observer does lower the RMSE. However, the improvement is not very large and the UKF variation with the adaptive R matrix is still providing worse performance. The RMSE in the nonlinear operating region is lowered by all observer variation, which points towards the benefit of the hybrid approach.

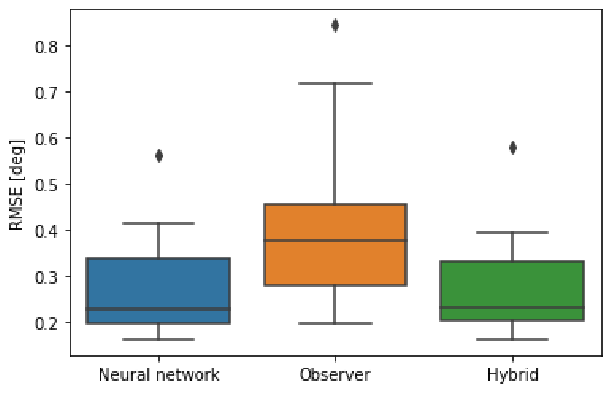

Figure 5.8: Boxplot RMSE of tests of UKF AD (full)

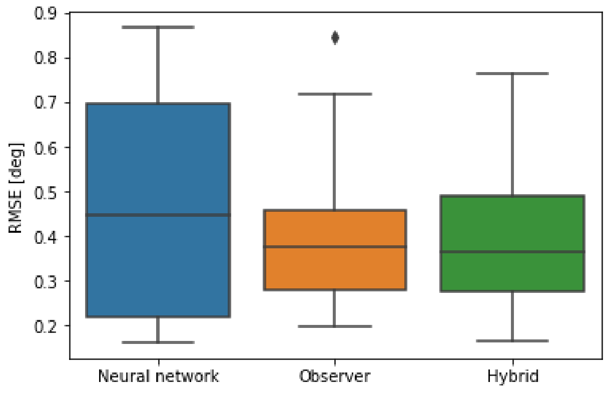

Figure 5.9: Boxplot RMSE of tests with UKF AD ($|a_y| < 5$)

In Figure 5.8 and 5.9 boxplots of the RMSE of individual tests are shown for the neural network estimator, the observer and the hybrid approach. Note that the UKF observer with an adaptive R matrix is used for this analysis. Every test in the test set represents a single point and using these boxplots it can be seen if there is a constant improvement over all tests when the hybrid structure is utilized. From the figures it becomes apparent that the hybrid approach achieves very similar performance as the neural network when this network is trained upon the complete dataset. When only a subset of the data is used and the neural network performance is significantly worse, the hybrid approach has slightly better performance than the observer. However, the hybrid approach does not seem able to perform much better than its individual components.

Upon further investigation, the observer did not seem to take over when the confidence level of the neural network estimator is low. It is found that this is caused by the scaling of the uncertainty level estimate. As discussed in Section 4.4.5 the sensor noise for the sideslip angle measurement is computed as a linear scaling of the estimated uncertainty level. However, the uncertainty level estimates are always of approximately the same order of magnitude. This causes a linear scale unable to differentiate well enough between high and low uncertainty levels. Therefore, the observer model does not take over estimation when the uncertainty level of the neural network estimator is low.

Exponential R scaling

To let different uncertainty levels have a larger influence on the sensor noise level of the sideslip angle measurement, an exponential scaling is evaluated. By using an exponential scaling more differentiation is created between approximately similar uncertainty levels. The exponential constant is of course an additional tuning parameter and the linear scaling has to compensate for the magnitude created by the exponential scaling. The new calculation of the sensor noise is shown in Equation 5.2. The corresponding results of the hybrid structures using this new scaling for calculation of the sensor noise are presented in Table 5.10.

$$R_\beta = R_{scale}\sigma_\beta^{R_{exp}} \tag{5.2}$$

	Metric	No obs	EKF ST	EKF TF	EKF XM	EKF AD	UKF ST	UKF XM	UKF AD
No NN	RMSE	-	0.407	0.406	0.401	**0.392**	0.443	0.452	0.456
	RMSE NL	-	0.600	0.596	0.594	0.582	0.567	**0.565**	**0.565**
	ME	-	2.265	**2.166**	2.384	2.270	2.534	2.383	2.381
UDE (full)	RMSE	**0.256**	0.263	0.258	0.260	0.260	0.260	0.258	0.258
	RMSE NL	**0.310**	0.313	**0.310**	0.312	0.312	0.319	0.318	0.316
	ME	2.205	2.043	**2.028**	2.168	2.188	2.409	2.177	2.049
UDE ($\|a_y\|$ < 5)	RMSE	0.417	0.405	0.386	0.389	0.371	**0.322**	0.333	0.333
	RMSE NL	0.786	0.742	0.687	0.688	0.646	**0.507**	0.547	0.543
	ME	3.314	2.704	2.865	2.458	2.580	2.949	2.563	**2.068**

Table 5.10: Non-optimal uncertainty estimation, exponentially scaled R matrix, RMSE [deg] and ME [deg].

There are several observations that can be made from the results shown in Table 5.10. The first obvious result is that the exponentially scaled uncertainty level significantly improves the performance. The hybrid estimators are able to improve the performance in terms of RMSE of the neural network estimator when it is trained on the subset of data by approximately 20%. Additionally, the hybrid models are able to outperform the pure observer model. This observation is a proof of concept of the hybrid structure, namely the combination of the neural network and the observer is outperforming its individual components. Moreover, the ME is almost equal for the UKF with the adaptive R matrix in the case of the optimal and suboptimal uncertainty deep ensemble.

