

# Road Surface Estimation Using Machine Learning

by

Brian Mao

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Master of Mathematics  
in  
Applied Mathematics

Waterloo, Ontario, Canada, 2023

© Brian Mao 2023

## **Author's Declaration**

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

Vehicle motion control systems are present on commercial vehicles to improve safety and driving comfort. Many of these control systems could be further improved given accurate online information about the road condition to accommodate for driving under poor weather conditions such as icy roads or heavy rain. However, sensors for direct friction measurement are not present on commercial vehicles due to production costs. Hence, it is beneficial to incorporate an online estimation scheme for road surface classification.

This thesis focuses on investigating two fundamentally different machine learning-based methods for road surface classification. The first is an artificial neural network that provides a global function approximation of the underlying dynamics. In particular, Long Short-Term Memory (LSTM) units are used to capture temporal relationships within the training data and to mitigate the vanishing gradient problem. The second is an instance-based learning method referred to as Nadaraya-Watson Kernel Regression, where local function approximations are generated around the input data.

Results indicated that both machine learning-based methods were able to classify road conditions to a reasonable degree of accuracy after tuning associated hyperparameters. However, each method has different benefits and drawbacks. The LSTM network model was capable of making accurate predictions on inputs drastically different from data points within the training data set, was generally more accurate on predictions associated with new driving maneuvers, required less storage for implementation, and had relatively short prediction times. Conversely, the Nadaraya-Watson Kernel Regression model was generally more accurate at making predictions on inputs that were very similar to data points within the training data set, did not require any training time to incorporate newly collected data into the model, and generated predictions that were more easily explainable.

## **Acknowledgements**

First and foremost, I'd like to thank my supervisors Professor Kirsten Morris, and Professor Amir Khajepour for their guidance and support. This thesis certainly would not have been possible without them. I'd also like to thank all of my wonderful friends across the Mathematics and Engineering faculties throughout my time at the University of Waterloo. They certainly made the entire experience far more enjoyable than it would have been otherwise. Finally, I'd also like to thank my other friends and family for all of their encouragement and support over the years.

# Table of Contents

List of Figures	vii
List of Tables	xi
<b>1 Introduction</b>	<b>1</b>
<b>2 Background and Literature Review</b>	<b>4</b>
2.1 Vehicle Parameters . . . . .	4
2.2 Classification of Road Surfaces . . . . .	7
2.3 Regression Application For Friction Coefficient Estimations . . . . .	8
2.4 Model-Based Tire-Road Friction Estimation . . . . .	10
2.5 Machine Learning-Based Approaches for Other Vehicle Parameters . . . . .	11
<b>3 Artificial Neural Networks</b>	<b>13</b>
3.1 Feedforward Neural Networks . . . . .	13
3.2 Training Artificial Neural Networks . . . . .	17
3.3 Recurrent Neural Networks . . . . .	21
3.4 Long Short-Term Memory . . . . .	24
3.5 Friction Estimation on an Inverted Pendulum Using LSTM . . . . .	26

<b>4</b>	<b>Instance-Based Learning</b>	<b>33</b>
4.1	K-Nearest Neighbors . . . . .	33
4.2	Nadaraya-Watson Kernel Regression . . . . .	35
4.3	Implementation of Nadaraya-Watson Kernel Regression . . . . .	44
4.4	Friction Estimation on an Inverted Pendulum using Nadaraya-Watson Kernel Regression . . . . .	45
<b>5</b>	<b>Road Surface Classification</b>	<b>47</b>
5.1	Feature Selection and Data Sets . . . . .	48
5.2	LSTM Network Architecture and Estimation Results . . . . .	53
5.3	Nadaraya-Watson Kernel Regression Hyperparameter Selection and Estimation Results . . . . .	62
5.4	Comparison of Approaches . . . . .	70
<b>6</b>	<b>Conclusions and Future Work</b>	<b>74</b>
6.1	Conclusions . . . . .	74
6.2	Future Work . . . . .	76
	<b>References</b>	<b>77</b>
	<b>APPENDIX</b>	<b>83</b>
	<b>A Driving Maneuvers</b>	<b>84</b>
	<b>B Measurement Data from Test Data Set</b>	<b>90</b>

# List of Figures

1.1	Control System Block Diagram Containing Observer . . . . .	2
2.1	Longitudinal and Lateral Dynamics and Kinematics . . . . .	5
2.2	Tire Force and Moment Components (Fig. 3, [9]) . . . . .	5
2.3	Slip Angle Visualization Modified from (Fig. 2.1, [25]) . . . . .	6
2.4	Vehicle Coordinate System (Fig. 1, [20]) . . . . .	7
3.1	Single Artificial Neural Network Node (Modified from Fig. 1, [49]) . . . . .	14
3.2	Example of a Feedforward Artificial Neural Network . . . . .	14
3.3	Gradient Descent Example in 1D (Fig 4.1, [10]) . . . . .	17
3.4	Example of Notation Used to Denote Weights (Fig. 1, [27]) . . . . .	18
3.5	Unfolded Recurrent Neural Network (Modified from Fig. 1, [7]) . . . . .	21
3.6	Various Recurrent Neural Network Architectures (Fig. 11.1, [1]) . . . . .	22
3.7	Depiction of Backpropagation Through Time (Fig. 1, [23]) . . . . .	23
3.8	LSTM Architecture (Modified from Fig. 8.5, [14]) . . . . .	24
3.9	Inverted Pendulum System (Fig. 1, [3]) . . . . .	26
3.10	Sample Angle Measurement Data . . . . .	29
3.11	Sample Position Measurement Data . . . . .	29
3.12	Sample Voltage Measurement Data . . . . .	30
3.13	Visualization of Processed Data for Friction Force Estimation . . . . .	30
3.14	LSTM Network Architecture Layout for Friction Estimation on Inverted Pendulum System . . . . .	31

3.15	Friction Estimation using LSTM on Inverted Pendulum System . . . . .	32
4.1	Visual Depiction of K-NN Algorithm . . . . .	35
4.2	Visual Depiction of Linear Regression . . . . .	36
4.3	Nadaraya-Watson Estimation with Well-Tuned Bandwidth, $\sigma = 1.9$ . . . . .	42
4.4	Nadaraya-Watson Estimation with Overfitting, $\sigma = 0.5$ . . . . .	43
4.5	Nadaraya-Watson Estimation with Underfitting, $\sigma = 4$ . . . . .	43
4.6	Friction Estimation using Nadaraya-Watson Kernel Regression on Inverted Pendulum System . . . . .	45
5.1	Overview of Road Classification Models . . . . .	47
5.2	Visualization of Processed Data for Road Surface Classification . . . . .	49
5.3	Sample CarSim Simulation . . . . .	50
5.4	LSTM Network Architecture Layout . . . . .	54
5.5	LSTM Network Estimation for 80 km/h Increasing Steer (Ice) . . . . .	55
5.6	LSTM Network Estimation for Acceleration and Brake (Ice) . . . . .	55
5.7	LSTM Network Estimation for 240 km/h Straight Driving (Ice) . . . . .	56
5.8	LSTM Network Estimation for 80 km/h Brake In Turn (Ice) . . . . .	56
5.9	LSTM Network Estimation for 85 km/h Rollover Steer (Wet) . . . . .	57
5.10	LSTM Network Estimation for 220 km/h S Turn with Clothoids (Wet) . . . . .	58
5.11	LSTM Network Estimation for 115 km/h Side Drift (Wet) . . . . .	58
5.12	LSTM Network Estimation for 145 km/h Sine Sweep Steer (Wet) . . . . .	59
5.13	LSTM Network Estimation for 65 km/h Double Lane Change (Dry) . . . . .	59
5.14	LSTM Network Estimation for Increasing Speed (Dry) . . . . .	60
5.15	LSTM Network Estimation for 135 km/h Side Drift (Dry) . . . . .	61
5.16	LSTM Network Estimation for 95 km/h S Turn with Clothoids (Dry) . . . . .	61
5.17	Nadaraya-Watson Estimation for 80 km/h Increasing Steer (Ice) . . . . .	63
5.18	Nadaraya-Watson Estimation for Acceleration and Brake (Ice) . . . . .	64
5.19	Nadaraya-Watson Estimation for 240 km/h Straight Driving (Ice) . . . . .	64



5.20	Nadaraya-Watson Estimation for 80 km/h Brake In Turn (Ice) . . . . .	65
5.21	Nadaraya-Watson Estimation for 85 km/h Rollover Steer (Wet) . . . . .	65
5.22	Nadaraya-Watson Estimation for 220 km/h S Turn with Clothoids (Wet) . . . . .	66
5.23	Nadaraya-Watson Estimation for 115 km/h Side Drift (Wet) . . . . .	66
5.24	Nadaraya-Watson Estimation for 145 km/h Sine Sweep Steer (Wet) . . . . .	67
5.25	Nadaraya-Watson Estimation for 65 km/h DLC (Dry) . . . . .	67
5.26	Nadaraya-Watson Estimation for Increasing Speed (Dry) . . . . .	68
5.27	Nadaraya-Watson Estimation for 135 km/h Side Drift (Dry) . . . . .	68
5.28	Nadaraya-Watson Estimation for 95 km/h S Turn with Clothoids (Dry) . . . . .	69
A.1	Double Lane Change Driving Maneuver . . . . .	84
A.2	Sine Sweep Steer Driving Maneuver . . . . .	85
A.3	Rollover Steer Driving Maneuver . . . . .	85
A.4	S Turn with Clothoids Driving Maneuver . . . . .	86
A.5	Straight Driving Maneuver . . . . .	86
A.6	Single Lane Change Maneuver . . . . .	87
A.7	Increasing Steer Driving Maneuver . . . . .	87
A.8	Increasing Speed Driving Maneuver . . . . .	88
A.9	Brake In Turn Driving Maneuver . . . . .	88
A.10	Throttle Command Throughout Accelerate and Brake Driving Maneuver . . . . .	89
A.11	Brake Command Throughout Accelerate and Brake Driving Maneuver . . . . .	89
B.1	Measurement Data for 80 km/h Increasing Steer (Ice) Simulation . . . . .	91
B.2	Measurement Data for Accelerate and Brake (Ice) Simulation . . . . .	92
B.3	Measurement Data for 240 km/h Straight Driving (Ice) Simulation . . . . .	93
B.4	Measurement Data for 80 km/h Brake In Turn (Ice) Simulation . . . . .	94
B.5	Measurement Data for 85 km/h Rollover Steer (Wet) Simulation . . . . .	95
B.6	Measurement Data for 220 km/h S Turn with Clothoid (Wet) Simulation . . . . .	96

B.7	Measurement Data for 115 km/h Side Drift (Wet) Simulation . . . . .	97
B.8	Measurement Data for 145 km/h Sine Sweep Steer (Wet) Simulation . . . . .	98
B.9	Measurement Data for 65 km/h Double Lane Change (Dry) Simulation . . . . .	99
B.10	Measurement Data for Increasing Speed (Dry) Simulation . . . . .	100
B.11	Measurement Data for 135 km/h Side Drift (Dry) Simulation . . . . .	101
B.12	Measurement Data for 95 km/h S Turn with Clothoid (Dry) Simulation . . . . .	102

# List of Tables

2.1	Classification of Wheel States . . . . .	6
3.1	Common Activation Functions . . . . .	16
3.2	Measurements Available from Inverted Pendulum System . . . . .	27
3.3	Inverted Pendulum System Parameter Values (Tab. 1 and 2, [3]) . . . . .	27
3.4	LSTM Network Architecture for Friction Estimation of Inverted Pendulum System . . . . .	31
3.5	LSTM Network Performance Metrics on Inverted Pendulum System . . . . .	32
4.1	Training Data for K-NN Example . . . . .	34
4.2	Common Kernel Functions . . . . .	38
4.3	Nadaraya-Watson Kernel Regression Performance Metrics on Inverted Pendulum System . . . . .	46
5.1	Relevant and Commonly Measurable/Estimated Features Among Commercial Vehicles . . . . .	48
5.2	Labels Associated with Each Classification Surface . . . . .	49
5.3	Collected Data Involving Wet ( $\mu = 0.5$ ) [Left] and Dry ( $\mu = 0.9$ )[Right] Surfaces . . . . .	51
5.4	Collected Data Involving Icy ( $\mu = 0.1$ ) Surfaces . . . . .	52
5.5	Testing Data Set Simulations . . . . .	52
5.6	Searchable Space of Investigated Hyperparameters . . . . .	54
5.7	LSTM Network Architecture Details . . . . .	54

5.8	Lookup Table for Surface Classification (Tab. 6, [43]) . . . . .	63
5.9	Summary of Accuracy Measures . . . . .	69
5.10	Comparison of Methods Summary . . . . .	73

# Chapter 1

## Introduction

Many control systems are present on modern vehicles to make driving both safer and easier. Examples include Anti-lock Braking Systems (ABS), Electronic Stability Control (ESC), Traction Control Systems (TCS), Lane Keep Assist (LKA), and Adaptive Cruise Control (ACC). Robust vehicle control systems are also a requirement to achieve fully autonomous driving, which includes driving under poor weather conditions such as heavy rain or icy roads.

Estimation is a mathematical field in which the internal state or parameters of a dynamical model are determined through observations of the inputs and outputs of the system. Estimation techniques can be very useful in controlling a system where the states or parameters cannot be directly measured. The estimator, also commonly referred to as the observer, can be viewed as a separate system that determines the state and/or parameters of the system. The output of the estimator is used as an input to the controller as shown in Figure 1.1.

Vehicle motion control systems can become more adaptable and achieve better performance with accurate online information of the road surface conditions. Other parameters that would typically require expensive sensors to measure, such as longitudinal and lateral tire forces, may also be estimated using this information. Therefore, it is beneficial to develop an estimation scheme to accurately classify road surface conditions.

Existing model-based observers that estimate tire-road friction coefficients can be found in the literature. However, simplifications of the true underlying dynamics are consistently made as with most mathematical models. This results in some uncaptured behaviour as demonstrated by discrepancies between generated estimates and experimental results.

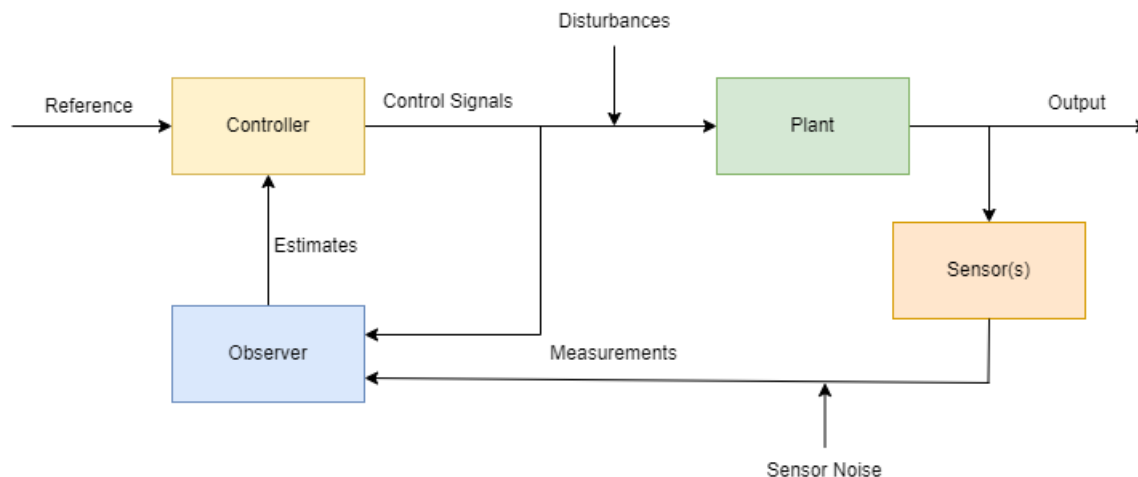


Figure 1.1: Control System Block Diagram Containing Observer

The focus of this thesis will be on applying machine learning-based techniques for road surface estimation. Techniques from the field of machine learning have traditionally been applied towards solving classification and regression problems in relevant areas of computer science such as computer vision, speech recognition, translation, and many others. However, promising applications of machine learning techniques continue to increase to other domains including control and estimation.

Machine learning-based approaches for estimation problems can be divided into two main categories. The first is to find a nonlinear function that approximates the target function globally across its entire domain. An example would be a neural network where the weights and biases are adjusted to best fit the entire training data set. The second is to find a local approximation of the target function each time a prediction is to be made. An example would be the K-Nearest Neighbours algorithm where the final prediction is based entirely on the data points from the training data that are closest to the input.

A literature review of machine learning and model-based approaches that have previously been applied towards road surface classification and tire-road friction estimation is in Chapter 2 of this thesis. Other applications of machine learning towards vehicle parameter estimation are also included. A brief overview of vehicle dynamics relevant to road surface estimation will also be reviewed in this chapter.

Chapter 3 primarily focuses on finding a global approximation of the target function. An emphasis is placed on recurrent neural networks, which capture temporal relationships within the training data. Issues with traditional recurrent neural networks, including

the vanishing gradient problem, are discussed in detail. This motivates the use of Long Short-Term Memory (LSTM), which is another recurrent neural network architecture. In addition, techniques from this chapter are applied to estimate friction forces on an inverted pendulum system.

A query is defined as a model prediction request for a particular input. Chapter 4 mainly focuses on the approach of finding local function approximations around each new query. A review of the K-Nearest Neighbours algorithm is initially presented. Next, an extension of this technique based on statistical literature known as Nadaraya-Watson Kernel Regression, is introduced. It is important that algorithms based on local approximations are implemented efficiently in hardware present onboard a vehicle. This is to provide online predictions at a low latency for vehicle control systems to react to sudden changes in the road condition. Hence, implementation details of the Nadaraya-Watson Kernel Regression method for computational efficiency are also briefly discussed. The chapter concludes with an implementation of the presented techniques on an inverted pendulum system, similar to the previous chapter.

Results of applying both methods focused on a global function approximation and a local approximation about a new query are presented in Chapter 5. A LSTM network is trained and implemented as a global function approximation approach to classify road surface conditions. Similarly, the Nadaraya-Watson Kernel Regression technique is implemented and tested. A quantitative comparison between the two methods is presented with an emphasis on characteristic benefits and drawbacks. Performance metrics include an accuracy calculation, training time, data storage required, and prediction times on unseen data.

Final conclusions from the comparative study as well as potential future work are summarized in Chapter 6.

# Chapter 2

## Background and Literature Review

The focus of this chapter is on reviewing previous works on road surface classification and tire-road friction estimation using both model-based observers and machine learning. In addition, the estimation of other vehicle parameters involving machine learning will also be reviewed. The chapter begins with a summary of various vehicle parameters commonly referenced in literature. Next, machine learning-based methods for road surface estimation are reviewed as two different problems types. Firstly, it is posed as a classification problem where the goal is to assign a categorical label as the final prediction. An example includes labeling the road surface as dry, wet, or ice. Secondly, the estimation is often posed as a regression problem in which the final prediction is a numerical value. In this case, the typical goal is to estimate the coefficient of friction between the tires and the road. Afterwards, model-based tire-road friction estimation schemes are reviewed. Finally, applications of machine learning towards the estimation of other vehicle parameters and states are covered.