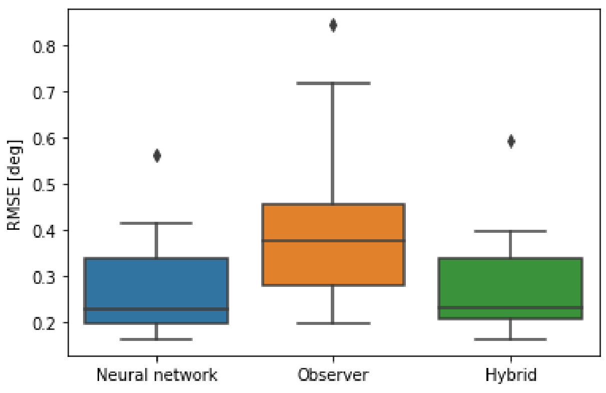

Figure 5.10: Boxplot RMSE of tests of UKF AD (full)

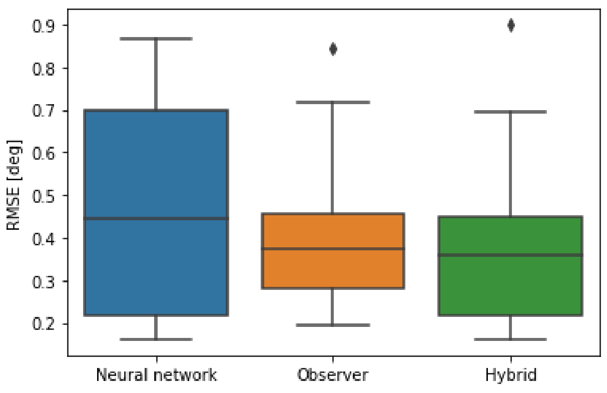

Figure 5.11: Boxplot RMSE of tests with UKF AD ($|a_y| < 5$)

Figure 5.10 and 5.11 show the boxplots of the tests using the UKF with the adaptive R matrix where the measurement noise is scaled exponentially. This has a small effect on the performance of the hybrid approach when the neural network is trained on the full dataset. However, when the suboptimal neural network is used, the hybrid approach more significantly outperforms the observer model than with the linear scaling in place. The mean is slightly lower, but the first quartile of the box is significantly lower.

Adaptive Q adjustment

The Q matrix, the process noise, is a diagonal matrix with a size equal to the number of states of the observer. For the previous hybrid estimators the values of the Q matrix remained constant. However, as discussed in Section 4.4.5 the process noise can also be adapted based on quality of the predictions made by the transition matrix. When this varies for different operating regions, the quality of the prediction can be projected to a corresponding amount of process noise.

For implementation of the adaptive Q adjustment two additional parameters are introduced. The first one is the scaling constant of the exponential moving average that is used to approximate the expected value. From exploratory testing it was found that $\alpha = 0.1$ resulted in the best performance. Secondly, the adaptive Q values have to be scaled exponentially to optimize the performance. This scaling parameter must be tuned for the different variations, which is again achieved using TSBO. Equation 5.3 shows the scaling function which scales the Q values linearly and exponentially. The results of the adaptive Q values are shown in Table 5.11.

$$Q_{new,ii} = Q_{scale,ii} Q_{ii}^{Q_{exp}} \tag{5.3}$$

	Metric	No obs	EKF ST	EKF TF	EKF XM	EKF AD	UKF ST	UKF XM	UKF AD
No NN	RMSE	-	0.407	0.406	0.401	**0.392**	0.443	0.452	0.456
	RMSE NL	-	0.600	0.596	0.594	0.582	0.567	**0.565**	**0.565**
	ME	-	2.265	**2.166**	2.384	2.270	2.534	2.383	2.381
UDE (full)	RMSE	**0.256**	0.264	0.259	0.260	0.260	0.282	0.263	**0.256**
	RMSE NL	0.310	0.314	0.311	0.312	0.311	0.344	0.310	**0.297**
	ME	2.205	2.219	**2.164**	2.268	2.269	3.827	2.397	2.272
UDE ($|a_y| < 5$)	RMSE	0.417	0.339	0.338	0.349	0.330	0.330	0.331	**0.324**
	RMSE NL	0.786	0.518	0.540	0.571	0.534	0.535	0.543	**0.509**
	ME	3.314	2.254	**2.201**	2.269	2.269	3.582	2.549	2.562

Table 5.11: Non-optimal uncertainty estimation, exponentially scaled R and adaptive Q matrix, RMSE [deg] and ME [deg].

The increase in performance over the exponentially scaled R matrix is relatively small. The adaptive UKF variation showed the best performance with an approximately equal RMSE as the standard UKF variation for the exponentially scaled measurement noise. The implementation of the adaptive Q values introduced two more hyperparameters to be trained. Adding many parameters increases the required time for the tuning process and can generally decrease generalisation capabilities. It is therefore not trivial that this adaptation of the Q matrix should be applied. It may be useful to evaluate other algorithms for adapting the process noise.

Figure 5.12 and 5.13 show the boxplots of the tests using the UKF with the exponentially scaled R and adaptive Q matrix. This slightly improved the performance of the hybrid approach when the neural network is trained on the full dataset. However, when the suboptimal neural network is used, the hybrid approach more significantly outperforms the observer model than is the case for only exponentially scaling the measurement noise. Especially the mean is significantly lower. Also the maximal RMSE is for the first time lower than is the case for the observer.

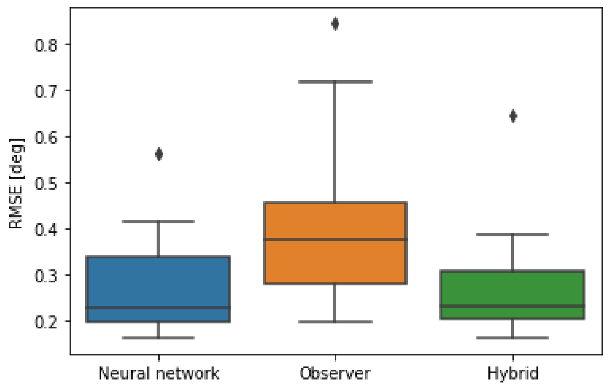

Figure 5.12: Boxplot RMSE of tests of UKF AD (full)

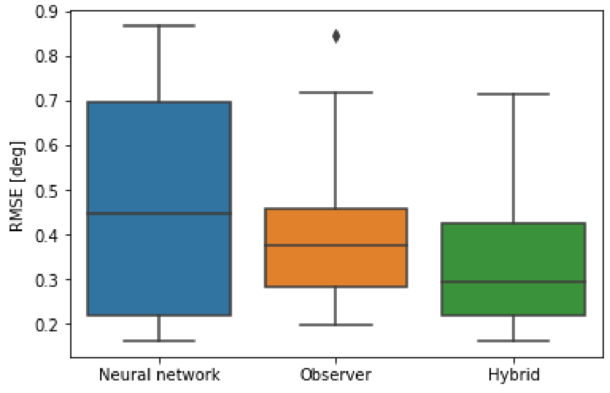

Figure 5.13: Boxplot RMSE of tests with UKF AD ($|a_y| < 5$)

5.4. Computational time

It is important to discuss the inference time of the hybrid estimators to judge if implementation in real driving applications is feasible. If the required computational time is too high, it cannot be used at the required frequency.

The code to run the hybrid structures is written in Python and the neural networks are trained using the Keras library, a front end to Tensorflow. The Tensorflow GPU library is used to train the network parallel on different cores of the Graphics Processing Unit (GPU). The observer variations are implemented using Simulink models in Matlab, these models are called from Python. The hardware used for training is a HP Omen 15 laptop with a AMD Ryzen 7 CPU 4000 series with a Nvidia GeForce GTX 1650 Ti GPU.

5.4.1. Training time

The training time is only a one-time thing causing a low training time to be desirable, but not essential. The training time of the hybrid model consists of two parts, training the ensemble of neural networks and tuning the R and Q matrix of the used observer. Training an ensemble of neural network models naturally takes longer caused by the higher number of models to train. The larger the ensemble, the higher the training time. Additionally, there is a significant difference in training time of a FFNN and a RNN, the FFNN trains much faster. An overview of the training time is presented in Table 5.12.

	Single model	UDE (20 models)
FFNN	45.5	910
RNN	384	7680

Table 5.12: Training time UDE in seconds

The next step is to train the hyperparameters of the model. As discussed, this is achieved using the TSBO algorithm introduced by Torun (2018). This algorithm allows optimization over a fixed number of function evaluations, thus, the user is able to influence the tuning time significantly. However, enough function evaluations are required to ensure that the optimal hyperparameters are found. For the observer variations with an adaptive R matrix more hyperparameters need to be tuned. Therefore, more iterations are required to find the optimal set of hyperparameters. The required number of function evaluations scales exponentially with the number of parameters to explore the hyperparameters space with equal accuracy.
The computational time required for the function used to tune the different hybrid estimators is presented in Table 5.13. For the tuning process of the ST, TF and XM variations 120 function evaluations are required. However, for tuning the AD variation with the adaptive R matrix 200 function evaluations are required. The required time for the total tuning process is also presented.

	Single eval	Tuning (120/200 evals)
EKF ST	2.9	348
EKF TF	3.0	360
EKF XM	3.1	372
EKF AD	3.2	640
UKF ST	2.6	312
UKF XM	2.7	324
UKF AD	2.9	580

Table 5.13: Required time for tuning of hybrid models in seconds

Adding up the two tasks, the hybrid model using the feedforward neural network can be trained and tuned in approximately 20 minutes. Tuning and training the hybrid model using the recurrent neural network takes approximately 2.5 hours due to the long training time of the recurrent neural network ensemble.

5.4.2. Inference time

The proposed algorithm needs to be able to operate real time. The required frequency of estimates being made is 100 Hz. The inference time essentially consists of two parts, the initialisation time and the estimation time. The initialisation time consists of loading the ensemble of models and connecting Python to Matlab for initialisation of the Simulink model used for the observer.
The inference time of the UDE in Python is around 0.06 seconds for a manoeuvre of 1000 samples, corresponding to is 10 seconds. The initialisation time is approximately equal for all types of hybrid structures. The same holds for the inference time, although this is obviously affected by the length of the manoeuvre. The initialisation time is around 14 seconds while the inference time for a manoeuvre of 10 seconds using the hybrid structure is around 0.8 seconds.

It must be noted that the used estimation pipeline for is far from optimal. For commercial implementation it is very advantageous to develop the neural network and observer model in the same programming language. Therefore, no further conclusions should be drawn from the presented inference time. For a better understanding one should gain intuition of the required number of operations for a single estimation step. The uncertainty neural network must be evaluated as well as the unscented Kalman filter. For the evaluation of a FFNN a lot less computations have to be performed compared to that of a RNN. The number of required operations for the UKF is lower than that of evaluation of the FFNN. Since evaluation of the UDE, an ensemble of 20 FFNNs, is 0.06 seconds for a manoeuvre of 1000 sample, it is expected that implementation of the hybrid estimator is feasible.