### 2.1 Vehicle Parameters

This section provides a basic overview regarding various vehicle parameters. Many of these parameters are used as features for machine learning-based models throughout many previous works in literature.

Vehicle modelling is composed of two main categories, which are coupled. The first is longitudinal kinematics and dynamics, and the second is lateral kinematics and dynamics. Longitudinal refers to the forwards and backwards motion of the vehicle and is controlled



by both the throttle and brake pedal inputs. Lateral refers to the sideways motion of the vehicle and is controlled by the steering wheel input. The inputs and outputs associated with the coupled longitudinal and lateral kinematics and dynamics are shown in Figure 2.1. The most relevant vehicle parameters governing the kinematics and dynamics differ between longitudinal and lateral motion.

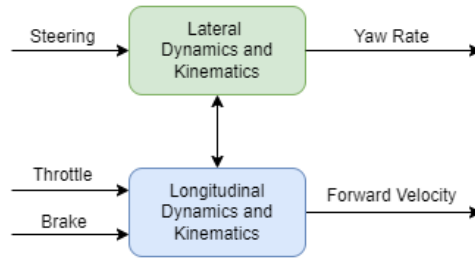


Figure 2.1: Longitudinal and Lateral Dynamics and Kinematics

Significant tire force and moment components are displayed in Figure 2.2. The longitudinal tire force  $F_x$  is the tire force component along the x-axis. The lateral tire force  $F_y$  is the tire force component along the y-axis, which is perpendicular to the longitudinal tire force. In addition, the rolling resistance moment  $M_y$  is the moment acting on the tire along the axis of wheel rotation.

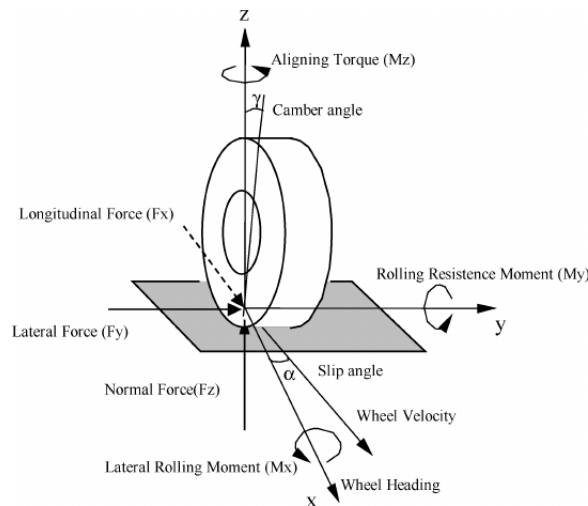


Figure 2.2: Tire Force and Moment Components (Fig. 3, [9])

**Definition 1 (Slip Ratio)** *The slip ratio is defined as*

$$S = \frac{\omega r_e - V}{V}$$

where  $\omega$  is the wheel angular speed,  $r_e$  is the effective tire radius, and  $V$  is the vehicle forward velocity.

The slip ratio is a significant parameter in longitudinal vehicle motion modelling. Table 2.1 summarizes the current state of the vehicle based on the tire velocity,  $\omega r_e$ , relative to the vehicle forward velocity.

Parameter Value	Condition
$\omega r_e < V$	Wheels are Skidding
$\omega r_e > V$	Wheels are Spinning
$\omega r_e = 0$	Wheels are Locked

Table 2.1: Classification of Wheel States

Skidding occurs during normal braking to decelerate the vehicle. Spinning may occur while accelerating the vehicle from a slippery surface such as an icy road. Wheels can become locked during heavy braking where the vehicle loses traction entirely.

**Definition 2 (Slip Angle)** *The slip angle is defined as*

$$\alpha = -\arctan\left(\frac{V_y}{V_x}\right)$$

where  $V_y$  is the lateral velocity, and  $V_x$  is the forward velocity.

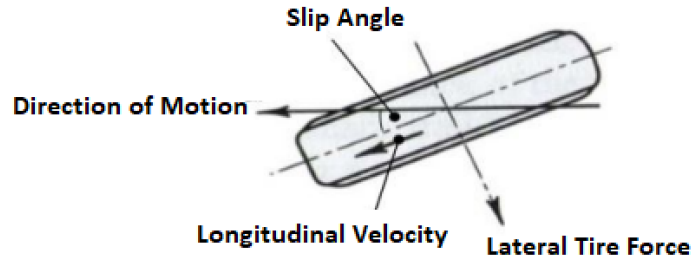


Figure 2.3: Slip Angle Visualization Modified from (Fig. 2.1, [25])

Slip angles can also be interpreted as the difference between the vehicle heading and the true heading, which is the direction the vehicle body points towards. The slip angle is one of the most relevant parameters in lateral vehicle motion modelling and is displayed in Figure 2.3. It is equal to zero under many normal driving conditions such as driving along a straight road. However, it is often non-zero when performing many lateral maneuvers such as a harsh lane change maneuver. Lateral tire forces are also induced by non-zero slip angles.

Another important vehicle state in lateral vehicle motion modelling is the yaw rate. It refers to the rate of rotation among the yaw axis as displayed in Figure 2.4. Note that the roll rate and roll angle are also relevant when modelling vehicles on banked surfaces such as a freeway entry ramp.

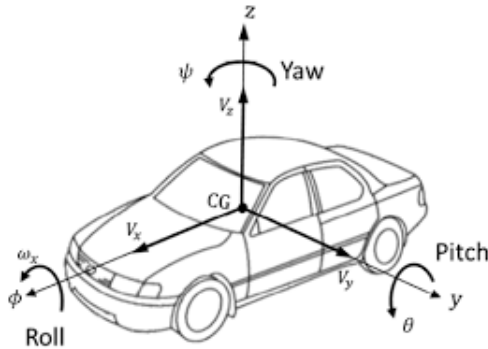


Figure 2.4: Vehicle Coordinate System (Fig. 1, [20])

## 2.2 Classification of Road Surfaces

Support vector machines (SVM) and various types of neural network architectures are among the most common machine learning techniques applied towards road surface classification. Applications of these techniques are examined throughout this section.

Kim et al. conducted a comparison between a trained support vector machine and a deep neural network (DNN) for road surface classification [18]. The study was posed as a binary classification problem between the labels of wet and dry. Only three feature vectors consisting of the slip ratio, normalized longitudinal force, and longitudinal acceleration were used by both the SVM and DNN for classification. The study found that both the SVM and DNN performed similarly classifying dry roads with the chosen hyperparameters.

However, the DNN performed better at classifying wet roads. SVMs were also used by Ward et al. to classify terrain data on passenger vehicles [52]. The online classification method used measured vibration data to identify terrain features such as the presence of potholes.

Convolutional neural networks (CNN) were also trained for road surface classification throughout various studies. Kim et al. used data collected from reflected ultrasonic signals paired with an ultrasonic transmitter [19]. Assigned road condition labels consisted of asphalt, cement, dirt, ice, marble, paint, snow, and water. Šabanovič et al. used data collected from a camera to train a deep neural network consisting of five convolutional layers [38]. The classification labels included dry and wet asphalt, dry and wet cobblestone, as well as dry and wet gravel. Similarly, Roychowdhury et al. used image data from a camera attached to the front of a car to train a CNN [37]. The network labelled images into the categories of dry asphalt, wet/water, slush, and snow/ice.

Finally, a different approach to road friction classification involving logistic regression, support vector machines, and artificial neural networks was developed by Panahandeh et al. [31]. The approach involved the collection of historical data from an entire fleet of cars rather than from a single car. Data regarding temporal and weather information was also incorporated as features. The final road friction estimates are provided over an entire geographical location throughout a certain time period instead of an instantaneous moment within a local spatial region around the tires.

## 2.3 Regression Application For Friction Coefficient Estimations

Road surface estimation is also commonly posed as a regression problem where the final prediction is a numerical value for the coefficient of friction between the tires and the road. Various works focused on applying machine learning towards estimating friction coefficients are reviewed in this section with an emphasis on neural networks.

An early attempt to estimate tire-road friction using machine learning was conducted by Pacejka et al. in 1998 [32]. A genetic algorithm was applied to optimize a feedforward neural network architecture. However, the technique was very computationally inefficient and infeasible for online estimation.

Song et al. identified three main categories of factors affecting the friction coefficient as vehicle, tire, and road parameters [45]. A standard backpropagation (BP) neural network consisting of 2 hidden layers containing 10 neurons each, was trained to output a numerical

value representing the tire-road friction coefficient. Inputs to the network included angular velocity, slip rate, yaw rate, longitudinal acceleration, lateral acceleration, and steering wheel angle. A two-layer radial basis function (RBF) neural network with Gaussian activation functions was used by Matusko et al. with the goal of estimating friction insensitive to modelling inaccuracies [24]. Relative velocity was used as the only input to the neural network with hidden layer weights trained offline. However, the weights of the final output layer of the neural network parameters were updated online to account for time-varying friction characteristics. A Lyapunov stability analysis was also performed to demonstrate convergence of the estimation. Ribeiro et al. used a single layered time-delayed neural network (TDNN) with 50 neurons to capture the temporal vehicle response due to changes in road friction [36]. An input-output relation is captured instead of an entire complex tire model to achieve real time performance. Inputs to the network included the slip angle, as well as longitudinal and lateral tire forces that were previously estimated using a Kalman filter. It is significant to note that the method allows for friction estimation at each individual tire rather than a holistic estimation for the entire vehicle.

Another study by Regolin et al. used several SVMs and nonlinear variants of the Kalman filter for road condition estimation [35]. First, the entire road friction curve, which consists of the friction coefficient as function of slip ratio, was classified for each test case using a SVM. Eight different labels were used for this classification including dry asphalt, wet asphalt, dry concrete, gravel, dry cobble, wet cobble, snow, and ice. The SVMs were trained offline and used to tune 3D look-up tables for online calculation. The next step was to use the classification from the SVMs to determine constraints on a Kalman filter based estimation for the tire-road friction coefficient. This step consisted of using both the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) to estimate the entire road friction curve. Another multi-step method that determined road friction indirectly from Kalman filters was proposed by Khaleghian et al. [17]. It was composed of a three step process where the normal load on the tires was first estimated using a 2-layer artificial neural network. The inputs included tire inflation pressure, peak time difference for circumferential acceleration, and wheel angular velocity. Next, both the tire forces and longitudinal velocity were estimated using Kalman filters. Finally, the friction coefficient was determined from the following expression involving previously estimated tire force components:

$$\rho = \frac{\sqrt{F_x^2 + F_y^2}}{F_z}.$$

A machine learning approach incorporating a measure for the estimation uncertainty was developed by Song et al. [44]. It used long-short term memory (LSTM) layers, which is a special type of recurrent neural network (RNN) layer that will be covered in more

detail in Chapter 3 of the thesis. The estimator consisted of a neural network containing both CNN and LSTM layers, as well as an ensemble to output a maximum road friction coefficient along with a corresponding uncertainty index. The ensemble consisted of five neural networks with the same architecture, and was used to quantify uncertainty in the predictions. Inputs were selected as variables that could be easily measured from on-board sensors for practicality. This included the steering wheel angle, steering wheel rate, slip ratio, wheel speed, brake cylinder pressure, throttle, acceleration, vehicle speed, engine torque, and yaw rate.

Finally, an estimation approach using Gaussian Process Regression (GPR) was developed by Berntorp [2]. All required inputs come from sensors typically installed onto production vehicles. The nonlinear function describing the dependence between wheel slip and tire friction is jointly estimated with the vehicle state in the presented approach. The method also combines a truncated basis function formulation of Gaussian processes under a particle filter framework.

## 2.4 Model-Based Tire-Road Friction Estimation

Other model-based techniques exist in literature to estimate tire-road friction without the use of any machine learning. This section reviews some significant previous works.

Rajamani et al. developed a real time estimation scheme to determine the friction coefficient for each individual wheel of a vehicle [33]. The overall approach consisted of first estimating the longitudinal tire force at each wheel, then estimating the slip ratio at each wheel, and finally using a recursive least squares parameter identification approach to determine the friction coefficient. A slip slope model was used in the study. Results were verified both experimentally and in simulation using CarSim. Similarly, Shao et al. used a recursive least squares approach [41]. However, a single track vehicle model was used instead along with a nonlinear observer for state estimation.

Gustafsson developed a tire-road friction coefficient estimation scheme for two-wheeled driven vehicles that only relied on wheel slip and other standard sensors present on commercial vehicles [12]. The estimation algorithm was based on the Kalman Filter with an additional change detection algorithm. Extensive experimental testing was also completed on a Volvo 850 GLT test car.

Villagra et al. developed an estimation scheme that was not an asymptotic observer-based estimation [51]. Instead, a combination of diagnostic tools and filtering techniques

for estimating derivatives of noisy data was presented. However, the proposed estimation scheme still required a reliable estimate of slip angles and lateral tire forces as a prerequisite.

Another real time tire-road friction estimation scheme based on the bicycle model and Pacejka tire model was proposed by Santani et al. [39]. The bicycle model was used to evaluate the state of the vehicle, and the Pacejka tire model was based on a slip-slope approach to evaluate the potential friction. The overall estimation scheme also began with sensor readings commonly found on production vehicles such as longitudinal speed, throttle and brake pedal positions, as well as the angular velocity of each wheel. Results were gathered through simulation using a vehicle dynamics model developed in MATLAB.

Finally, Huang et al. developed an estimation scheme based on a limited-memory filter combined with an Extended Kalman Filter [16]. The limited-memory filter reduced the effects of old measurement data, which improved accuracy. A nonlinear vehicle dynamics model that used a simplified Dugoff tire model was implemented. Simulations were conducted using both CarSim and MATLAB with Simulink. In addition, experimental testing was also performed on a four-wheeled independent drive off-road vehicle.

## 2.5 Machine Learning-Based Approaches for Other Vehicle Parameters

Many other vehicle parameters and states have also been estimated using machine learning-based techniques in literature. A select number of these examples are reviewed in this section.

Firstly, slip angle is a commonly estimated value as it directly relates to lateral tire forces. Gräber et al. developed a side slip angle estimation scheme involving recurrent neural networks [11]. In particular, gated recurrent units (GRU) were used alongside a single track kinematic model. The estimation scheme was tested on experimental data gathered from a four-wheeled drive Porsche 911 Turbo. Another slip angle estimation scheme, based on intelligent tire technology and accurate up to 10 degrees, was developed by Xu et al. [53]. Data was collected from a tire testing platform along the contact patch region. Five different machine learning techniques were investigated including random forests, support vector machines, and gradient boosting machines. The most accurate technique was from an artificial neural network consisting of three hidden layers and using the resilient backpropagation (Rprop) algorithm. Another slip angle estimation scheme was developed by Novi et al. using both an artificial neural network and an Unscented Kalman Filter [28]. The estimation scheme only required readings from an Inertial Measurement

Unit (IMU). A pseudo-slip angle was first estimated from an artificial neural network trained offline. Next, the pseudo-slip angle was used in an Unscented Kalman Filter, which was based on a sensor model rather than a vehicle model to remain robust to different vehicle types. Finally, a correction step was implemented to generate the final slip angle estimate.

Torabi et al. trained a feedforward neural network containing 3 layers to estimate both road grade angles and vehicle mass [48]. Data was collected through the simulation of heavy-duty vehicles with different masses. The output layer of the neural network consisted of 2 neurons, where one corresponded to the estimated vehicle mass and the other corresponded to the estimated road grade angle. An artificial neural network was also used by Dong et al. to estimate road inclination angles [6]. The network consisted of one hidden layer and used 3 inputs consisting of velocity, steering wheel angle, and total roll angles. The results were used to improve the accuracy of a developed rollover model. Guzmán et al. trained an artificial neural network to estimate roll angles in real time [13]. Each of the inputs could be measured via low-cost sensors and consisted of the lateral acceleration, longitudinal acceleration, yaw rate, and roll rate. The artificial neural network consisted of 3 total layers including one hidden layer containing 15 neurons. The trained network was also implemented onto low-cost Internet of Things (IoT) devices, and was able to satisfy both performance constraints and achieve a required accuracy.

Statistical techniques for the online estimation of internal combustion engine torque values were investigated by Rakotomamonjy et al. [34]. The methods investigated included linear least squares, support vector machines, and nonlinear neural networks. The input data consisted of signals readily measurable from conventional vehicles including the accelerator pedal position, engine rotational speed, and current gear engaged. It was found that nonlinear neural networks performed with the greatest accuracy while linear models were unable to predict torque values correctly, especially at low gears. In addition, torque estimate accuracy deteriorated profoundly when there was an absence in accelerator pedal position signals. A major limitation to the study was that the developed model was not robust to changes in working conditions such as significant deviations from the ambient pressure or temperature.



# Chapter 3

## Artificial Neural Networks

Artificial neural networks approximate a target function based on training data and are widely used for both classification and regression problems. Larger high quality training data sets can lead to more accurate predictions. The focus of this chapter is on reviewing the theoretical background of feedforward neural networks and recurrent neural networks, which are two different network architectures. It concludes with an application to estimate friction forces on an inverted pendulum system.

### 3.1 Feedforward Neural Networks

**Definition 3 (Node)** *An individual node within an artificial neural network, also known as a neuron, is a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  that transforms a set of inputs  $x_1, x_2, \dots, x_n$  into a single output  $y$ .*

Artificial neural networks (ANN) are comprised of a finite number of layers that may contain a varying number of nodes per layer. The components associated with each individual node also have a corresponding set of weights for each input  $w_1, w_2, \dots, w_n$ , a bias term  $b$ , and an activation function  $\sigma$ . The final output  $y$  from an individual node is

$$y = \sigma\left(\sum_{i=1}^n w_i x_i + b\right). \quad (3.1)$$

A visual depiction of an individual node is displayed in Figure 3.1.

The  $\sum_{i=1}^n w_i x_i$  term is a weighted sum of the individual inputs to the node. Each  $w_i$  represents the importance of input  $x_i$  relative to the other inputs. The activation function,  $\sigma$ , normalizes the output of the individual node in the artificial neural network. The bias term  $b$ , shifts the activation function  $\sigma$ . A further discussion regarding the desirable properties of activation functions is presented later in this section.

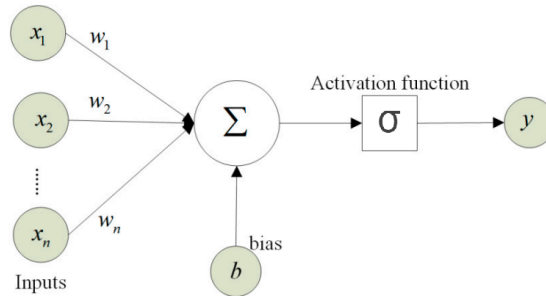


Figure 3.1: Single Artificial Neural Network Node (Modified from Fig. 1, [49])

An artificial neural network is typically composed of many individual nodes where the outputs of previous nodes are used as inputs to subsequent nodes. Many different neural network architectures are widely used [50].

**Definition 4 (Feedforward Neural Network)** *An artificial neural network is classified as a feedforward neural network if the connections between nodes do not form any cycles.*

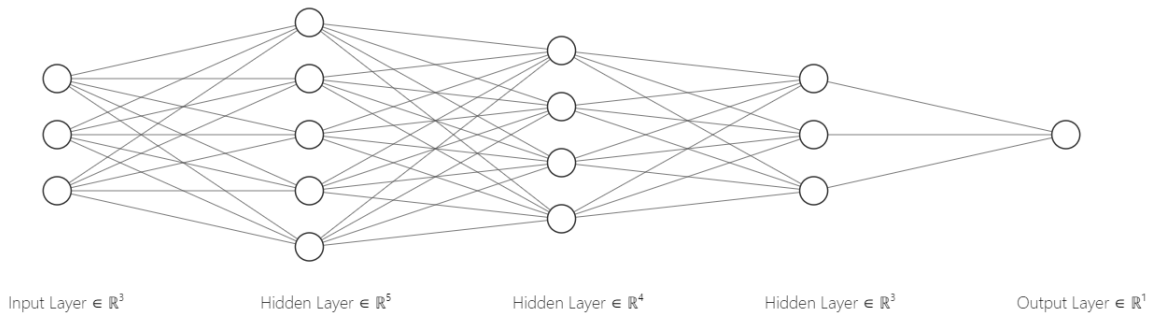


Figure 3.2: Example of a Feedforward Artificial Neural Network

Artificial neural networks begin with an input layer where training data is inserted. The training data is often preprocessed for normalization or to achieve a desired format. Next, outputs from the input layer nodes are sent through a series of hidden layers, which may contain a varying number of nodes per layer. Finally, the prediction from the neural network is interpreted from the output layer. In a classification problem, the output layer contains multiple nodes where the output from each node represents the probability of the original input belonging to a certain class. In a regression problem, outputs from nodes within the output layer correspond to predicted numerical values.

Feedforward artificial neural networks can have an arbitrary positive number of layers affecting their depth, and an arbitrary positive number of nodes per layer affecting their width. This selection has a large effect on the approximation of the target function and hence the final predictions made by the network on unseen test data. The overall artificial neural network  $g(X)$  can be expressed mathematically through function composition as

$$g(X) = \sigma^L \left( W^L \left( \sigma^{L-1} \left( W^{L-1} \dots \sigma^1 (W^1 X + b^1) \dots \right) + b^{L-1} \right) + b^L \right), \quad (3.2)$$

where  $X = (x_1, x_2, \dots, x_n)$  is the input feature vector,  $\sigma^l$  is the activation function at layer  $l$  with layer  $L$  as the last layer, and  $W^l$  is a matrix representing the weights between layer  $l$  and  $l - 1$ . Note that  $W^l = (w_{jk}^l)$  where  $(w_{jk}^l)$  is the weight between  $j^{\text{th}}$  node in layer  $l$  and the  $k^{\text{th}}$  node in layer  $l - 1$ .

**Definition 5 (Nonlinear Activation Function)** *A nonlinear activation function is a nonlinear function  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  that determines the output of an individual node.*

Nonlinearity is one of the most important properties in selecting an activation function. Nonlinear activation functions allow neural networks with at least one hidden layer to approximate an arbitrary continuous nonlinear function according to the Universal Approximation Theorem [5] [40]. Several different versions of this theorem exist under varying assumptions. Another desirable property for an activation function is to be continuously differentiable. This is desirable for training neural networks using gradient-based optimization methods, which will be covered in Section 3.2, where derivatives of the activation function are used throughout. Finally, inputs with a small magnitude generally yield relatively small outputs, and inputs with a large magnitude generally yield relatively large outputs.

A selection of common activation functions are shown in Table 3.1. Note that the desirable properties previously stated are not necessarily satisfied by each of the activation functions listed. For example, the Rectified Linear Unit (ReLU) activation function is not continuously differentiable, and the Identity activation function is linear.

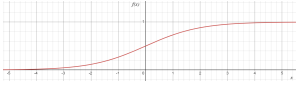

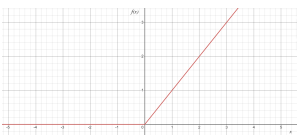

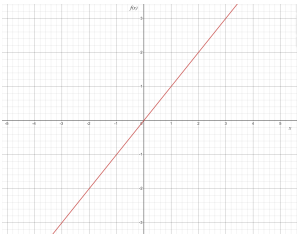
Activation Function	Expression	Plot	Range	Continuity
Sigmoid ( $\sigma$ )	$f(x) = \frac{1}{1+e^{-x}}$		$(0, 1)$	$C^\infty$
Hyperbolic Tangent ( $\tanh$ )	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$		$(-1, 1)$	$C^\infty$
Rectified Linear Unit (ReLU)	$f(x) = \max\{0, x\}$		$[0, \infty)$	$C^0$
Softplus	$f(x) = \ln(1 + e^x)$		$(0, \infty)$	$C^\infty$
Identity	$f(x) = x$		$(-\infty, \infty)$	$C^\infty$

Table 3.1: Common Activation Functions

## 3.2 Training Artificial Neural Networks

Training a neural network involves adjusting the weights  $w_i$  and biases  $b_i$  to minimize a cost function. The cost function under a supervised learning paradigm is typically defined as an error between the predicted label values from the artificial neural network and the true label values. A common cost function is defined using the  $L_2$  norm as

$$C(W) = \frac{1}{2} \sum_{i=1}^N \left( \|f(x_i, W, b) - y_i\|_2 \right)^2. \quad (3.3)$$

Note that  $f(x_i, W, b)$  are the predictions generated by the neural network due to the weights  $W$  and the biases  $b$ , and  $y_i$  are the true label values. Thus, training an artificial neural network can be posed as an optimization problem. One of the most widely-used methods to adjust the weights biases to minimize the cost function is to use gradient descent and backpropagation [10].

**Definition 6 (Gradient Descent)** *A class of optimization algorithms that minimizes a cost function with regard to a training data set. It involves calculating the gradient, which is a vector of partial derivatives of the cost function with respect to the input variables.*

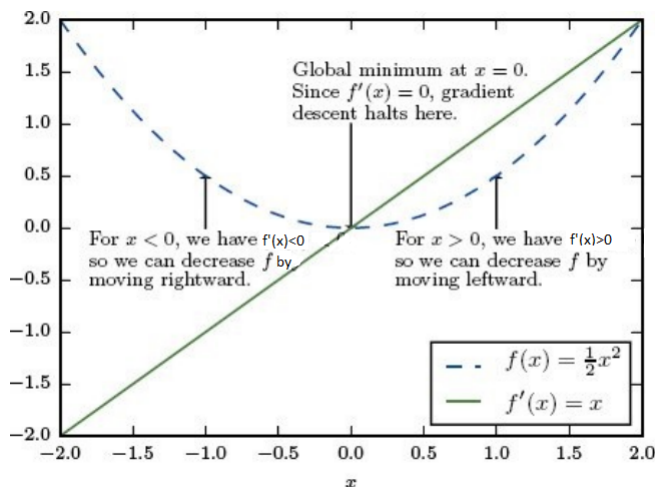


Figure 3.3: Gradient Descent Example in 1D (Fig 4.1, [10])

Individual weights and biases for nodes within the hidden layer of an artificial neural network only containing one hidden layer can be adjusted through

$$w_{ij} = w_{ij} - \alpha \frac{\delta C}{\delta w_{ij}}, \quad (3.4)$$

and

$$b_i = b_i - \alpha \frac{\delta C}{\delta b_i}. \quad (3.5)$$

Individual weights are adjusted in proportion to their contribution to the overall error as determined by the  $\frac{\delta C}{\delta w_{ij}}$  term. The learning rate,  $\alpha$ , is a hyperparameter that affects the magnitude of weight adjustments.

The weights and biases for nodes of an artificial neural network containing multiple hidden layers are adjusted through backpropagation, which is an algorithm utilizing the chain rule from calculus to compute derivatives in a computational graph such as a neural network. Denote  $w_{jk}^l$  to be the weight between the  $j^{\text{th}}$  node in layer  $l$  and the  $k^{\text{th}}$  node in layer  $l - 1$ . In addition,  $b_j^l$  denotes the bias of the  $j^{\text{th}}$  node in the  $l^{\text{th}}$  layer. Finally, let  $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$  denote the weighted input to the activation function for the node  $j$  within layer  $l$ .

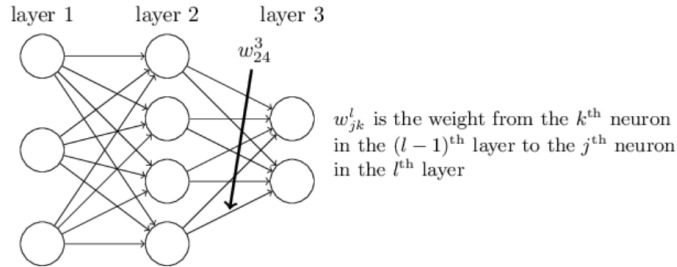


Figure 3.4: Example of Notation Used to Denote Weights (Fig. 1, [27])

An error expression for a node in the output layer is

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L). \quad (3.6)$$

The Hadamard product, denoted by  $\odot$ , is a symbol representing elementwise multiplication between two vectors. Using this notation, equation (3.6) can also be expressed as

$$\delta^L = \nabla_a C \odot \sigma'(z^L), \quad (3.7)$$

where  $\nabla_a C$  is a vector whose components are the partial derivatives of  $\frac{\partial C}{\partial a_j^l}$ . Next, the error  $\delta^l$  can be expressed in terms of the error in the next layer  $\delta^{l+1}$  as

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \quad (3.8)$$

where  $(w^{l+1})^T$  is the transpose of the weight matrix  $w^{l+1}$  for the  $(l+1)^{th}$  layer. Equations (3.7) and (3.8) can be used together to calculate the error for any network layer. The rate of change of the error function with respect to a bias term is simply calculated as

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \quad (3.9)$$

Finally, the rate of change of the error function with respect to any weight of the network is expressed as

$$\frac{\partial C}{\partial w_{ij}^l} = a_k^{l-1} \delta_j^l. \quad (3.10)$$

Equations (3.7), (3.8), (3.9), and (3.10) depict the steps of the backpropagation algorithm for a single training example. The gradients for many training samples are calculated, and the average among the individual gradients from each training sample is used to make adjustments to the weights and biases. Variations to the above method also exist for computational efficiency [10] [29].

**Definition 7 (Epoch)** *A complete pass of an entire training data set through a model.*

In practice, the training data set is often randomly split into several batches. The weights and biases throughout the neural network are adjusted after iterating through a single batch, rather than after iterating through the entire training data set. This can improve the convergence rate of training a neural network [10].

**Definition 8 (Batch Size)** *The total number of training samples within a single batch.*

**Definition 9 (Mini-Batch Gradient Descent)** *Mini-batch gradient descent is a form of gradient descent where parameters are adjusted after computing the gradient of the error with respect to a subset of the training set.*

The full procedure is described in Algorithm 1, where  $m$  is the batch size.

---

**Algorithm 1** Backpropagation and Gradient Descent

---

1. Input a set of training samples

For each training sample  $x$ :

- (a) **Forward Pass:** Determine the activation  $a^{x,1}$  for the input layer.

In addition, for  $l = 2, 3, \dots, L$  compute

$$z^{x,l} = w^l a^{x,l-1} + b^l \text{ and } a^{x,l} = \sigma(z^{x,l})$$

- (b) **Error:** Compute the vector

$$\delta^{x,L} = \nabla_a C \odot \sigma'(z^{x,L})$$

- (c) **Backpropagation:** For  $l = L - 1, L - 2, \dots, 2$  compute

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

2. **Gradient Descent:** For each  $l = L - 1, L - 2, \dots, 2$  update the weights as

$$w^l \rightarrow w^l - \alpha \left(\frac{1}{m}\right) \sum_x \delta^{x,l} (a^{x,l-1})^T$$

$$b^l \rightarrow b^l - \alpha \left(\frac{1}{m}\right) \sum_x \delta^{x,l}$$


---

Splitting the data into multiple batches is computationally efficient, and also reduces the memory required as only a subset of the training set is processed by the network at a time [46]. Overall, properly trained neural networks have great potential in approximating functions based on collected training data. However, developing a reliable prediction model based on an artificial neural network is an iterative process. Many different heuristics exist to determine appropriate neural network architectures, input features, and hyperparameters.



### 3.3 Recurrent Neural Networks

Many other neural network architectures exist beyond the standard feedforward architecture that may be better suited towards making predictions on certain data [50]. For example, recurrent neural networks have an architecture that is well-suited for capturing temporal relationships present within training data. This feature makes it far more applicable towards making predictions involving dynamical systems, which includes friction estimation.

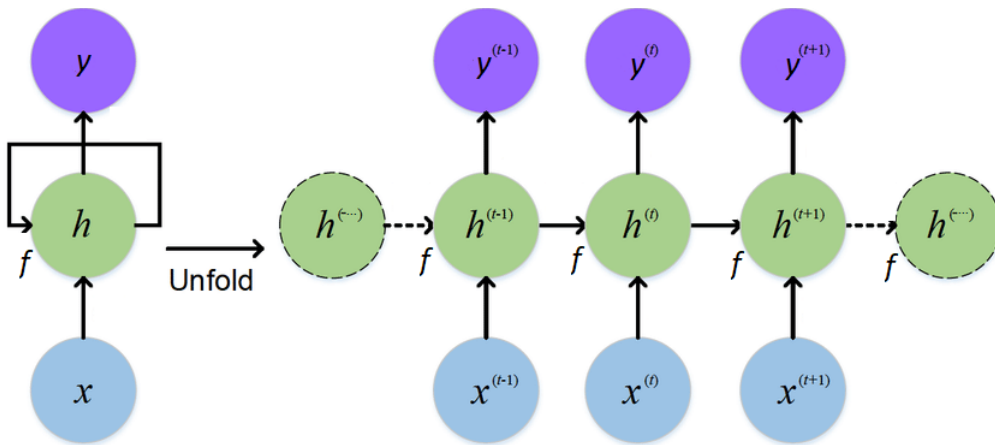


Figure 3.5: Unfolded Recurrent Neural Network (Modified from Fig. 1, [7])

The key feature of recurrent neural networks compared to traditional feedforward neural networks is that information from a previous hidden state, which is the output from the network at a previous time step, is used to make a prediction for the current time step. An unfolded diagram of a recurrent neural network is displayed in Figure 3.5. It is noteworthy that the standard recurrent neural network architecture allows for data of varying length to be input into the network sequentially at each time step. A hidden state vector is initialized for  $t = 0$ . The relationship between hidden layers and inputs for  $t \in \{\mathbb{Z}^+ \mid 1 \leq t \leq T\}$  is

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}), \quad (3.11)$$

where  $T$  is the length of the entire input sequence.

One example for the function  $f$  would be  $f = \sigma(W^h h^{(t-1)} + W^x x^{(t)})$ , where  $\sigma$  is an activation function,  $W^h$  are the weights associated with the hidden state, and  $W^x$  are the weights associated with the input.

The specific architecture of a recurrent neural network varies between the types of input and output data, as displayed in Figure 3.6. The many to one architecture is the most relevant to estimating a friction coefficient. The input is sequential measurement data gathered from various sensors, and the output is a vector containing a single entry representing the friction coefficient. Many to many architectures are relevant in tasks such as language translation, one to many architectures are relevant in tasks such as image captioning, and one to one architectures are relevant in tasks such as image classification.

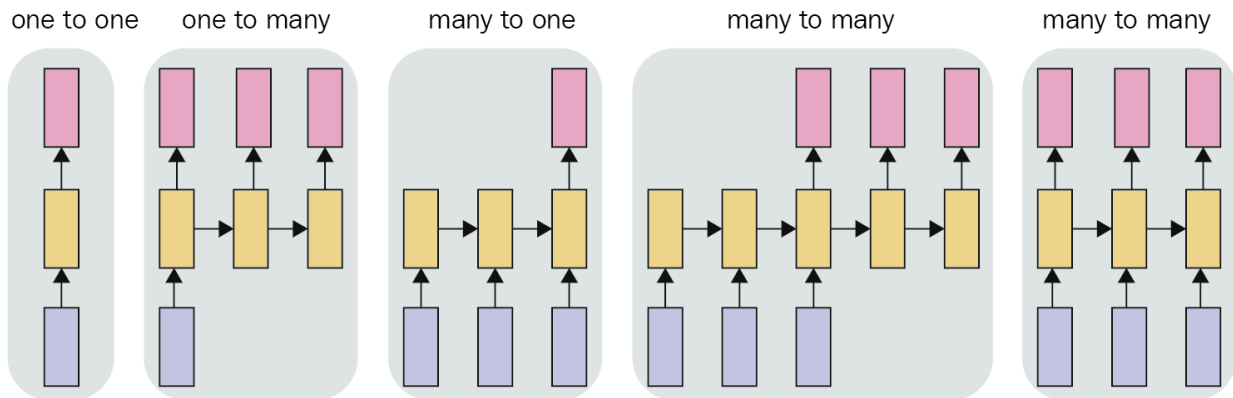


Figure 3.6: Various Recurrent Neural Network Architectures (Fig. 11.1, [1])

In each of the depicted recurrent neural network architectures, the problem of vanishing or exploding gradients may arise. Recurrent neural networks are trained using backpropagation through time, which is a very similar process to backpropagation for feedforward neural networks. The process is shown visually from the unfolded network in Figure 3.7. Errors are calculated and accumulated for each individual time step. The weights,  $W^h$  and  $W^x$ , are updated as a result of the error across all time steps.

In equation (3.11), the function  $f$  is the same between all time steps as the weights are only updated once a total error has been determined after accounting for all time steps. Hence, the weights  $W^h$  are also the same at each time step. The gradient of the weight between two arbitrary nodes of a network is dependent on gradients with respect to other weights in the network closer to the output layer. This can be induced from equation (3.10), which determines the partial derivative of the cost function with respect to an arbitrary weight term.

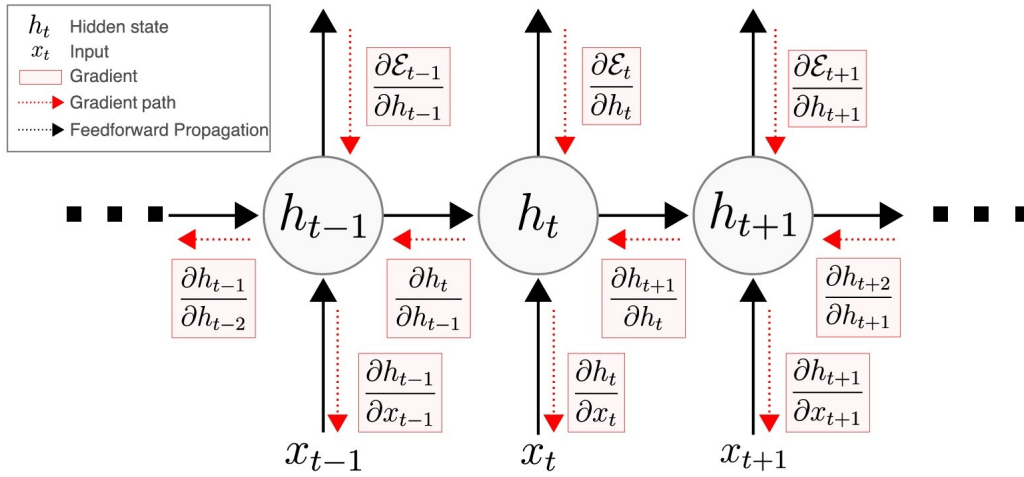


Figure 3.7: Depiction of Backpropagation Through Time (Fig. 1, [23])

If the weights  $W^h$  are less than one, then repeatedly multiplying by  $W^h$  for terms early in the sequence will lead to a very small value for the gradient, possibly very close to zero. This is known as the vanishing gradient problem where early weights in a neural network are adjusted very minimally from their initial values due to small gradients. Hence, nodes that are relatively far from the output layer have a minimal effect on the final predictions generated.