5.5. Summary

To conclude the highest performing models are summarized in Table 5.14. For a fair comparison there is chosen to exclude the two Hockenheim manoeuvres where a road bank angle is present from the test set. Therefore, the performance of the neural network and uncertainty estimators may differ slightly from earlier reported results.
The performance of the different neural networks is very similar, each different architecture performed best on several of the manoeuvres. Overall the RNN has a slight edge in performance over the other architectures in terms of RMSE. However, the feedforward neural network has the lowest ME. It is very difficult to find a proper positional encoding for the Transformer model, if one is eventually developed this could severely boost the performance of this architecture.

Model	Optimality NN	Best variation	RMSE [deg]	RMSE NL [deg]	ME [deg]
Neural network	-	RNN	0.266	0.322	2.570
Uncertainty estimator	-	UDE	0.256	0.310	2.205
Linear R	Optimal	UKF AD	0.254 (+0.0%)	0.293 (+0.0%)	1.710 (+0.0%)
Exponential R		EKF TF	0.258 (+1.6%)	0.310 (+5.8%)	2.028 (+18.6%)
Adaptive Q		UKF AD	0.256 (+0.8%)	0.297 (+1.4%)	2.272 (+32.9%)
Linear R	Suboptimal	UKF ST	0.376 (+0.0%)	0.490 (+0.0%)	2.024 (+0.0%)
Exponential R		UKF ST	0.322 (-14.4%)	0.507 (+3.5%)	2.949 (+45.7%)
Adaptive Q		UKF AD	0.324 (-13.8%)	0.509 (+3.9%)	2.562 (+26.6%)

Table 5.14: Overview performance estimation architectures

The FFNN and RNN have extensions to estimate not just the VSA itself, but also the respective confidence level. The commonly used MCDO and MCBN methods are surpassed by the performance of the UDE consisting of multiple FFNNs. It is unexpected to observe that this uncertainty estimation method outperformed the original FFNN, no explanation is found for this unexpected result. One possibility is that ensemble models are necessary in the task to achieve better results. These individual models may model different modes of the system, which causes the average to become an accurate estimation.

In hybrid structures the uncertainty level of the estimation is required to adapt the sensor noise on the sideslip angle measurement in the observer. Different model variations are evaluated in combination with the extended and unscented Kalman filter. From the table it can be concluded that when an optimal UDE is used, exponential scaling of the sensor noise and adaptive process noise has a small effect on the performance of the hybrid model. This is expected since in this case the hybrid model follows the UDE estimation closely. The effect of the observer is minimal and thus changes in these noise levels have little effect on the estimation.

Additionally, a suboptimal UDE is used to test if the observer is able to deal with suboptimal estimations. When using a suboptimal UDE, the UKF significantly outperformed the EKF variations. This is expected since UKF variations are able to capture higher order nonlinear effects. The fact that the addition of the observer actually caused an improvement in performance shows the great promise of the hybrid structures. The performance is maximized with a linear and exponentially scaled uncertainty level for the measurement noise. It must be noted that the ME is significantly increased by this scaling. However, further analysis showed this is an isolated case. The adaptive process noise does not lead to an increase in performance.

The inference time of the hybrid structure is deemed acceptable. It is difficult to make hard conclusions since an optimal pipeline has not been used in this thesis. The linking of Python with Matlab surely has clear shortcomings and online estimation with this pipeline is not possible at a rate of 100 Hz. Therefore, an analysis of the required number of calculations is more useful. There are 20 small FFNNs that have to be evaluated along with an UKF with nine sigma points. The inference time of the UDE is 0.06 seconds for a manoeuvre of 10 seconds. The use of a FFNN over a RNN decreases the inference time significantly. The computational time required for the observer is much lower when considering the limited number of required mathematical operations. Therefore, it seems reasonable to assume that the hybrid model can be used for online estimation when a suitable estimation pipeline is in place. Of course this is assuming that a processor with the same order of magnitude of computational power is used.
In the next chapter the conclusion based on these results follow. Additionally, several aspects of this study require further investigation, this is summarized in the future work section.

6 Conclusion and Future work

6.1. Conclusion

In this thesis the additional value of tire force information on VSA estimation is evaluated. There can be concluded that the addition of these led to a significant increase of performance in terms of RMSE using a FFNN. The next step is to evaluate different types of neural network architectures to optimize the respective performance. It is concluded that the FFNN and RNN yield similar performance while the Transformer model is unable to match this. This is probably due to an unsuited positional encoding, causing the model unable to retrieve the order of the measurements. Due to the lower neural network complexity and shorter training and inference times, the FFNN is selected for further use in the hybrid structure.

The goal is to design a hybrid structure using the FFNN in combination with an observer model. Different observer variations developed in literature are replicated to compare performance. To create the hybrid structure, the FFNN needs to output an estimation as well as a corresponding uncertainty level. Several methods exist to obtain the uncertainty level of a neural network estimation. These methods are compared using the RMSE as well as the PLL and CRPS. Based on these metrics there is concluded that the UDE consisting of FFNNs provides the best performance. Therefore, the UDE is used for the hybrid approach. Unexpectedly, the UDE also outperformed the developed FFNN and RNN by reducing the RMSE with respectively 17.2% and 14.9%.

To combine the UDE with the observer, the uncertainty level of the estimation needs to be scaled to the measurement noise corresponding to the VSA measurement. Since the developed UDE is accurate across the complete testing set with a RMSE of 0.256 degrees, the measurement noise corresponding to the sideslip angle is set very low. This causes similar performance for all hybrid estimator variations. The estimator does not use the capabilities of the observer, thus, the hybrid approach performs equally well as the purely data-driven approach.

To determine if the observer is able to add value when the quality of the UDE estimations is low, a suboptimal UDE is developed. This UDE is only trained on measurements with absolute lateral accelerations lower than 5 m/s^2, mimicking sparse data of high sideslip angles. Thus, the developed UDE provides less accurate estimations of high sideslip angles.
It is evaluated if the observer is able to correct these estimations. Creating a suboptimal neural network to test the hybrid structure is a new approach that provides insight into the robustness of the estimator. Without this method the true potential of the hybrid structure cannot be evaluated properly.

In current literature a linear scale is used to simply match the variance of the other sensor noise levels. However, using this linear scaling the observer is not able to correct the UDE estimation. However, it is found that scaling the uncertainty level not only linearly, but also exponentially the performance can be increased significantly. This scaling allows a higher differentiation between high and low uncertainty level estimations. When using a suboptimal neural network this change leads to a decrease of 14.4 % for the RMSE. Adapting the process noise based on the quality of the transition matrix estimation results in equal performance. Since this adaptation increases the number of parameters to tune, it is inadvisable to implement it.

6.2. Future work

There are several aspects on the subject of VSA using hybrid structures that need to be explored further. First of all, the measurement noise of the sideslip angle is computed using a linear and exponential scaling, which proved beneficial. The main objective of changing the measurement noise as well as the process noise is to influence the Kalman gain to achieve maximal performance. However, it is possible that the Kalman gain should not be solely varied based on the uncertainty level of the sideslip angle estimation, but also based on other states. This can be explored by computing the Kalman gain using a neural network, as proposed by Revach et al. (2021). For example, it may become apparent that the Kalman gain must also be changed for different lateral acceleration levels.

Next, the observer is unable to deal with road bank angles. This requires the elimination of two Hockenheim manoeuvres from the testing set when evaluating the hybrid approach. The VSA error of the observer during corners with a bank angle is very large. Of course, before it can be implemented commercially this problem needs to be fixed. It cannot be the case that driving on roads with a bank angle causes inaccurate VSA estimation. There already exist extensions of observers that attempt to estimate the banking angle as a state to correct for this. However, these extensions have not been used in combination with data-driven approaches.

Data augmentation can be used to try and improve the performance of data-driven models in high VSA situations. It is interesting to compare the performance of data augmented data-driven models to hybrid models.
Additionally, the dataset used for this study unfortunately solely consisted of tests in dry conditions. It would be insightful to evaluate the performance of the neural network and observer in various road conditions such as wet or icy.

Bibliography

Acosta, M., & Kanarachos, S. (2018). Tire lateral force estimation and grip potential identification using Neural Networks, Extended Kalman Filter, and Recursive Least Squares. *Neural Computing and Applications*, *30*(11), 3445–3465.

Aggarwal, C. (2018). *Neural networks and deep learning* (Vol. 10). Springer.

Akhlaghi, S., Zhou, N., & Huang, Z. (2017). Adaptive adjustment of noise covariance in kalman filter for dynamic state estimation. *2017 IEEE power & energy society general meeting*, 1–5.

Ayyad, A., Prohm, C., Gräber, T., Unterreiner, M., & Hilbert, M. (2021). Uncertainty estimation for neural time series with an application to sideslip angle estimation. *SAE International Journal of Connected and Automated Vehicles*, *4*(3). https://doi.org/10.4271/12-04-03-0020

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *ArXiv*, *abs/1607.06450*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, *13*(1), 281–305.

Boada, B. L., Boada, M. J. L., & Diaz, V. (2016). Vehicle sideslip angle measurement based on sensor data fusion using an integrated ANFIS and an Unscented Kalman Filter algorithm. *Mechanical Systems and Signal Processing*, *72*, 832–845.

Bonfitto, A., Feraco, S., Tonoli, A., & Amati, N. (2020). Combined regression and classification artificial neural networks for sideslip angle estimation and road condition identification. *Vehicle system dynamics*, *58*(11), 1766–1787.

Brownlee, J. (2018). *Better deep learning: Train faster, reduce overfitting, and make better predictions*. Machine Learning Mastery.

Chakraborty, I., Tsiotras, P., & Lu, J. (2011). Vehicle posture control through aggressive maneuvering for mitigation of T-bone collisions. *2011 50th IEEE Conference on Decision and Control and European Control Conference*, 3264–3269.

Chatrath, K., Zheng, Y., & Shyrokau, B. (2020). Vehicle dynamics control using model predictive control allocation combined with an adaptive parameter estimator. *SAE International Journal of Connected and Automated Vehicles*, *3*(12-03-02-0009), 103–117.

Cheli, F., Ivone, D., & Sabbioni, E. (2015). Smart tyre induced benefits in sideslip angle and friction coefficient estimation. *Sensors and instrumentation, volume 5* (pp. 73–83). Springer.

Chen, B.-C., & Hsieh, F.-C. (2008). Sideslip angle estimation using extended Kalman filter. *Vehicle System Dynamics*, *46*(S1), 353–364.