Conversely, if the weights  $W^h$  are greater than one, repeatedly multiplying by  $W^h$  for terms far from the end of the sequence will lead to very large gradient values. This is known as the exploding gradient problem where weight adjustments are very drastic between nodes that are far from the output layer. This may lead to weights never achieving their optimal values as each adjustment is too large in magnitude.

Vanishing and exploding gradient problems are still present in deep feedforward neural networks. However, the problem is less drastic in comparison. This is due to differing weights between each layer of a feedforward neural network. The weights may be greater than one between some layers, and less than one between other layers. Hence, effects from the magnitude of the weights often cancel out such that the gradient neither vanishes nor explodes. However, this is not the case with recurrent neural networks as the same weights  $W^h$  are used throughout each time step. Therefore, the vanishing and exploding gradient problems can be far more significant in recurrent neural networks.

### 3.4 Long Short-Term Memory

A notable method to mitigate the vanishing gradient problem is to use a recurrent neural network architecture where certain information is retained or lost at each time step. Long short-term memory (LSTM) is one such architecture and was first proposed by Hochreiter et al [15]. In this architecture, a cell state is included in addition to the hidden state in between each time step.

An LSTM unit consists of three gates that output values between 0 or 1. The purpose of the gates is to control the information that flows through a LSTM unit, as displayed in Figure 3.8.

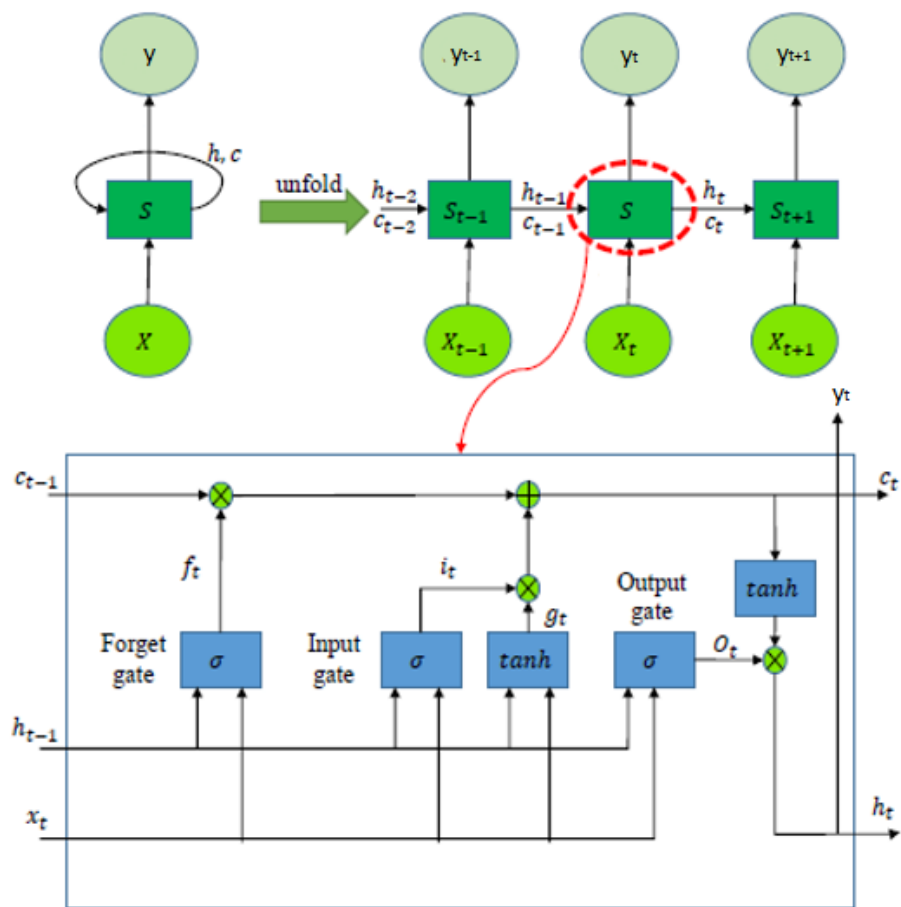


Figure 3.8: LSTM Architecture (Modified from Fig. 8.5, [14])

Expressions for the three gates are given below.

- **Forget Gate:**

$$f_t = \sigma(W^{(xf)}x_t + W^{(hf)}h_{t-1} + b_f) \quad (3.12)$$

- **Input Gate:**

$$i_t = \sigma(W^{(xi)}x_t + W^{(hi)}h_{t-1} + b_i) \quad (3.13)$$

- **Output Gate:**

$$o_t = \sigma(W^{(xo)}x_t + W^{(ho)}h_{t-1} + b_o) \quad (3.14)$$

Note that  $\sigma$  specifically refers to the sigmoid activation function in the three gate equations. The sigmoid activation function is used since it is differentiable and restricts the output range between 0 and 1. Subsequent states are determined using the proceeding equations.

- **Process Input:**

$$g_t = \tanh(W^{(xg)}x_t + W^{(hg)}h_{t-1} + b_g) \quad (3.15)$$

- **Cell Update:**

$$c_t = f_t \times c_{t-1} + i_t \times g_t \quad (3.16)$$

- **Output:**

$$y_t = h_t = O_t \times \tanh(C_t) \quad (3.17)$$

The forget gate controls whether the cell state  $c_{t-1}$  is to be preserved. An output value close to 1 from the forget gate implies that most of the previous cell state should be used in subsequent computations. Conversely, a value close to 0 implies that most of the previous cell state should not be used. The input gate determines how the cell state  $c_{t-1}$  is to be updated for the next time step. The cell state  $c_t$  is determined as a weighted combination of  $c_{t-1}$  and the process input  $g_t$ . The output gate affects the value of the next hidden state, which also corresponds to the output of the LSTM unit at the current time step. Information from the current cell state  $c_t$  is combined with the output from the gate to determine the hidden state for the next time step.

The tanh activation function is used on the process input  $g_t$  since it has a zero-centered range from -1 to 1. This distributes the gradients well over a long operation, which allows cell state information to flow longer without vanishing or exploding. Overall, training a LSTM unit involves adjusting the parameters  $W^{(xi)}$ ,  $b_i$ ,  $W^{(xf)}$ ,  $b_f$ ,  $W^{(xo)}$ ,  $b_o$ ,  $W^{(xg)}$ , and  $b_g$ .

### 3.5 Friction Estimation on an Inverted Pendulum Using LSTM

Consider an inverted pendulum system as shown in Figure 3.9.

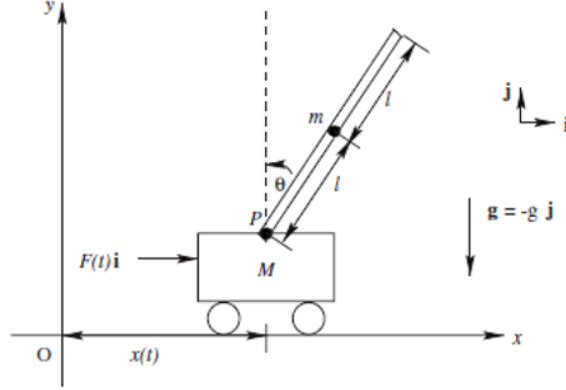


Figure 3.9: Inverted Pendulum System (Fig. 1, [3])

The equations of motion for the cart and pendulum can be derived using Hamilton's principle. They are

$$(M + m)\ddot{x}(t) - ml \sin(\theta(t))\dot{\theta}(t)^2 + ml \cos(\theta(t))\ddot{\theta}(t) = F_{applied}(t) + F_{fric}(t) \quad (3.18)$$

$$ml \cos(\theta(t))\ddot{x}(t) - mgl \sin(\theta(t)) + \frac{4}{3}ml^2\ddot{\theta}(t) = 0 \quad (3.19)$$

where

$$F_{fric} = \begin{cases} F_{static} & \dot{x} = 0 \\ -(\mu_c + (\mu_s - \mu_c))e^{(\frac{\dot{x}}{v_s})^\gamma} F_N \text{sgn}(\dot{x}) - \epsilon \dot{x} & \dot{x} \neq 0 \end{cases} \quad (3.20)$$

$$F_{applied}(t) = \alpha V(t) - \beta \dot{x}(t) \quad (3.21)$$

$$F_N = (m + M)g \quad (3.22)$$

$$F_{static} = \begin{cases} -F_{applied} & |F_{applied}| < \mu_s F_N \\ -\mu_s F_N \text{sgn}(F_{applied}) & |F_{applied}| \geq \mu_s F_N. \end{cases} \quad (3.23)$$

The applied force is due to a motor in the cart that is controlled by the supplied voltage  $V$ . The parameter  $\alpha$  represents the voltage to force conversion factor, and the parameter  $\beta$  represents the electrical resistance within the motor. In addition, an exponential friction model is selected in modelling the system to maintain a smooth transition between a stationary and moving cart. The three variables which can be measured from an experimental setup of the system are listed in Table 3.2, and system model parameter values are in Table 3.3.

Measurement	Units	Description
$\theta$	rad	Angle with Respect to the Vertical Axis
$x$	m	Cart Position
$V$	V	Applied Voltage

Table 3.2: Measurements Available from Inverted Pendulum System

Parameter	Symbol	Value
Mass of the Cart	M	0.815 kg
Mass of the Pendulum	m	0.210 kg
Distance From the Pivot to Center of Mass of the Pendulum	l	0.305 m
Gravitational Constant	g	$9.8 \frac{m}{s^2}$
Voltage to Force Conversion Factor	$\alpha$	$1.719 \frac{N}{V}$
Electrical Resistance to Force Conversion Factor	$\beta$	$7.682 N \frac{s}{m}$
Coefficient of Viscous Friction	$\epsilon$	$3 N \frac{s}{m}$
Stribeck Velocity	$\nu_s$	$0.105 \frac{m}{s}$
Form Factor	$\gamma$	2
Coefficient of Static Friction	$\mu_s$	0.08610
Coefficient of Coulomb Friction	$\mu_c$	0.04287

Table 3.3: Inverted Pendulum System Parameter Values (Tab. 1 and 2, [3])

Training data was collected from a simulation of the system created using MATLAB Simulink. A linear quadratic regulator, with details found in [3], was also applied with the aim to stabilize the pendulum in an upright position. The ode23tb solver was used to generate time series measurement data for  $\theta$ ,  $x$ , and  $V$  over 0.5 seconds starting from various initial conditions. The initial conditions included the starting cart position, pendulum

angle, cart velocity, and pendulum velocity:

$$\begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix}.$$

Initial conditions were selected to contain a diverse range of pendulum angles, cart velocities, and pendulum velocities to capture system dynamics at varying conditions. A sample rate of 0.001 seconds was used for measurement data collection for a total of 500 data points per data set. The training data consisted of 24 total data sets generated in this manner with differing initial conditions.

$$\mu_c = 0.04287 \quad t \geq 0 \quad (3.24)$$

$$\mu_s = 0.08610 \quad t \geq 0 \quad (3.25)$$

$$\mu_c = \begin{cases} 0.009 & 0 \leq t < 0.125 \\ 0.04287 & 0.125 \leq t < 0.3755 \\ 0.02 & t \geq 0.3755 \end{cases} \quad (3.26)$$

$$\mu_s = \begin{cases} 0.011 & 0 \leq t < 0.125 \\ 0.08610 & 0.125 \leq t < 0.3755 \\ 0.05 & t \geq 0.3755 \end{cases} \quad (3.27)$$

The first 12 of the data sets were generated using constant values for the friction coefficients,  $\mu_c$  and  $\mu_s$ , as displayed in equations (3.24) and (3.25), respectively. The other 12 data sets were generated using friction coefficients varying over time as displayed in equations (3.26) and (3.27). Data from the measurements  $\theta$ ,  $x$ , and  $V$  were normalized as

$$\theta_{i_{Normalized}} = \frac{\theta_i - \theta_{max}}{\theta_{max} - \theta_{min}} \quad (3.28)$$

$$x_{i_{Normalized}} = \frac{x_i - x_{max}}{x_{max} - x_{min}} \quad (3.29)$$

$$V_{i_{Normalized}} = \frac{V_i - V_{max}}{V_{max} - V_{min}}. \quad (3.30)$$



Noise was added to each of the individual measurements  $y_i$ :

$$y_i = \begin{bmatrix} \theta_i \\ x_i \\ V_i \end{bmatrix} + \epsilon_i \text{ where } \epsilon_i \sim N(0, 0.02).$$

Examples of measurement data gathered from an individual data set are shown in Figures 3.10, 3.11, and 3.12.

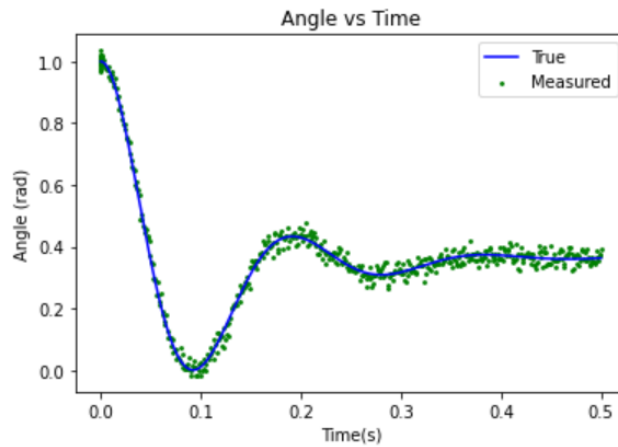


Figure 3.10: Sample Angle Measurement Data

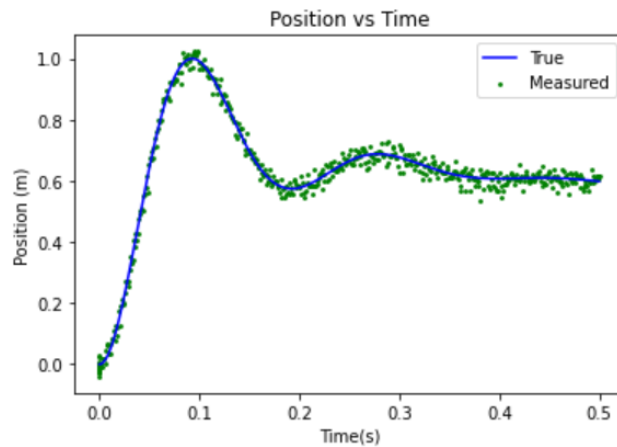


Figure 3.11: Sample Position Measurement Data

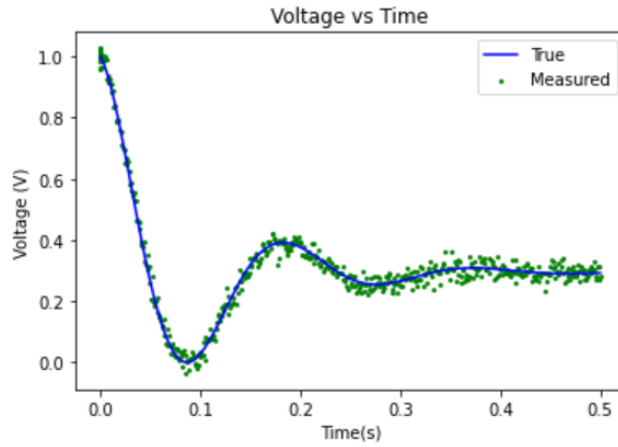


Figure 3.12: Sample Voltage Measurement Data

The training data was processed into intervals containing 10 consecutive time series measurements with the corresponding label as the friction force at the subsequent time step. That is, each training data point was assigned the label  $F_{fric}$  at time step  $i+1$  for  $i = 10, 11, 12, \dots, 499$  as displayed in Figure 3.13.

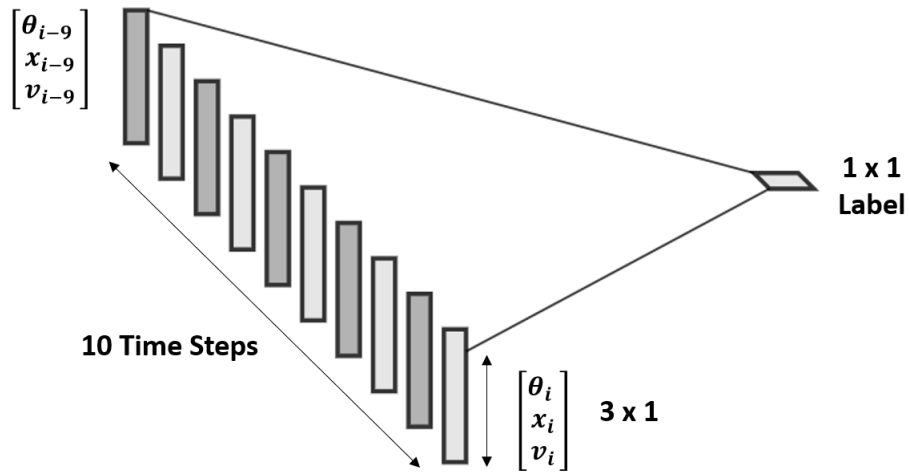


Figure 3.13: Visualization of Processed Data for Friction Force Estimation

A recurrent neural network containing stacked LSTM units was implemented to estimate the friction using the three measurements as input data. The network was implemented using Keras, which is an open-source software library that provides an interface for artificial neural networks. Figure 3.14 displays the network architecture and Table 3.4 contains further architecture details.



Figure 3.14: LSTM Network Architecture Layout for Friction Estimation on Inverted Pendulum System

Layer #	Layer Type	Dimension of Output	Output Activation Function
1	LSTM	10	ReLU
2	LSTM	10	ReLU
3	Dense	1	Linear

Table 3.4: LSTM Network Architecture for Friction Estimation of Inverted Pendulum System

The dense layer type refers to a typical feedforward layer and is simply used as the output layer in this setup. The ReLU activation function was applied to the output of each LSTM layer. Training was performed over 100 epochs using a batch size of 15. Hyperparameters for the LSTM architecture, including the number of layers and output dimensions, were selected through trial and error. The test set consisted of a single time series data set for  $\theta$ ,  $x$ ,  $V$  generated in the same manner as the data within the training data set. It was generated with an initial condition that did not appear within the training set of

$$\begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 \\ 0.05 \\ 0 \\ 0 \end{bmatrix}.$$

Each query contained a history of the data over the past 9 time steps to incorporate the temporal dependence between data points within a close time interval, similar to the processing of the training data.

A total of 490 friction force estimates were made within the time interval between 0.01 seconds to 0.5 seconds, after every 0.001 seconds. Figure 3.15 displays the friction estimate that was generated by the LSTM network over the test data set and Table 3.5 summarizes the resulting estimation. The average prediction time listed is the average time elapsed for the artificial neural network to yield an estimate for a given input across all 490 estimates.

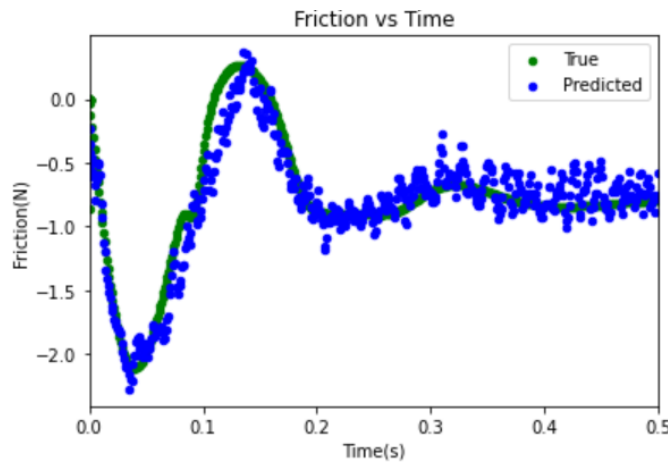


Figure 3.15: Friction Estimation using LSTM on Inverted Pendulum System

Feature	Value
Error (RMSE) Over Entire Time Interval	5.623
Training Time	781 seconds
Required Storage	61 KB [Model Size (Architecture + Weights)]
Average Prediction Time (Across all 490 Queries)	0.000728 seconds

Table 3.5: LSTM Network Performance Metrics on Inverted Pendulum System

A further discussion regarding the estimation results from the LSTM network is presented at the end of Section 4.4.

# Chapter 4

## Instance-Based Learning

Instance-based learning methods do not attempt to fit a function globally to the entire training data set contrary to neural networks. Local approximations of the target function around each new query are created instead. This can be advantageous when the target function is a very complex function that could be represented by a collection of simpler local approximations [26].

One of the key features of instance-based learning methods is that no training process is required prior to making predictions. The training phase is an important and timely process when creating a global function approximation. For example, the weights and biases of a neural network must be carefully adjusted based on the training data prior to generating new predictions on unseen test data. Conversely, no training phase exists for instance-based learning methods as the training data is simply collected and stored. However, computation costs to make predictions from new queries can be relatively high compared to making predictions from a trained neural network. This difference will be discussed in Chapter 5.

### 4.1 K-Nearest Neighbors

K-Nearest Neighbors is one of the most widely used instance-based learning techniques. Key steps are outlined in Algorithm 2. Note that K, which represents the number of training points to consider near a new query, is a hyperparameter that must be adjusted for each training set. Values of K which are too high lead to underfitting and values of K which are too low tend towards overfitting [47].

---

**Algorithm 2** K-Nearest Neighbors

---

1. Calculate distances between the new query and all previously collected training points
  2. Determine the K training points with the smallest distance from the query
  3. For Numerical Labels: The prediction is equal to the average label value between the K closest training points  
For Categorical Labels: The prediction is equal to the most frequent label among the K closest training points
- 

**Example:** Consider the data set in Table 4.3 consisting of 2 features (X,Y), and 3 categorical labels (Red, Blue, Green).

X	Y	Label
100	100	Red
200	200	Red
120	200	Red
50	120	Red
320	155	Red
81	319	Red
278	100	Blue
450	94	Blue
291	200	Blue
374	190	Blue
256	187	Blue
313	291	Blue
432	430	Green
170	385	Green
250	380	Green
281	282	Green
455	321	Green
390	403	Green

Table 4.1: Training Data for K-NN Example

Applying the K-NN algorithm with K=5 around a new query of (X,Y) = (240,250) results in a prediction of Blue. This classification is displayed visually in Figure 4.1.

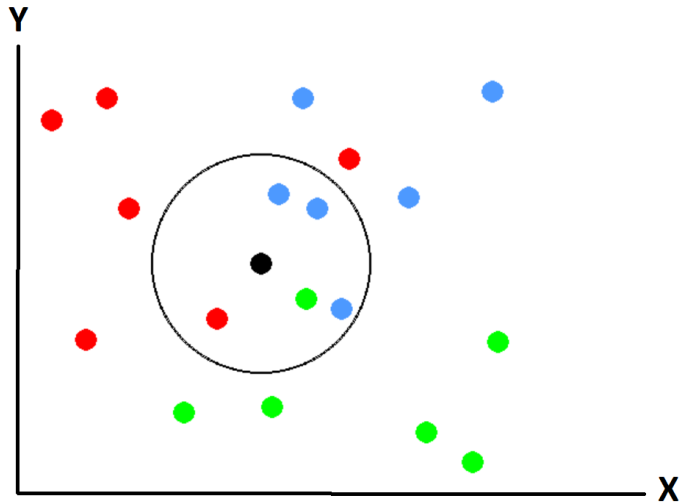


Figure 4.1: Visual Depiction of K-NN Algorithm

The K-Nearest Neighbors method is relatively simple and exhibits each of the properties of instance-based learning methods previously mentioned. It involves no training phase as all the required computation only occurs once a prediction is to be made from a new query.

## 4.2 Nadaraya-Watson Kernel Regression

Nadaraya-Watson Kernel Regression is another instance-based learning approach that exhibits many of the same properties as the K-Nearest Neighbors algorithm. It can be used when dealing with numerical training data. It also provides a more sophisticated weighting scheme compared to simply taking an average from K-Nearest Neighbors. Several definitions are first presented.

**Definition 10 (Statistical Model)** *A statistical model is a pair  $(S, \mathcal{P})$  where  $S$  is the set of all possible observations, also known as the sample space, and  $\mathcal{P}$  is a set of probability distributions on  $S$ .*

**Definition 11 (Parametric)** *A parametric model is a statistical model  $(S, \mathcal{P})$  with  $\mathcal{P} = \{\mathcal{P}_\theta : \theta \in \Theta\}$  where the set of parameters  $\Theta$  is finite dimensional.*

Parametric models are created under the assumption that the sample data comes from a population that can be adequately modeled by a probability distribution containing a fixed set of parameters. Hence, the number of parameters in the model do not change as the size of the data set changes. A simple example of a parametric model is a linear regression model as follows:

$$Y = mX + b. \tag{4.1}$$

There are only 2 fixed parameters in this model; the slope  $m$ , and the y-intercept  $b$ . Linear regression is a process that determines values for these 2 parameters from collected training data. This is performed by tuning the parameters to minimize a loss function, which would often be the squared residual error:

$$\sum_i e_i^2 = \sum_i (Y_i - \hat{Y}_i)^2, \tag{4.2}$$

where  $Y_i$  is the true value and  $\hat{Y}_i$  is the predicted value.

An example of a linear regression model is displayed in Figure 4.2. Clearly, the resulting predictive model would be a straight line regardless of the training data. The linear model is parametric as it always retains the form of a straight line regardless of the training data.

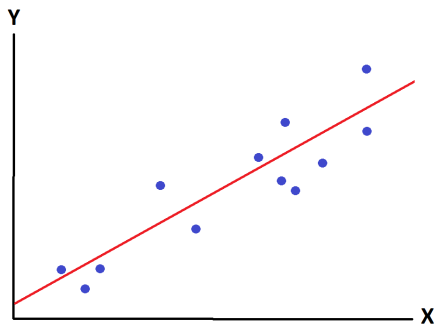


Figure 4.2: Visual Depiction of Linear Regression

Common parametric models widely used in statistics include the Gaussian, Poisson, and Binomial models [8]. It is also clear that a prediction model may be very inaccurate if a poor choice is made for the distribution beforehand. An example would be using a linear model to make predictions on a process where the true underlying distribution is Gaussian. Predictions made using a linear model would certainly have a large error when predicting values far from the mean.



**Definition 12 (Non-Parametric)** *A non-parametric model is a statistical model  $(S, \mathcal{P})$  with  $\mathcal{P} = \{\mathcal{P}_\theta : \theta \in \Theta\}$  where the set of parameters  $\Theta$  is infinite dimensional.*

Non-parametric models are created without any assumptions on the form or parameters. They can also be interpreted as distribution-free models. Non-parametric regression is the process of determining a function that best fits collected training data with no prior assumptions on the relationship between the predictors and dependent variable.

A simple example of a non-parametric model is a histogram. It provides an estimate for an underlying probability distribution without assuming any particular form of the distribution. Non-parametric models tend to require larger training data set sizes since both the model structure and model parameters must be determined from the data [8].

**Definition 13 (Kernel)** *In non-parametric regression, a kernel  $K(x)$  is any symmetric and non-negative function satisfying the following properties for  $x \in \mathbb{R}$ :*

$$\int_{-\infty}^{\infty} K(x) dx = 1 \tag{4.3}$$

$$\int_{-\infty}^{\infty} xK(x) dx = 0 \tag{4.4}$$

$$0 < \int_{-\infty}^{\infty} x^2K(x) dx < \infty \tag{4.5}$$

The relevance of each training data point when making new predictions is quantified using a kernel, which can act as a weighting function in some non-parametric regression models. Some commonly used kernels are shown in Table 4.2 where  $u = \frac{x-X_i}{\sigma}$  [42].

**Definition 14 (Bandwidth)** *The bandwidth of a kernel function,  $\sigma$ , is a tunable parameter that affects the overall size and shape of the kernel function.*

The value of the bandwidth determines the size of the local region around a new query where previously collected training data is allocated relatively larger weights. Note that this is a value that must be manually tuned for each training data set. A bandwidth that is too large leads to underfitting by incorporating data points that are too far from the query. A bandwidth that is too small leads to overfitting by only including a very small number of data points in the prediction.

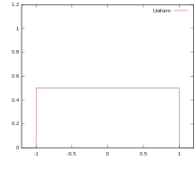
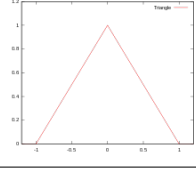
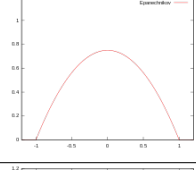
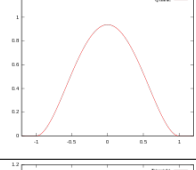
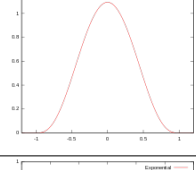
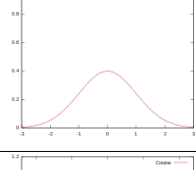
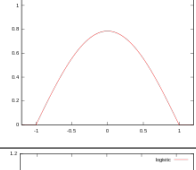
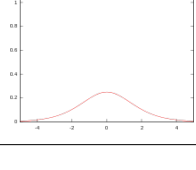
Kernel Function	Expression	Plot
Uniform	$K(u) = \frac{1}{2}\mathbf{I}( u  \leq 1)$	
Triangular	$K(u) = 1 -  u \mathbf{I}( u  \leq 1)$	
Epanechnikov (Parabolic)	$K(u) = \frac{3}{4}(1 - u^2)\mathbf{I}( u  \leq 1)$	
Quartic (Biweight)	$K(u) = \frac{15}{16}(1 - u^2)^2\mathbf{I}( u  \leq 1)$	
Triweight	$K(u) = \frac{35}{32}(1 - u^2)^3\mathbf{I}( u  \leq 1)$	
Gaussian	$K(u) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}u^2}$	
Cosine	$K(u) = \frac{\pi}{4} \cos(\frac{\pi}{2}u)\mathbf{I}( u  \leq 1)$	
Logistic	$K(u) = \frac{1}{e^u + 2 + e^{-u}}\mathbf{I}( u  \leq 1)$	

Table 4.2: Common Kernel Functions

**Definition 15 (Univariate Kernel Density Estimate)** Let  $X = (x_1, x_2, \dots, x_n)$  be a set of independent and identically distributed samples drawn from a univariate distribution at some point  $x$ . The univariate kernel density estimate is defined as

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^n K_h(x - X_i) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - X_i}{h}\right),$$

where  $K$  is a given kernel function, and  $h$  is the bandwidth.

Kernel density estimation is a non-parametric method to estimate the probability density function of a random variable from collected sample data. An extension exists for the multivariate case.

**Definition 16 (Multivariate Kernel Density Estimate)** Let  $X = (x_1, x_2, \dots, x_n)$  where each  $x_i \in \mathbb{R}^p$  for  $i = 1, 2, \dots, n$ , be a set of independent and identically distributed samples drawn from a multivariate distribution containing  $p$  parameters at some point  $x$ . The multivariate kernel density estimate is defined as

$$\hat{f}(x) = \frac{1}{n|H|^{\frac{1}{2}}} \sum_{i=1}^n K\left(H^{-\frac{1}{2}}(x - X_i)\right),$$

where  $K$  is a multivariate kernel function with  $p$  parameters, and  $H$  is the  $p \times p$  symmetric and positive definite bandwidth matrix.

A common simplification is that the bandwidth matrix is assumed to be diagonal. That is,  $H = \text{diag}(h_1^2, h_2^2, \dots, h_p^2)$ . The resulting multivariate kernel density estimate becomes:

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^n K_{h_1}(x_1 - X_{i,1}) \times \dots \times K_{h_p}(x_p - X_{i,p}),$$

where  $K_{h_i}$  is a kernel function with corresponding bandwidth  $h_i$  for  $i=1, 2, \dots, n$ .

The objective of Nadaraya-Watson Kernel Regression is to determine a regression function  $m(x) = \mathbb{E}(Y|X = x)$  given a set of training data  $\{X_i, Y_i\}_{i=1}^n$ , where  $X_i \in \mathbb{R}^n$  and  $Y_i \in \mathbb{R}$ . Using basic properties of continuous random variables, the regression function can be written in terms of the joint density distribution  $f(x, y)$  and the marginal density distribution  $f_X(x)$  as

$$m(x) = \mathbb{E}(Y|X = x) = \int y f_{Y|X=x}(y) dy = \frac{\int y f(x, y) dy}{f_X(x)}. \quad (4.6)$$

The kernel density estimate of  $f_X$  is

$$\hat{f}_X(x) = \frac{1}{n} \sum_{j=1}^n K_{\sigma_1}(x - X_j), \quad (4.7)$$

and the multivariate kernel density estimate of  $f(x, y)$  is

$$\hat{f}(x, y) = \frac{1}{n} \sum_{i=1}^n K_{\sigma_1}(x - X_i) K_{\sigma_2}(y - Y_i). \quad (4.8)$$

A non-parametric estimate for  $m(x)$ ,  $\hat{m}(x)$ , can be made by replacing the density functions in equation 4.6 with their kernel density estimates. Substituting equations (4.7) and (4.8) into equation (4.6) yields

$$\begin{aligned} \hat{m}(x) &= \frac{\int y \hat{f}(x, y) dy}{\hat{f}_X(x)} = \frac{\int y \frac{1}{n} \sum_{i=1}^n K_{\sigma_1}(x - X_i) K_{\sigma_2}(y - Y_i) dy}{\frac{1}{n} \sum_{j=1}^n K_{\sigma_1}(x - X_j)} \\ &= \frac{\frac{1}{n} \sum_{i=1}^n K_{\sigma_1}(x - X_i) (\int y K_{\sigma_2}(y - Y_i) dy)}{\frac{1}{n} \sum_{j=1}^n K_{\sigma_1}(x - X_j)} \\ &= \frac{\frac{1}{n} \sum_{i=1}^n K_{\sigma_1}(x - X_i) Y_i}{\frac{1}{n} \sum_{j=1}^n K_{\sigma_1}(x - X_j)} \\ &= \sum_{i=1}^n \frac{K_{\sigma_1}(x - X_i)}{\sum_{j=1}^n K_{\sigma_1}(x - X_j)} Y_i. \end{aligned}$$

Hence

$$\hat{m}(x) = \sum_{i=1}^n \frac{K_{\sigma}(x - X_i)}{\sum_{j=1}^n K_{\sigma}(x - X_j)} Y_i. \quad (4.9)$$

The resulting estimator is referred to as the Nadaraya-Watson kernel estimator. It can be expressed as

$$\hat{y}(x) = w(x)Y_i \tag{4.10}$$

where

$$w(x) = \frac{\sum_{i=1}^n K_\sigma(x - X_i)}{\sum_{j=1}^n K_\sigma(x - X_j)}. \tag{4.11}$$

Here,  $K_\sigma$  is a kernel with bandwidth  $\sigma$ . One of the most commonly used kernels is the Gaussian kernel [4]:

$$K_\sigma = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-X_i)^2}{2\sigma^2}}. \tag{4.12}$$

**Example:** Suppose predictions are to be made from a given a set of observations sampled from a distribution with the following function:

$$f(x) = x - x^2.$$

The training data consists of the noisy observations

$$y_i = f(x_i) + \epsilon_i,$$

where

$$\epsilon_i \sim N(0, 100).$$

The plot in Figure 4.3 displays 25 equally spaced predictions made along the domain  $\{x \in \mathbb{R} \mid -50 \leq x \leq 50\}$  using the Nadaraya-Watson Kernel Regression technique with the Gaussian kernel. A curve is interpolated in between the predictions.

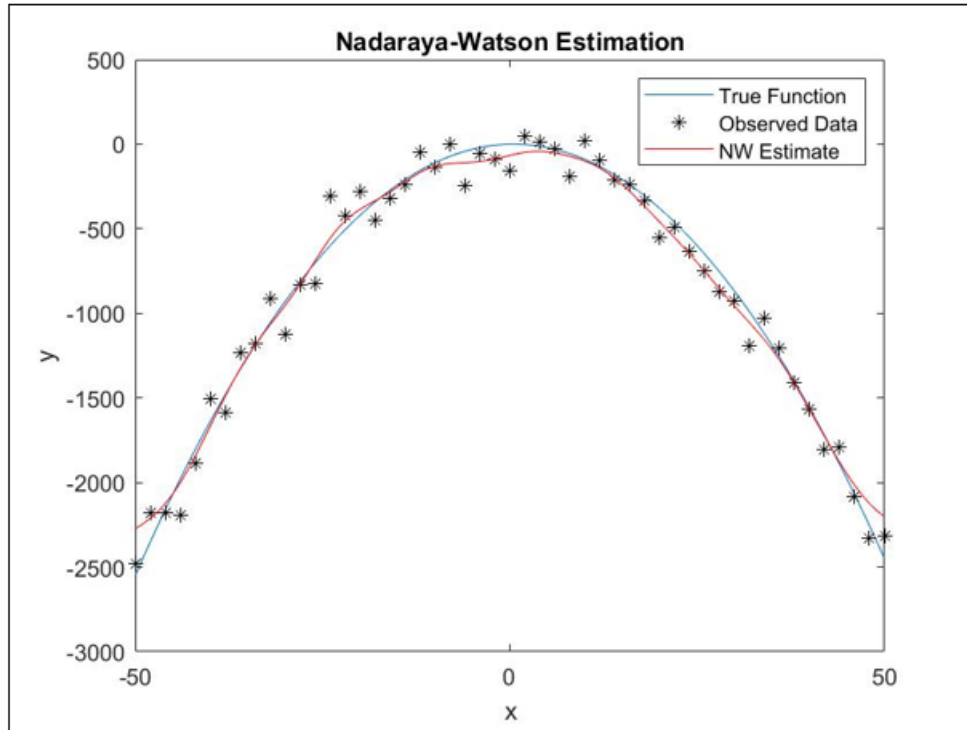


Figure 4.3: Nadaraya-Watson Estimation with Well-Tuned Bandwidth,  $\sigma = 1.9$

A bandwidth value of  $\sigma = 1.9$  was selected in Figure 4.3. Clearly, the bandwidth had been tuned to approximate the target function quite well. Resulting predictions that are clearly overfit and underfit from non-ideal bandwidth values are displayed in Figures 4.4 and 4.5, respectively. A small bandwidth value of  $\sigma = 0.5$  was selected for Figure 4.4 and a large bandwidth value of  $\sigma = 4$  was selected for Figure 4.5.