Chen, J., Song, J., Li, L., Jia, G., Ran, X., & Yang, C. (2016). UKF-based adaptive variable structure observer for vehicle sideslip with dynamic correction. *IET Control Theory & Applications*, *10*(14), 1641–1652.

Chen, Y., Ji, Y., & Guo, K. (2014). A reduced-order nonlinear sliding mode observer for vehicle slip angle and tyre forces. *Vehicle System Dynamics*, *52*(12), 1716–1728.

Cherouat, H., Braci, M., & Diop, S. (2005). Vehicle velocity, side slip angles and yaw rate estimation. *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.*, *1*, 349–354.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734. https://doi.org/10.3115/v1/D14-1179

Chowdhri, N., Ferranti, L., Iribarren, F. S., & Shyrokau, B. (2021). Integrated nonlinear model predictive control for automated driving. *Control Engineering Practice*, *106*, 104654. https://doi.org/https://doi.org/10.1016/j.conengprac.2020.104654

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Ding, N., Chen, W., Zhang, Y., Xu, G., & Gao, F. (2014). An extended Luenberger observer for estimation of vehicle sideslip angle and road friction. *International Journal of Vehicle Design*, *66*(4), 385–414.

Doumiati, M., Victorino, A. C., Charara, A., & Lechner, D. (2010). Onboard real-time estimation of vehicle lateral tire–road forces and sideslip angle. *IEEE/ASME Transactions on Mechatronics*, *16*(4), 601–614.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7).

Dugoff, H., Fancher, P. S., & Segel, L. (1969). *Tire performance characteristics affecting vehicle response to steering and braking control inputs* (tech. rep.).

Durbin, J., & Koopman, S. J. (2012). *Time series analysis by state space methods*. Oxford university press.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *international conference on machine learning*, 1050–1059.

Ghosh, J., Tonoli, A., & Amati, N. (2018). *A deep learning based virtual sensor for vehicle sideslip angle estimation: Experimental results* (tech. rep.). SAE Technical Paper.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* (Vol. 1). MIT press Cambridge.

Goodfellow, I., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Graves, A., Mohamed, A. R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *2013 IEEE international conference on acoustics, speech and signal processing*, 6645–6649.

Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, *14*(8), 2.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. *International conference on learning and intelligent optimization*, 507–523.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*, 448–456.

Julier, S. J., & Uhlmann, J. K. (1997). New extension of the kalman filter to nonlinear systems. *Signal processing, sensor fusion, and target recognition VI*, *3068*, 182–193.

Kazemi, S. M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P., & Brubaker, M. (2019). Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*.

Keller, M., Haß, C., Seewald, A., & Bertram, T. (2015). A model predictive approach to emergency maneuvers in critical traffic situations. *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 369–374.

Kerst, S., Shyrokau, B., & Holweg, E. (2016). Reconstruction of wheel forces using an intelligent bearing. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, *9*(2016-01-0092), 196–203.

Kerst, S., Shyrokau, B., & Holweg, E. (2019). A model-based approach for the estimation of bearing forces and moments using outer ring deformation. *IEEE Transactions on Industrial Electronics*, *67*(1), 461–470.

Kim, D., Min, K., Kim, H., & Huh, K. (2020). Vehicle sideslip angle estimation using deep ensemble-based adaptive Kalman filter. *Mechanical Systems and Signal Processing*, *144*, 106862.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2016). Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*.

Liu, W., Liu, W., Ding, H., & Guo, K. (2012). Side-slip angle estimation for vehicle electronic stability control based on sliding mode observer. *Proceedings of 2012 International Conference on Measurement, Information and Control*, *2*, 992–995.

Lu, Q., Gentile, P., Tota, A., Sorniotti, A., Gruber, P., Costamagna, F., & De Smet, J. (2016). Enhancing vehicle cornering limit through sideslip and yaw rate control. *Mechanical Systems and Signal Processing*, *75*, 455–472.

Mazzilli, V., Ivone, D., De Pinto, S., Pascali, L., Contrino, M., Tarquinio, G., Gruber, P., & Sorniotti, A. (2021). On the benefit of smart tyre technology on vehicle state estimation. *Vehicle System Dynamics*, 1–26.

Melzi, S., & Sabbioni, E. (2011). On the vehicle sideslip angle estimation through neural networks: Numerical and experimental results. *Mechanical Systems and Signal Processing*, *25*(6), 2005–2019.

Naets, F., van Aalst, S., Boulkroune, B., El Ghouti, N., & Desmet, W. (2017). Design and experimental validation of a stable two-stage estimator for automotive sideslip angle and tire parameters. *IEEE Transactions on Vehicular Technology*, *66*(11), 9727–9742.

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 2018). Determination press San Francisco.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *International conference on machine learning*, 1310–1318.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M.,

& Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peng, B., Yao, K., Jing, L., & Wong, K.-F. (2015). Recurrent neural networks with external memory for spoken language understanding. *Natural Language Processing and Chinese Computing*, 25–35.

Rajamani, R. (2011). *Vehicle dynamics and control*. Springer Science & Business Media.

Revach, G., Shlezinger, N., Ni, X., Escoriza, A. L., van Sloun, R. J., & Eldar, Y. C. (2021). Kalmannet: Neural network aided kalman filtering for partially known dynamics. *arXiv preprint arXiv:2107.10043*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization? *Advances in Neural Information Processing Systems*, 2483–2493.

Segel, L. (1956). Theoretical prediction and experimental substantiation of the response of the automobile to steering control. *Proceedings of the Institution of Mechanical Engineers: Automobile Division*, *10*(1), 310–330.