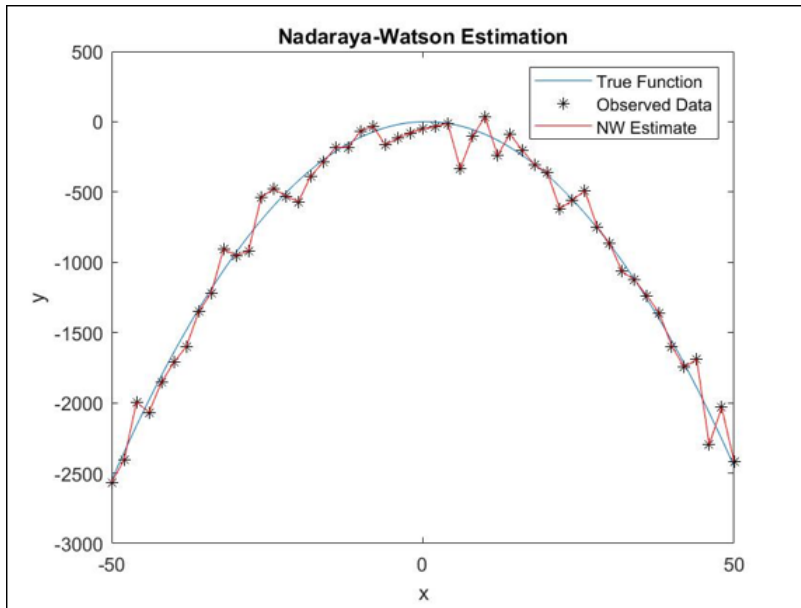


Figure 4.4: Nadaraya-Watson Estimation with Overfitting,  $\sigma = 0.5$

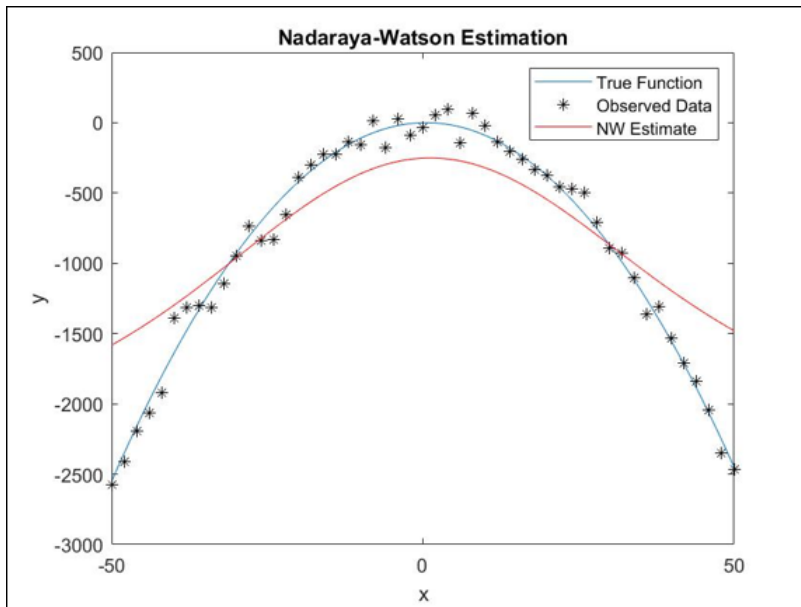


Figure 4.5: Nadaraya-Watson Estimation with Underfitting,  $\sigma = 4$

### 4.3 Implementation of Nadaraya-Watson Kernel Regression

Using the equations for Nadaraya-Watson Kernel Regression presented in the previous section can become computationally inefficient for very large training data sets. This would not be practical for online estimation on a system that continually requires new estimates within a short period time. Therefore, modifications are made to the previous equations to become more computationally efficient for implementation.

Suppose we have the training data set  $\{(X_i, Y_i)\}_{i=1}^n$ , where

$$X_i = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \text{ with corresponding label } Y_i, \text{ and a new query } Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}.$$

The Nadaraya-Watson Kernel Regression method shown in equation (4.10) is modified to only consider points within the training data set that are within a predefined threshold to the new query. The modified method is fully described in Algorithm 3.

---

**Algorithm 3** Nadaraya-Watson Kernel Regression Implementation

---

```

Set  $T \in \mathbb{R}^+$  as a threshold dictating a maximum allowable value for a distance metric
Initialize closest_points_list to be empty
Initialize  $\hat{y} = 0$ 
for each training data point  $X_i$  do
     $distance(X_i, Z) = \sum_{i=1}^n (x_i - z_i)^2$ 
    if  $distance(X_i, Z) < T$  then
        Append  $(X_i, Z)$  to closest_points_list
    end if
end for
for each  $(X_i, Z)$  in closest_points_list do
    Evaluate:  $\sum_{j=1}^n K_\sigma(x - X_j)$ 
     $\hat{y} = \hat{y} + \sum_{i=1}^n \frac{K_\sigma(x - X_i)}{\sum_{j=1}^n K_\sigma(x - X_j)} Y_i$ 
end for
Return  $\hat{y}$ 

```

---

The modified Nadaraya-Watson Kernel Regression method is now applied to the example of the inverted pendulum system from Section 3.5.



## 4.4 Friction Estimation on an Inverted Pendulum using Nadaraya-Watson Kernel Regression

The friction estimation problem for the inverted pendulum system from Chapter 3.5 was repeated using Nadaraya-Watson Kernel Regression instead of an artificial neural network. The same parameter values, training data sets, and test data set from Chapter 3.5 were used. The Nadaraya-Watson Kernel Regression method was implemented using the modified technique described in Section 4.3. A bandwidth of  $\sigma = 0.21$  and a closeness threshold of 0.63, which is equal to  $3\sigma$ , was used. The resulting predictions are displayed in Figure 4.6.

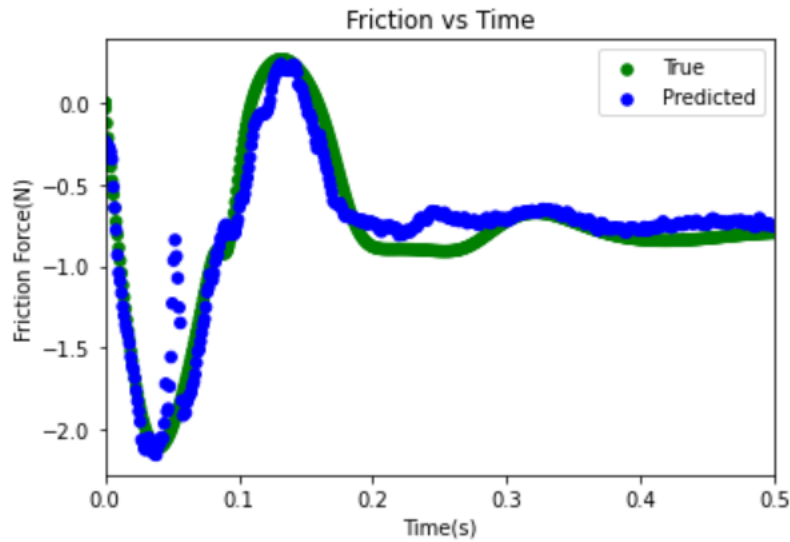


Figure 4.6: Friction Estimation using Nadaraya-Watson Kernel Regression on Inverted Pendulum System

Predicted values have a relatively larger error up to around 0.05 seconds compared to the predicted values in the time interval afterwards. Table 4.3 summarizes the resulting estimation.

<b>Feature</b>	<b>Value</b>
Error (RMSE) Over Entire Time Interval	3.794
Training Time	0 seconds
Required Storage	312 KB [Data Set Size]
Average Prediction Time (Across all 490 Queries)	0.330 seconds

Table 4.3: Nadaraya-Watson Kernel Regression Performance Metrics on Inverted Pendulum System

Both the LSTM network and Nadaraya-Watson Kernel Regression models were able to estimate friction forces for this system using the same training data sets to a reasonable degree of accuracy. However, there are distinct differences in the performance metrics. The Nadaraya-Watson Kernel Regression model achieved a lower RMSE of 3.794 compared to a RMSE of 5.623 for the LSTM model (See Table 3.5 on page 32). In addition, no training time was required for the Nadaraya-Watson Kernel Regression model.

On the other hand, the LSTM model notably had a far shorter average prediction time of 0.000728 seconds compared to an average prediction time of 0.330 seconds from the Nadaraya-Watson Kernel Regression model. This difference may be significant for implementation on a physical system that relies on an online estimate of friction forces. The storage required was also lower for the LSTM model, which required 61 KB to store the model architecture, including the weights and biases. In comparison, 312 KB were required to save the entire training data set, which is required for an implementation of the Nadaraya-Watson Kernel Regression model. These distinguishing characteristics between each method are investigated further in the road surface classification problem in the following chapter.

# Chapter 5

## Road Surface Classification

Both artificial neural networks and instance-based learning techniques described throughout chapters 3 and 4 are applied to classify road surface conditions in this chapter. First, the features and data sets used to train, validate, and test the performance of each model are described. Details regarding hyperparameter selection for each machine learning-based method are also discussed alongside a display of classification results. Finally, benefits and drawbacks between both methods are also compared with considerations towards achieving a practical implementation. A high-level overview of the classification methods are displayed in Figure 5.1.

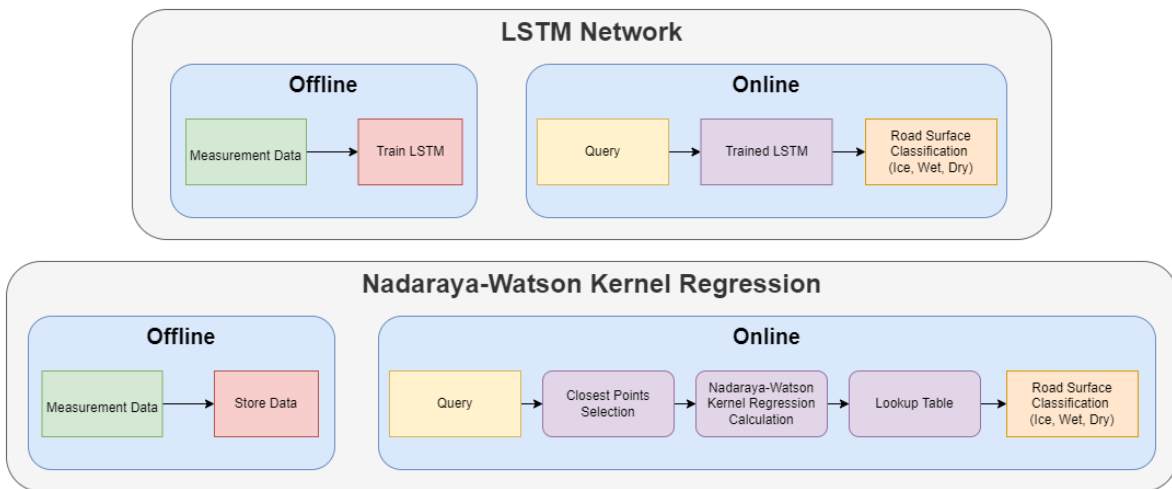


Figure 5.1: Overview of Road Classification Models

## 5.1 Feature Selection and Data Sets

One of the main constraints to developing an online classification scheme for deployment onto commercial vehicles is the limited number of onboard sensors. The number of sensors are limited on production vehicles due to their relatively expensive costs. Therefore, any model for road surface classification must have input variables restricted to features that can be commonly measured on commercial vehicles. The features shown in Table 5.1 were determined to be physically relevant towards classifying road surfaces and can all be directly measured or easily estimated using commonly available sensors on commercial vehicles. Each feature from the raw measurement data was first normalized as follows:

$$Feature_i = \frac{Feature_i - Feature_{max}}{Feature_{max} - Feature_{min}}. \quad (5.1)$$

<b>Feature</b>	<b>Units</b>	<b>Description</b>
AVy_L1	rpm	Front-Left Wheel Rotation Speed
AVy_L2	rpm	Front-Right Wheel Rotation Speed
AVy_R1	rpm	Rear-Left Wheel Rotation Speed
AVy_R2	rpm	Rear-Right Wheel Rotation Speed
AVz	deg/s	Yaw Rate
Ax_SM	g	Longitudinal Acceleration
Ay_SM	g	Lateral Acceleration
My_WC_L1	Nm	Front-Left Wheel Torque
My_WC_L2	Nm	Front-Right Wheel Torque
My_WC_R1	Nm	Rear-Left Wheel Torque
My_WC_R2	Nm	Rear-Right Wheel Torque
My_SW	Nm	Steering Wheel Torque
Steer_SW	deg	Steering Wheel Angle
Vx_SM	km/h	Longitudinal Speed
Vy_SM	km/h	Lateral Speed

Table 5.1: Relevant and Commonly Measurable/Estimated Features Among Commercial Vehicles

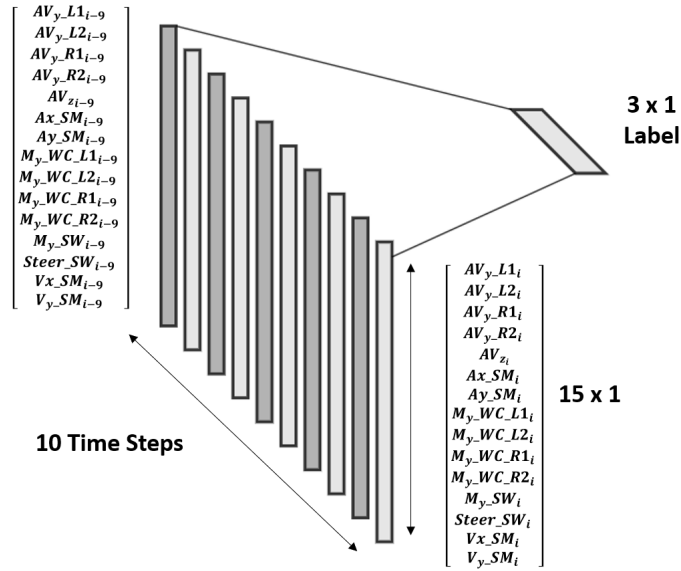


Figure 5.2: Visualization of Processed Data for Road Surface Classification

The data was processed into intervals containing 10 consecutive time series measurements containing each of the 15 features as displayed in Figure 5.2. Each training data point was assigned a one-hot encoded label according to Table 5.2. Queries generated during testing were also processed in a similar manner.

Surface	Label
Ice	$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$
Wet	$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$
Dry	$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Table 5.2: Labels Associated with Each Classification Surface

Data sets were gathered using CarSim, which is a multi-body vehicle dynamics simulator. Simulations were conducted on a C-Class Hatchback vehicle along various driving maneuvers at various target speeds. A sample rate of 0.1 seconds was selected to remain within reasonable latency requirements of communication networks present on physical vehicles, such as a CAN bus. A screenshot of a sample simulation for a Double Lane Change driving maneuver on a dry road is displayed in Figure 5.3.

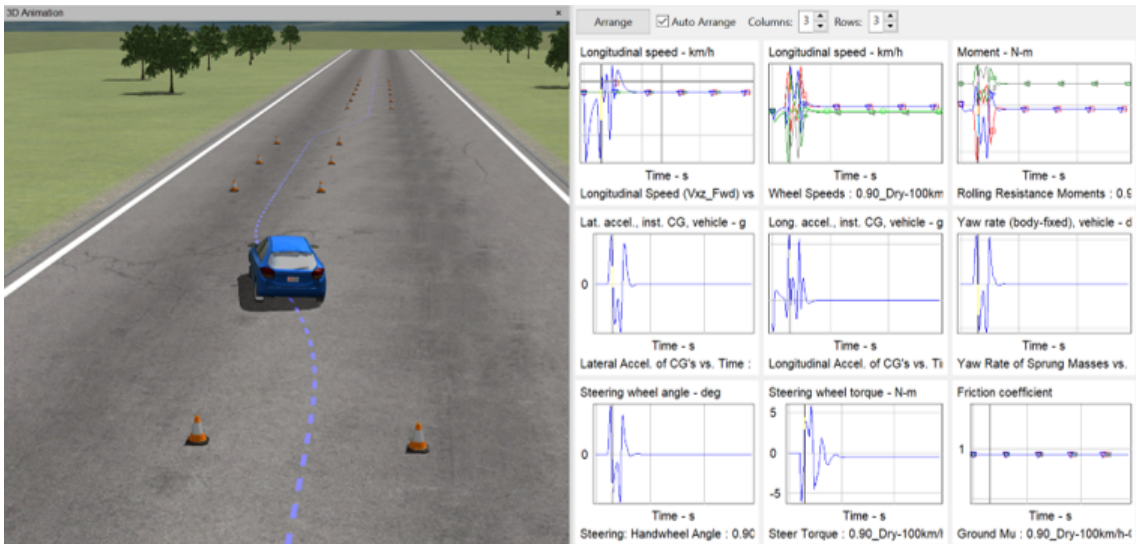


Figure 5.3: Sample CarSim Simulation

Tables 5.3 and 5.4 depict the simulations conducted to collect training and validation data used for both the LSTM network and Nadaraya-Watson Kernel Regression models. A larger number of simulations were created for wet and dry roads due to initial inaccuracies in classifying those surfaces using both models. In addition, the friction coefficients within each of the simulations were set to 0.1, 0.5, and 0.9 for the Ice, Wet, and Dry surfaces, respectively.

The initial simulations conducted were randomly split into training and validation sets with 90% of the simulations used for training data and 10% of the simulations used to assemble validation data. Separate simulations for the testing data set were also specifically created to assess the performance of each model, which are shown in Table 5.5. It is important to note that none of these simulations were present within the training or validation data sets. Each differed with a completely new driving maneuver and/or a new target speed from what had been observed within the training and validation data sets. Driving maneuvers are discussed in further detail throughout Appendix A.