Sen, S., Chakraborty, S., & Sutradhar, A. (2015). Estimation of tire slip-angles for vehicle stability control using Kalman filtering approach. *2015 International Conference on Energy, Power and Environment: Towards Sustainable Growth (ICEPE)*, 1–6.

Sieberg, P. M., Blume, S., & Schramm, D. (2021). Side-slip angle estimation by artificial neural networks for vehicle dynamics control applications. *AmE 2021-Automotive meets Electronics; 12th GMM-Symposium*, 1–6.

Stephant, J., Charara, A., & Meizel, D. (2007). Evaluation of a sliding mode observer for vehicle sideslip angle. *Control Engineering Practice*, *15*(7), 803–812.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Teye, M., Azizpour, H., & Smith, K. (2018). Bayesian uncertainty estimation for batch normalized deep networks. *International Conference on Machine Learning*, 4907–4916.

Torun, H. M., Swaminathan, M., Davis, A. K., & Bellaredj, M. L. F. (2018). A global bayesian optimization algorithm and its application to integrated system design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *26*(4), 792–802.

Tuononen, A. J. (2009). Vehicle lateral state estimation based on measured tyre forces. *Sensors*, *9*(11), 8761–8775. https://doi.org/10.3390/s91108761

van Aalst, S. (2020). Virtual sensing for vehicle dynamics: A model-based approach for indirect measurement of the vehicle motion states and tyre forces.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Viehweger, M., Vaseur, C., van Aalst, S., Acosta, M., Regolin, E., Alatorre, A., Desmet, W., Naets, F., Ivanov, V., Ferrara, A., et al. (2021). Vehicle state and tyre force estimation: Demonstrations and guidelines. *Vehicle System Dynamics*, *59*(5), 675–702.

Wan, E. A., & Van Der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, 153–158.

Wu, N., Green, B., Ben, X., & O'Banion, S. (2020). Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*.

Xu, N., Huang, Y., Askari, H., & Tang, Z. (2021). Tire slip angle estimation based on the intelligent tire technology. *IEEE Transactions on Vehicular Technology*, *70*(3), 2239–2249.

Xu, Y., & Goodacre, R. (2018). On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of Analysis and Testing*, *2*.

A
AVEC paper

AVEC'22

Model-Based vs Data-Driven Estimation of Vehicle Sideslip Angle and Benefits of Tyre Force Measurements

A. Bertipaglia[*], D. de Mol[*], M. Alirezaei[+], R. Happee[*] & B. Shyrokau[*]
[*]Delft University of Technology, Delft, The Netherlands.
[+]Eindhoven University of Technology, Eindhoven, The Netherlands.
E-mail: A.Bertipaglia@tudelft.nl

Topics/ Identification and Estimation, Sideslip Angle Estimation

January 31, 2022

This paper compares model-based and data-driven approaches for sideslip angle estimation and analyses the benefits of adding measured tyre forces. The analysed model-based approaches are the extended Kalman filter and the unscented Kalman filter, in which tyre forces are incorporated as measurements in the observation model. An adaptive covariance matrix is introduced to minimize the tyre model mismatch during evasive manoeuvres. The data-driven approaches focus on the usage of a feed-forward and a recurrent neural network. Using the large-scale experimental dataset covering 216 various vehicle manoeuvres, we demonstrate a significant improvement in accuracy using data-driven approaches compared to model-based ones. Results show that tyre force measurements improve the performance of both model-based and data-driven approaches.

1 INTRODUCTION

The ability to estimate the vehicle sideslip angle in real-time is essential to strengthening the performance of active vehicle control. A large variety of driving conditions, such as normal or at the handling limits, steady-state or transient manoeuvres, makes the estimation challenging. Especially, the highly nonlinear behaviour of tyres leads to a substantial limitation in tyre model accuracy. Many different solutions for vehicle sideslip angle estimation have been proposed in the past. They could be split into two different approaches, such as model-based and data-driven [1-4]. Both approaches allow the incorporation of tyre forces, measured by load sensing bearings [5] or smart tyres [3]. This paper provides a fair evaluation of the accuracy of each approach and aims to quantify the benefits of adding tyre force measurements for each of the proposed solutions. Model-based and data-driven approaches have already been analysed in surveys; however, the comparison is based on limited simulation data [6, 7]. Furthermore, an additional research goal is related to tyre force measurements providing different benefits, and it is not previously addressed in the literature. This paper proposes a comparison based on the large-scale real-world experimental dataset. It considers standard vehicle dynamics manoeuvres, e.g. double lane change, slalom, random steer, J-turn, spiral, braking in a turn, and steady-state circular tests, together with recorded laps at the Papenburg track. The dataset contains a great diversity of driving situations. Statistical outlier removal is applied to remove extreme outliers. The dataset consists of 216 manoeuvres which correspond to 2 hrs of driving. The distribution of the sideslip angle and lateral acceleration is represented in Figure 1. The lower availability of high sideslip angle data points is due to the difficulties of driving in such conditions, and it influences the training of the data-driven approach. An extended Kalman Filter (EKF) and a unscented Kalman Filter (UKF) using a single-track vehicle model are implemented for model-based approach, while a Feed-Forward Neural Network (FFNN) and a Recurrent Neural Network (RNN) are considered for data-driven approach. During the development of the estimators, the longitudinal velocity is assumed known as in [1, 2].