Driving Maneuver	Target Speed (km/h)	Driving Maneuver	Target Speed (km/h)
Sine Sweep Steer	50	Sine Sweep Steer	50
	60		60
	70		70
	80		80
	90		90
	100		100
	110		110
	120		120
	130		130
	140		140
	150		150
	160		160
	170		170
	180		180
190	190		
200	200		
Double Lane Change	70	Double Lane Change	70
	80		80
	90		90
	100		100
	110		110
	120		120
	130		130
	140		140
	150		150
	160		160
	170		170
180	180		
190	190		
200	200		
S Turn with Clothoids	110	S Turn with Clothoids	110
	120		120
	130		130
	140		140
	150		150
	160		160
	170		170
	180		180
	190		190
200	200		
Rollover Steer	40	Rollover Steer	40
	50		50
	60		60
	70		70
	80		80
	90		90
	100		100
	110		110
	120		120
	130		130
	140		140
	150		150
	160		160
	170		170
	180		180
	190		190
	200		200

Table 5.3: Collected Data Involving Wet ( $\mu = 0.5$ ) [Left] and Dry ( $\mu = 0.9$ )[Right] Surfaces

Driving Maneuver	Target Speed ( <i>km/h</i> )
Sine Sweep Steer	40
	60
	80
	100
	120
Double Lane Change	40
	60
	80
	100
	120
S Turn with Clothoids	60
	80
	100
	120
Rollover Steer	20
	40
	60
	80
	100
	120

Table 5.4: Collected Data Involving Icy ( $\mu = 0.1$ ) Surfaces

Road Surface	Target Speed ( <i>km/h</i> )	Driving Maneuver
Ice ( $\mu = 0.1$ )	80	Increasing Steer
Ice ( $\mu = 0.1$ )	-	Accelerate and Brake
Ice ( $\mu = 0.1$ )	240	Straight Driving
Ice ( $\mu = 0.1$ )	80	Brake In Turn
Wet ( $\mu = 0.5$ )	85	Rollover Steer
Wet ( $\mu = 0.5$ )	220	S Turn with Clothoids
Wet ( $\mu = 0.5$ )	115	Side Drift
Wet ( $\mu = 0.5$ )	145	Sine Sweep Steer
Dry ( $\mu = 0.9$ )	65	Double Lane Change
Dry ( $\mu = 0.9$ )	-	Increasing Speed
Dry ( $\mu = 0.9$ )	135	Side Drift
Dry ( $\mu = 0.9$ )	95	S Turn with Clothoids

Table 5.5: Testing Data Set Simulations



## 5.2 LSTM Network Architecture and Estimation Results

Classification problems require a different type of activation function for the output layer such that a most probable label can be identified, rather than a numerical value. The softmax function is selected as the activation function of the last layer to perform multi-class classification since it converts the vector of numbers outputted from the previous layer into a vector of probabilities, where the elements of the final output sums to 1. The softmax function,  $f(\vec{s}) : \mathbb{R}^K \rightarrow [0, 1]^K$  where each element of the vector  $f(\vec{s})_i : \mathbb{R} \rightarrow [0, 1]$ , is

$$f(\vec{s})_i = \frac{e^{s_i}}{\sum_{j=1}^K e^{s_j}}. \quad (5.2)$$

Note that  $K$  is the number of classes, which was selected to be 3 for the road surface classification problem as the road surface would be labelled as either ice, wet, or dry.

The loss function that was minimized for this multi-class classification problem was the categorical cross-entropy loss:

$$CE = - \sum_{i=1}^K t_i \log(f(s)_i). \quad (5.3)$$

Note that  $f(s)$  is the softmax function from equation 5.2 and  $t_i$  is the true label from the overall target vector  $\vec{t}$ . This equation could also be simplified due to the one-hot encoding of the classification labels since there is only one element of the target vector that is non-zero. Only one term of the summation remains and the loss function simplifies to

$$CE = - \log \left( \frac{e^{s_p}}{\sum_{j=1}^K e^{s_j}} \right), \quad (5.4)$$

where  $p$  is the index of the only term in the target vector with a value of 1.

Values must be selected for several hyperparameters throughout the LSTM network model. Some examples include the number of hidden layers within the neural network, the dropout rate between layers, and the type of regularization applied along with its learning rate. KerasTuner, a framework within Keras, was used to optimize some of the hyperparameters with respect to validation accuracy [30]. In particular, a variation of the HyperBand algorithm presented in [22] was used. Table 5.6 displays investigated values for various hyperparameters.

Hyperparameter	Searchable Space
Number of LSTM Layers	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Output Dimension for each LSTM Layer	5, 10, 15...140, 145, 150
Dropout Factor (In Between each LSTM Layer)	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8
L1 Regularization Learning Rate	$1E^{-6}$ , $1E^{-5}$ , $1E^{-4}$ , $1E^{-3}$ , $1E^{-2}$ , $1E^{-1}$

Table 5.6: Searchable Space of Investigated Hyperparameters

The LSTM network architecture determined by the HyperBand algorithm is shown throughout Figure 5.4 and Table 5.7. L1 regularization was selected over L2 regularization as the validation accuracy was higher using L1 regularization after tuning the learning rate for both using HyperBand. In addition, ReLU was selected as the activation function for all LSTM layers since it lead to the greatest validation accuracy among the activation functions listed in Table 3.1.

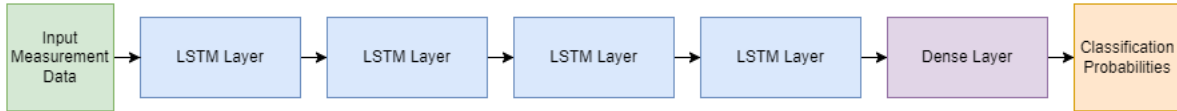


Figure 5.4: LSTM Network Architecture Layout

Layer #	Layer Type	Dimension of Output	Output Activation Function
1	LSTM	130	ReLU
2	LSTM	30	ReLU
3	LSTM	125	ReLU
4	LSTM	115	ReLU
5	Dense	3	softmax

Table 5.7: LSTM Network Architecture Details

A dropout factor of 0.5 in between layers 1 and 2, 2 and 3, as well as 3 and 4, was determined from the HyperBand algorithm. A learning rate of  $1E^{-4}$  was determined in a similar manner. In addition, the LSTM network was trained with 100 epochs and a batch size of 15. Training took approximately 1 hour and 40 minutes on a Intel Xeon CPU with two cores at 2.20 GHz with 13 GB of DDR4 RAM.

The first simulation was for an Increasing Steer driving maneuver with a target speed of 80 km/h on an icy road. The Increasing Steer driving maneuver was not present within the training or validation data sets. Measurement data used from the simulation is displayed in the Appendix as Figure B.1. A pie graph summarizing the predictions made by the neural network is displayed in Figure 5.5. Overall, there were 321 total predictions made. 316 of the predictions correctly classified the road surface as Ice, yielding an accuracy of 98.44% for the data within this simulation. It is also worth noting that the average prediction time was 0.000424 seconds.

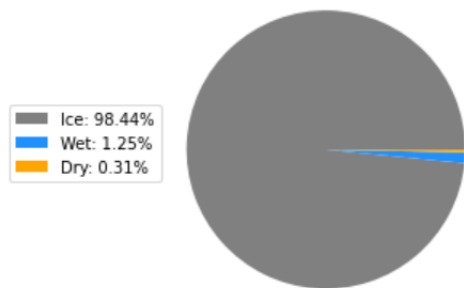


Figure 5.5: LSTM Network Estimation for 80 km/h Increasing Steer (Ice)

A simulation of a vehicle on an icy road performing an Acceleration and Brake driving maneuver was also part of the testing data set. Measurement data for this simulation is displayed in Figure B.2. Prediction results are displayed within the pie chart in Figure 5.6. 167 of the total 241 queries are correctly classified as Ice, which is an accuracy of 69.29%. On average, each test point took 0.000483 seconds to classify.

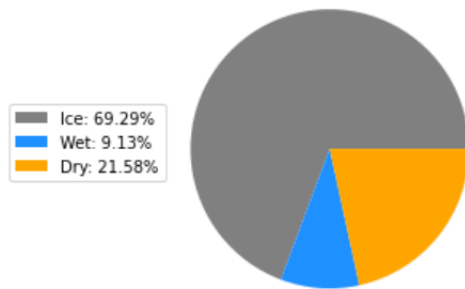


Figure 5.6: LSTM Network Estimation for Acceleration and Brake (Ice)

The overall prediction accuracy is worse on this driving maneuver compared to the Increasing Steer driving maneuver and most subsequent driving maneuvers within the test

set since it was drastically different from any of the driving maneuvers within the training or validation data sets.

Another simulation within the testing data set was for a Straight Driving maneuver on an icy road at a target speed of 240 km/h, which was faster than any simulation within the training data set. Figure B.3 displays the measurement data associated with this simulation. A summary of the predictions generated by the neural network is displayed within Figure 5.7. The road surface of all 419 queries were correctly identified as Ice for this simulation, and the average query prediction time was 0.000409 seconds.

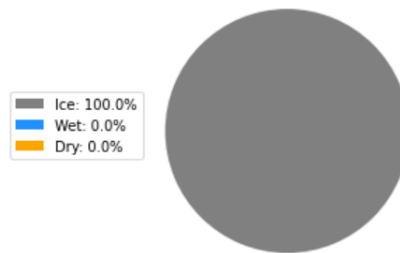


Figure 5.7: LSTM Network Estimation for 240 km/h Straight Driving (Ice)

The prediction accuracy was very high since many of the driving maneuvers within the training and validation test sets contained segments where the vehicle was driving straight. In addition, it was observed that measurements became more distinct between different road surfaces at higher speeds.

The next simulation within the testing data set was for a Brake In Turn driving maneuver from a speed of 80 km/h on an icy road. The measurement data acquired from this simulation is displayed in Figure B.4. A summary of the prediction results is displayed in Figure 5.8. Out of 191 queries, 179 of them were correctly classified as Ice, yielding a 93.72% accuracy. The average prediction time for each query was 0.000474 seconds.

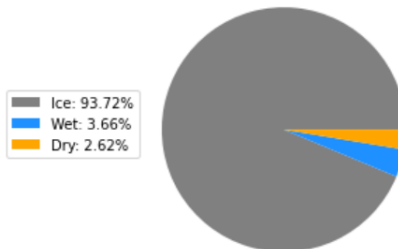


Figure 5.8: LSTM Network Estimation for 80 km/h Brake In Turn (Ice)

This driving maneuver was significantly different from any other driving maneuver present within the training or validation data sets since changing throttle and brake pedal commands were not issued significantly throughout those driving maneuvers. However, the prediction accuracy was still better than the prediction accuracy for the Accelerate and Brake maneuver since the Brake In Turn maneuver only uses the brake pedal without the throttle, while the Accelerate and Brake maneuver uses both. This additional deviation from the driving maneuvers within the training data led to a lower prediction accuracy on the Accelerate and Brake driving maneuver compared to the Brake In Turn driving maneuver.

The first simulation within the testing data set on a wet surface was for a Rollover Steer driving maneuver at a target speed of 95 km/h. This was an unseen target speed from any simulation within the training or validation data sets. Measurement data for this simulation is displayed within Figure B.5 and prediction results are displayed within Figure 5.9. The neural network yielded a 99.24% accuracy by classifying 130 out of 131 queries as Wet. The average prediction time among the queries was 0.000689 seconds.

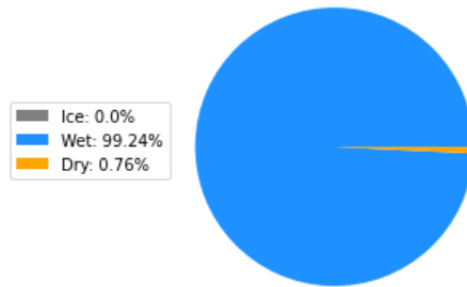


Figure 5.9: LSTM Network Estimation for 85 km/h Rollover Steer (Wet)

Overall, the prediction accuracy was relatively high since the training and validation data sets contained simulations of a vehicle performing the Rollover Steer driving maneuver at different speeds.

Next, a simulation of a vehicle with a target speed of 220 km/h on a wet road following an S Turn with Clothoids was investigated within the testing data set. Note that 220 km/h was beyond any target speed among the simulations present within the training or validation data sets. The measurements acquired from this simulation are displayed in Figure B.6 and neural network prediction results are displayed in Figure 5.10. A 100% accuracy rating is generated from this simulation as the 121 queries were all correctly classified as Wet. In addition, 0.000833 seconds was the average prediction time among these queries.

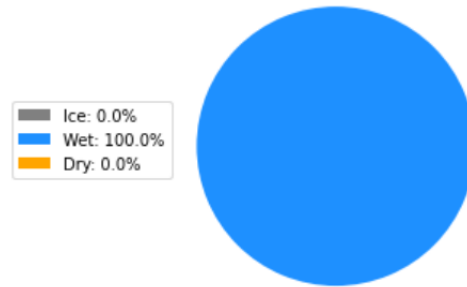


Figure 5.10: LSTM Network Estimation for 220 km/h S Turn with Clothoids (Wet)

The yielded prediction accuracy was due to the large speed of 220 km/h. Measurements were more distinct between different road surfaces at higher speeds, which made it easier to classify accurately.

A simulation for a vehicle on a wet road and a target speed of 115 km/h performing a Side Drift driving maneuver was also tested. The Side Drift driving maneuver is a new driving maneuver that was not present within the training or validation data sets. The measurement data associated with this simulation is displayed within Figure B.7. Figure 5.11 displays the results of the predictions made by the neural network. 40 out of the 61 total test points were correctly classified as Wet, which is an accuracy of 65.57%. The average prediction time for each test point for this simulation was 0.000969 seconds.

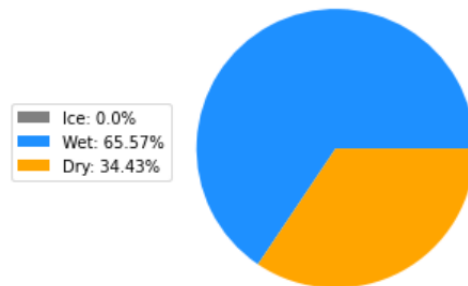


Figure 5.11: LSTM Network Estimation for 115 km/h Side Drift (Wet)

The Side Drift driving maneuver was relatively similar to the Double Lane Change driving maneuver, which was present within the training data set. However, the prediction accuracy on the Side Drift driving maneuver is worse compared to other driving maneuvers within the test set similar to those within the training and validation sets. The Side Drift

driving maneuver is still distinct from any driving maneuver at any speed from the training and validation data sets.

Another simulation was conducted for a Sine Sweep Steer driving maneuver on a wet road with a target speed of 145 km/h. Measurement data from this simulation is displayed in Figure B.8 and Figure 5.12 summarizes the predictions from the neural network. 232 out of the 291 queries were correctly classified as Wet, resulting in an accuracy of 79.73%. An average prediction time of 0.000456 seconds was also recorded.

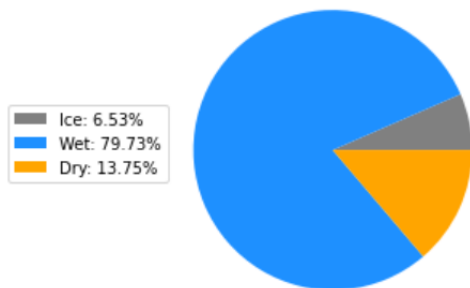


Figure 5.12: LSTM Network Estimation for 145 km/h Sine Sweep Steer (Wet)

The first simulation within the testing data set performed on a dry surface was for a vehicle on a dry road performing a Double Lane Change driving maneuver at a target speed of 65 km/h. This was an unseen target speed from any simulation within the training or validation data sets. Measurement data gathered from this simulation is shown in Figure B.9. Prediction results from the neural network are summarized in Figure 5.13. There were 95 queries correctly classified as Dry out of a total of 106 queries, which yielded an accuracy of 89.62%. The average test point prediction time was 0.000723 seconds.

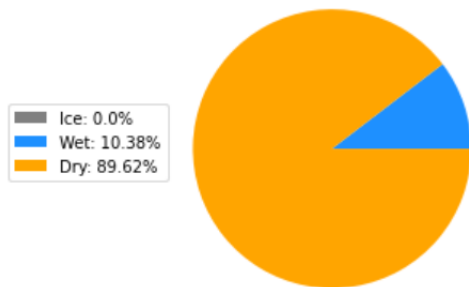


Figure 5.13: LSTM Network Estimation for 65 km/h Double Lane Change (Dry)

Next, the testing data set also contained a simulation for a vehicle on a dry road with the Increasing Speed driving maneuver, which was an unseen driving maneuver from the training and validation data sets. Measurement data from this test is shown in Figure B.10 and prediction results from the neural network are displayed in Figure 5.14. Out of 191 total queries, 144 were correctly classified as Dry, which yielded an accuracy of 75.39%. In addition, the average prediction time was 0.000548 seconds.

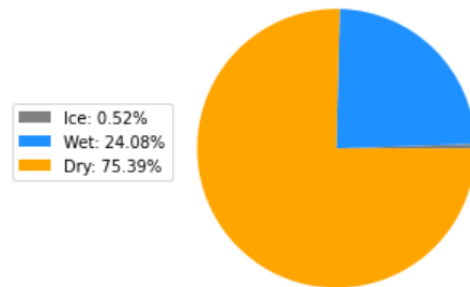


Figure 5.14: LSTM Network Estimation for Increasing Speed (Dry)

The Increasing Speed driving maneuver involves significant adjustments to the throttle command throughout the maneuver. However, throttle commands were generally held constant throughout many of the driving maneuvers within the training or validation data sets. This led to a relatively low prediction accuracy for this driving maneuver.

A simulation for a vehicle performing a Side Drift maneuver with a target speed of 135 km/h on a dry road was also conducted. The Side Drift driving maneuver was distinct from all driving maneuvers within the training and validation data sets. In addition, 135 km/h was an unseen target speed among simulations within the training and validation data sets. Measurement data from this simulation is displayed within Figure B.11. A summary of the prediction results generated by the neural network is displayed in Figure 5.15. An accuracy of 75.41% was generated from correctly classifying 46 out of 61 test points. The average prediction time among test points was 0.000976 seconds.



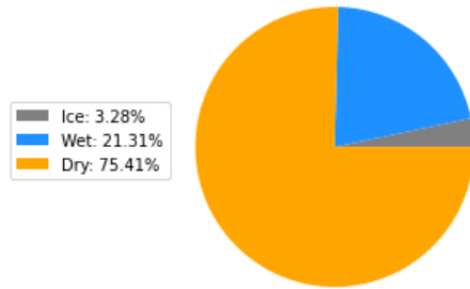


Figure 5.15: LSTM Network Estimation for 135 km/h Side Drift (Dry)

The prediction accuracy for this driving maneuver was worse compared to other driving maneuvers within the test set. This is similar to the predictions made for the simulation of a Side Drift maneuver performed at 115 km/h on a wet road. The Side Drift driving maneuver is distinct from any driving maneuver at any speed from the training and validation data sets.

Finally, a simulation was created for a vehicle following an S Turn with Clothoids with a target speed of 95 km/h on a dry road. Figure B.12 displays the measurement data acquired from this simulation and predictions generated from the neural network are summarized in Figure 5.16. The neural network was able to correctly classify 360 out of 369 queries as Dry, which resulted in a 97.56% accuracy. In addition, 0.000416 seconds was the average prediction time across queries.