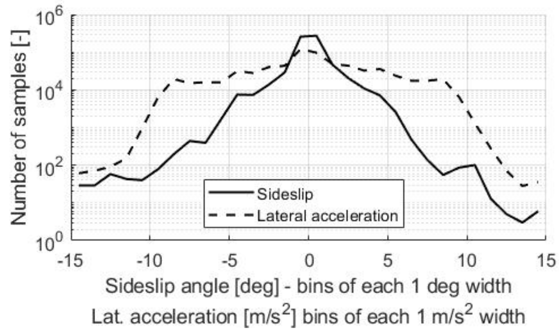

Figure 1: The distribution of measured sideslip angle and lateral acceleration data points. The lateral acceleration distribution is wide, while the sideslip angle distribution focuses on small values.

The contribution of this paper is twofold. First, the accuracy of model-based and data-driven approaches are fairly compared using the mentioned dataset, representing various driving manoeuvres. Second, for each proposed approach, the benefit of adding tyre force measurements is analysed and assessed

2 SIDESLIP ANGLE ESTIMATION

2.1 Model-based approach

The single-track model with tyre axle forces computed by the Dugoff tyre is chosen as the vehicle model. The states are the lateral velocity v_y and the yaw rate while, the inputs are the longitudinal velocity v_x and the longitudinal acceleration a_x. The measurement vector is composed of the lateral acceleration a_y, and the tyre force measurements when they are considered. The process and measurement noise covariance matrix are tuned using a two-stage Bayesian optimisation [8][6]. The measurement noise matrix is continuously adapted according to the absolute difference between the estimated and the measured tyre forces to improve the accuracy of the estimation of vehicle sideslip angle vehicle sideslip angle estimation [2][1].

2.2 Data-driven approach

FFNN is considered because it is the most straightforward implementation, but it does not have any time dependence, which can could improve the overall estimation performance. Thus, a RNN is also considered to evaluate the predictive power level of previous measurements on the current time-step. The performance of FFNN and RNN is strongly related to the training dataset. The manoeuvres used in the training dataset are not reused in the test set to avoid overfitting. Two different measurement sets are considered: the first, considers measurements just coming from the inertial measurement unit (IMU), i.e. a_y, a_x, , steering wheel angle and v_x, while the second dataset also considers the measured tyre forces in all the three directions. FFNN and RNN are both trained with a back-propagation algorithm using Adam optimiser.

3 RESULTS

The final results, Table 1, are evaluated on a validation dataset composed of 23 manoeuvres taken from the available dataset described in the Introduction. The key performance indicators (KPI) are the root mean square error (RMSE) and the maximum error (ME). On average, the UKF is more accurate than EKF, even if their performance is very close when using tyre force measurements. FFNN and RNN have very similar performance, and the difference is due to the optimisation routine rather than the different architecture. Their performance is even slightly worse than the model-based approach due to a very high error when the sideslip angle reaches 5 degrees. Data-driven approaches outperform model-based ones when the dataset includes tyre force measurements. A comparison between the RNN and UKF, both with tyre force measurements, is shown in Figure 2. The selected manoeuvre is a J-turn at the handling limit with vehicle active control is switched off.

4 REFERENCES

[1] F. Cheli, D. Ivone, and E. Sabbioni, "Smart Tyre Induced Benefits in Sideslip Angle and Friction Coefficient Estimation," Sensors and Instrumentation, Volume 5. pp. 73-83.
[2] A. Bonfitto, S. Feraco, A. Tonoli, and N. Amati, "Combined regression and classification artificial neural networks for sideslip angle estimation and road condition identification," Vehicle System Dynamics, vol. 58, no. 11, pp. 1766-1787, 2020/11/01, 2020.
[3] V. Mazzilli, D. Ivone, S. De Pinto, L. Pascali, M. Contrino, G. Tarquinio, P. Gruber, and A. Sorniotti, "On the benefit of smart tyre technology on vehicle state estimation," Vehicle System Dynamics, pp. 1-26, 2021.
[4] N. Xu, Y. Huang, H. Askari, and Z. Tang, "Tire slip angle estimation based on the intelligent tire technology," IEEE Transactions on Vehicular Technology, vol. 70, no. 3, pp. 2239-2249, 2021.
[5] S. Kerst, B. Shyrokau, and E. Holweg, "A Model-Based Approach for the Estimation of Bearing Forces and Moments Using Outer Ring Deformation," IEEE Transactions on Industrial Electronics, vol. 67, no. 1, pp. 461-470, 2020.
[6] D. Chindamo, B. Lenzo, and M. Gadola, "On the Vehicle Sideslip Angle Estimation: A Literature Review of Methods, Models, and Innovations," Applied Sciences, vol. 8, no. 3, pp. 355, 2018.
[7] J. Liu, Z. Wang, L. Zhang, and P. Walker, "Sideslip angle estimation of ground vehicles: a comparative study," IET Control Theory and Applications, vol. 14, no. 20, pp. 3490-3505, 2020.
[8] H. M. Torun, M. Swaminathan, A. K. Davis, and M. L. F. Bellaredj, "A global Bayesian optimization algorithm and its application to integrated system design," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 4, pp. 792-802, 2018.

KPI	EKF	EKF + tyre meas	UKF	UKF + tyre meas	FFNN	FFNN + tyre meas	RNN	RNN + tyre meas
RMSE [deg]	0.391	0.350	0.373	0.347	0.360	0.221	0.390	0.225
ME [deg]	1.012	0.858	0.867	0.790	1.419	0.892	1.649	0.783

Table 1: Average results for the sideslip angle estimation of the validation dataset composed by 23 manoeuvres.