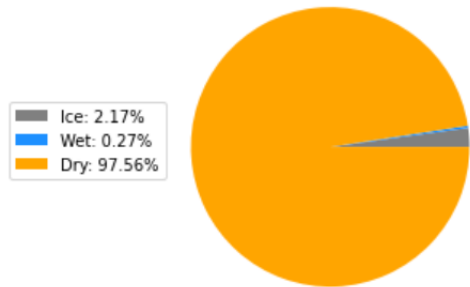


Figure 5.16: LSTM Network Estimation for 95 km/h S Turn with Clothoids (Dry)

## 5.3 Nadaraya-Watson Kernel Regression Hyperparameter Selection and Estimation Results

The implementation of the Nadaraya-Watson Kernel Regression algorithm was slightly modified from Section 4.4 for the road surface classification problem. The data set was much larger compared to the inverted pendulum problem. Therefore, a max heap data structure was used to first determine the  $K \in \mathbb{Z}^+$  closest points to the query, which reduces the overall prediction time.

Suppose we have the training data set  $\{(X_i, Y_i)\}_{i=1}^n$  where

$$X_i = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \text{ with corresponding label } Y_i, \text{ and a new query } Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}.$$

The  $K$  closest points to the query are determined as described in Algorithm 4.

---

### Algorithm 4 K Closest Points with Max Heap

---

Set  $K \in \{\mathbb{Z}^+ \mid 1 \leq K \leq n\}$  to be the number of points closest to the query to consider

Initialize empty heap

**for** each training data point  $X_i$  **do**

$$\text{distance}(X_i, Z) = \sum_{i=1}^n (x_i - z_i)^2$$

**if** (Size of the heap) < K **then**

    Push  $[\text{distance}(X_i, Z), X_i]$  to the heap

**else if**  $\text{distance}(X_i, Z) < \max(\text{distance}(X_i, Z))$  within heap **then**

    Pop element with the largest distance from the heap

    Push  $[\text{distance}(X_i, Z), X_i]$  to the heap

**end if**

**end for**

The  $K$  elements remaining within the heap are the  $K$  closest points to the query

---

The rest of the implementation was identical to the previous implementation from Section 4.4. Algorithm 4 can be interpreted as a specific implementation of the K-Nearest Neighbours algorithm depicted in Algorithm 2 using a max heap.

There were only two hyperparameters that had to be tuned for the Nadaraya-Watson Kernel Regression model. The first was the bandwidth  $\sigma$ , which affected the overall size and

shape of the kernel function. The second was the number of points  $K$  within the training data set to consider around the local neighbourhood of each new query. A bandwidth value of  $\sigma = 0.505$  and a total of  $K = 10$  points were selected from trial and error to yield the greatest accuracy for the validation data set.

The Nadaraya-Watson Kernel Regression model outputs a numerical value representing the estimated friction coefficient. The lookup table displayed in Table 5.8 was used to convert the numerical estimate into a specific road surface classification. In addition, the simulations used within the testing data set for the LSTM network model were the exact same as those used to test the Nadaraya-Watson Kernel Regression method for direct comparison.

Friction Coefficient	Surface Classification
[0.0-0.4)	Ice
[0.4-0.7)	Wet
[0.7-1.0]	Dry

Table 5.8: Lookup Table for Surface Classification (Tab. 6, [43])

The first test was on an icy road with an Increasing Steer driving maneuver at a target speed of 80 km/h. Prediction results are summarized in Figure 5.17. 286 out of 321 queries were correctly classified as Ice, which yielded an accuracy of 89.10%. The average prediction time using Nadaraya-Watson Kernel Regression was 0.293 seconds.

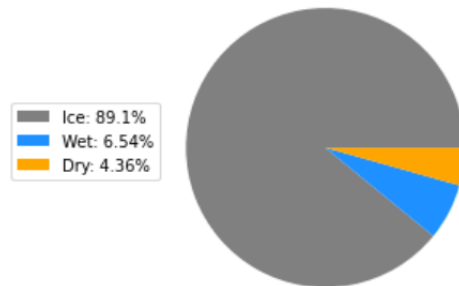


Figure 5.17: Nadaraya-Watson Estimation for 80 km/h Increasing Steer (Ice)

The prediction accuracy on this driving maneuver using the Nadaraya-Watson Kernel Regression model is lower compared to the LSTM network. This will be discussed throughout a comparison between the two methods in Section 5.4.

A simulation of a vehicle executing the Acceleration and Brake driving maneuver on an icy road was also present within the testing data set. A summary of prediction results is displayed in Figure 5.18. An accuracy of only 39.00% was generated from correctly identifying 94 out of 241 total queries as Ice. This was a result of the Acceleration and Brake maneuver being drastically different from any other driving maneuver within the training or validation data sets since changing throttle and brake pedal commands were not issued significantly throughout those driving maneuvers. This will be discussed further in Section 5.4. The average prediction time was 0.262 seconds.

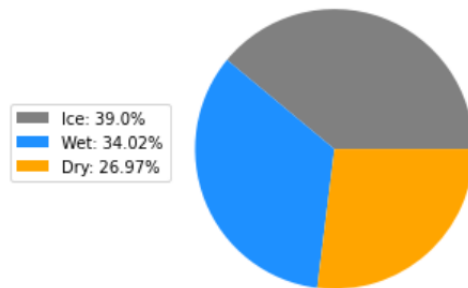


Figure 5.18: Nadaraya-Watson Estimation for Acceleration and Brake (Ice)

In addition, a simulation of a vehicle performing a Straight Driving maneuver at a target speed of 240 km/h on an icy road was investigated. A pie chart summarizing the predictions is displayed in Figure 5.19. 491 total queries were generated, in which 468 were correctly classified as Ice. This yielded an accuracy of 95.32%, which was lower than the LSTM network accuracy. The average prediction time was 0.275 seconds for this simulation.

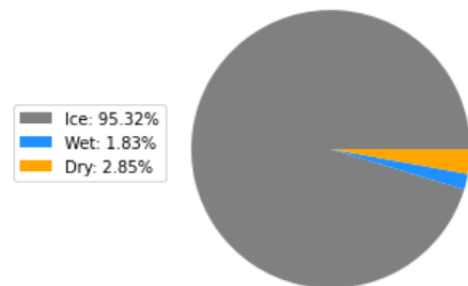


Figure 5.19: Nadaraya-Watson Estimation for 240 km/h Straight Driving (Ice)

The next test was for a Brake In Turn driving maneuver performed at 80 km/h on an icy road. A summary of the prediction results is displayed in Figure 5.20. Only 76 out of 191 queries were correctly classified as Ice with an average prediction time of 0.282 seconds. The 39.79% accuracy is a consequence of the Brake In Turn driving maneuver being very different from any driving maneuver present within the training or validation data sets. A further discussion is presented in Section 5.4.

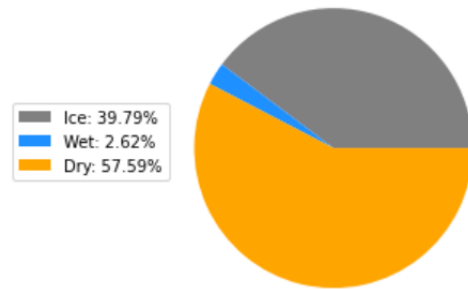


Figure 5.20: Nadaraya-Watson Estimation for 80 km/h Brake In Turn (Ice)

The first simulation within the testing data set for a wet road was with a Rollover Steer driving maneuver at a target speed of 85 km/h. Nadaraya-Watson Kernel Regression results are summarized in Figure 5.21. All 131 queries were correctly classified as Wet yielding a 100% accuracy for this simulation, which is higher than the prediction accuracy from the LSTM network. The average prediction time for data regarding this simulation was 0.265 seconds.

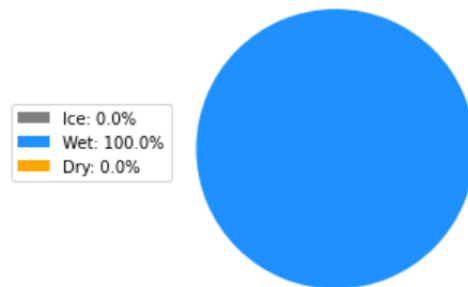


Figure 5.21: Nadaraya-Watson Estimation for 85 km/h Rollover Steer (Wet)

The prediction accuracy was relatively high since the training and validation data sets contained simulations of a vehicle performing the Rollover Steer driving maneuver at

different speeds. This is the case where the Nadaraya-Watson Kernel Regression model performs especially well and a further discussion is presented within Section 5.4.

Next, a simulation of a vehicle on a wet road following an S Turn with Clothoids at a target speed of 220 km/h was investigated. Prediction results are displayed in Figure 5.22. 121 out of 121 queries were correctly classified as Wet for a 100% accuracy. In addition, the average prediction time was 0.269 seconds.

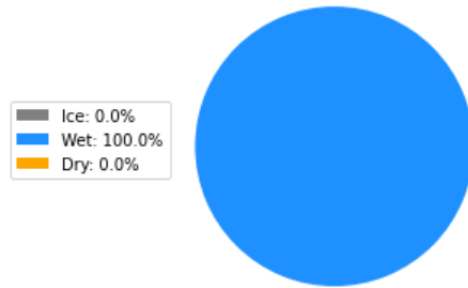


Figure 5.22: Nadaraya-Watson Estimation for 220 km/h S Turn with Clothoids (Wet)

The 100% prediction accuracy was also present within the predictions generated by the LSTM network and was due to the large target speed of 220 km/h. Measurements were more distinct between different road surfaces at higher speeds, which made it easier for both models to classify accurately.

A Side Drift driving maneuver with a target speed of 115 km/h on a wet road was also simulated and tested. Figure 5.23 displays a summary of the prediction results. 44 out of 61 queries were correctly classified as Wet for an accuracy of 72.13%, which was higher than the LSTM network. An average query time of 0.256 seconds was recorded.

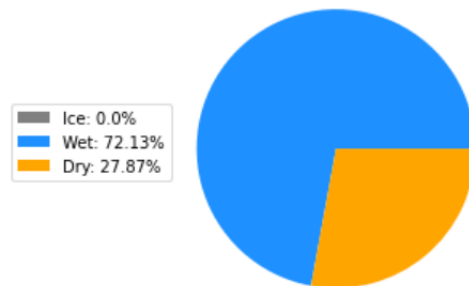


Figure 5.23: Nadaraya-Watson Estimation for 115 km/h Side Drift (Wet)

The Side Drift driving maneuver is distinct from any driving maneuver within the training and validation data sets. Therefore, the prediction accuracy for this driving maneuver is relatively low compared to other driving maneuvers within the test set.

In addition, a Sine Sweep Steer driving maneuver on a wet road with a target speed of 145 km/h was also simulated and tested. Prediction results are summarized in Figure 5.24. 277 out of 291 queries were correctly classified as Wet, yielding an accuracy of 95.19%, which was a higher prediction accuracy compared to the LSTM network. The average query prediction time was 0.256 seconds.

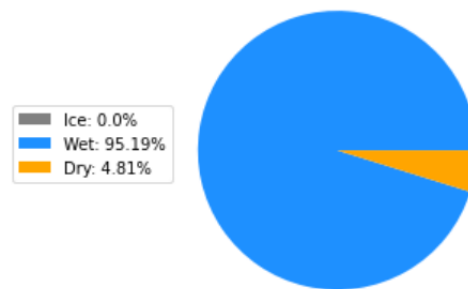


Figure 5.24: Nadaraya-Watson Estimation for 145 km/h Sine Sweep Steer (Wet)

A simulation of a vehicle performing a Double Lane Change driving maneuver on a dry surface with a target speed of 65 km/h was also present within the testing data set. Prediction results are summarized in Figure 5.25. All 106 queries were correctly classified as Dry, yielding a 100% accuracy, which was higher than the LSTM network. It is also worth noting that the average prediction time was 0.263 seconds.

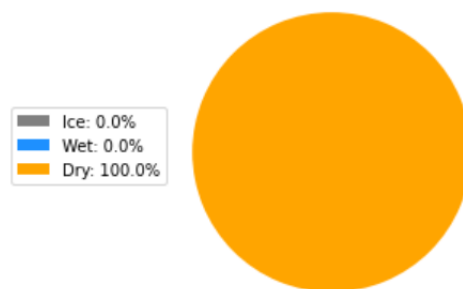


Figure 5.25: Nadaraya-Watson Estimation for 65 km/h DLC (Dry)

The testing data set also contained a simulation of the Increasing Speed driving maneuver performed on a dry surface. Figure 5.26 outlines the prediction results from this simulation. Out of a total of 191 queries, 133 were correctly classified as Dry, yielding an accuracy of 69.63%, which was lower compared to the LSTM network. The lower accuracy was because the Increasing Speed driving maneuver is different from any driving maneuver within the training data set. The average prediction time among all the queries was 0.234 seconds.

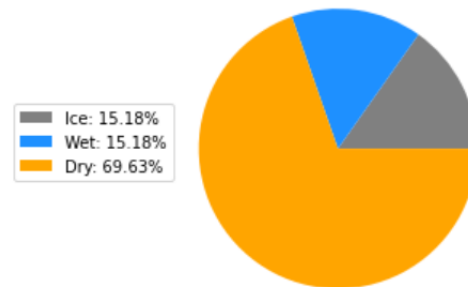


Figure 5.26: Nadaraya-Watson Estimation for Increasing Speed (Dry)

Next, a Side Drift driving maneuver was simulated on a dry road with a target speed of 135 km/h. A summary of prediction results is displayed in Figure 5.27. 49 out of 61 queries were correctly classified as Dry, yielding an 80.33% accuracy, which was higher than the LSTM network. In addition, the average prediction time was 0.254 seconds.

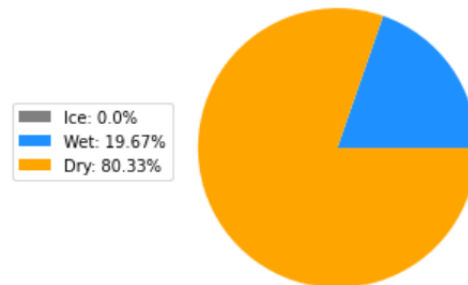


Figure 5.27: Nadaraya-Watson Estimation for 135 km/h Side Drift (Dry)

The prediction accuracy for this driving maneuver was relatively worse compared to other driving maneuvers within the test set that were close to those within the training set. This is similar to the predictions made for a Side Drift maneuver performed at 115 km/h



on a wet road. The Side Drift driving maneuver is different from any driving maneuver within the training data set.

Finally, the testing data set also contained a simulation of a vehicle on a dry road following an S Turn with Clothoids with a target speed of 95 km/h. Prediction results are displayed in Figure 5.28. 364 out of 369 queries were correctly classified as Wet. This results in a 98.64% accuracy, which was higher than the LSTM network. The average prediction time was 0.260 seconds.

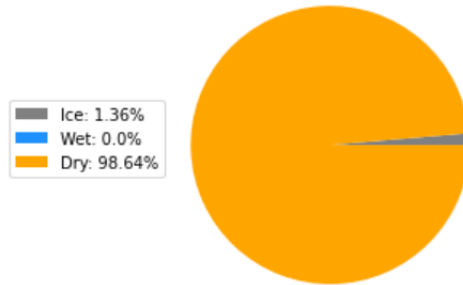


Figure 5.28: Nadaraya-Watson Estimation for 95 km/h S Turn with Clothoids (Dry)

A summary of accuracy measures from both the LSTM network and Nadaraya-Watson Kernel Regression models is displayed in Table 5.9.

Test Simulation	LSTM Accuracy	Nadaraya-Watson Accuracy
80 km/h Increasing Steer (Ice)	98.44%	89.10%
Accelerate and Brake (Ice)	69.29%	39.00%
240 km/h Straight Driving (Ice)	100.00%	95.32%
80 km/h Brake In Turn (Ice)	93.72%	39.79%
85 km/h Rollover Steer (Wet)	99.24%	100.00%
220 km/h S Turn with Clothoids (Wet)	100.00%	100.00%
115 km/h Side Drift (Wet)	65.57%	72.13%
145 km/h Sine Sweep Steer (Wet)	79.73%	95.19%
65 km/h Double Lane Change (Dry)	89.62%	100.00%
Increasing Speed (Dry)	75.39%	69.63%
135 km/h Side Drift (Dry)	75.41%	80.33%
95 km/h S Turn with Clothoids (Dry)	97.56%	98.64%

Table 5.9: Summary of Accuracy Measures

## 5.4 Comparison of Approaches

There are numerous benefits and drawbacks between the LSTM network and Nadaraya-Watson Kernel Regression models. Comparative advantages and disadvantages are highlighted and discussed throughout this section.

### Prediction Accuracy

The LSTM network was able to correctly classify 2321 out of 2575 queries across all twelve simulations within the testing data set. This yielded an overall accuracy of 90.14% for the LSTM network model. In comparison, the Nadaraya-Watson Kernel Regression model was able to correctly classify 2149 out of 2575 total queries throughout the entire testing data set. The overall accuracy of the Nadaraya-Watson Kernel Regression model was 83.46%.

Accuracy measures across individual testing data sets were generally comparable between the two methods. However, the largest difference occurred between the Brake In Turn as well as the Accelerate and Brake driving maneuvers. The LSTM network was able to classify the road surface correctly with an accuracy of 93.72% and 69.29%, respectively. However, this was not the case with the Nadaraya-Watson Kernel Regression model, where accuracy measures were 39.79% and 39.00%, respectively. This is a consequence of the Brake In Turn as well as the Accelerate and Brake driving maneuvers being drastically different from any other driving maneuver within the training and validation data sets. This can be seen throughout Appendix A, where different driving maneuvers are described. Without accounting for these two driving maneuvers, the accuracy of the Nadaraya-Watson Kernel Regression model improves to 92.34% from correctly classifying 1979 out of 2143 queries.

A neural network attempts to develop an approximation of a target function globally. Therefore, a properly trained neural network will perform better on new testing data that is drastically different from anything within the training or validation data sets. Conversely, the Nadaraya-Watson Kernel Regression model generates a local approximation of the target function around each query. Predictions are often more accurate on queries that are relatively similar to some data already within the training data set. However, the predictions become less accurate if the testing data deviates greatly from anything stored within the training data set.

Both of these trends were observed from the tests conducted. The LSTM network performed better on queries involving the driving maneuvers of Increasing Steer, Straight Driving, Increasing Speed, Brake In Turn, as well as Accelerate and Brake. These driving maneuvers were all different from any driving maneuvers present within the training data set. Conversely, the Nadaraya-Watson Kernel Regression model performed better on queries involving the driving maneuvers of Rollover Steer, Sine Sweep Steer, Side Drift, and Double Lane Change. It also performed better on queries from the test simulation of a vehicle driving along an S Turn with Clothoids at 95 km/h on a dry road. None of the simulations used within the testing data set were present within the training data as each were conducted with different target speeds. However, the Nadaraya-Watson Kernel Regression model yielded higher accuracy measures than the LSTM network model for queries generated from these driving maneuvers as there is a closer resemblance to data present within the training data set. The Side Drift driving maneuver was the only driving maneuver that was not present within the training data set where the Nadaraya-Watson Kernel Regression model outperformed the LSTM network model. However, this driving maneuver was still relatively similar to the Double Lane Change driving maneuver, which was present within the training data set. There was also one test simulation involving a vehicle driving along an S Turn with Clothoids at 220 km/h on a wet road where both the LSTM network and Nadaraya-Watson Kernel regression models were equally accurate.

### **Handling of Newly Acquired Data**

One of the largest comparative advantages of the Nadaraya-Watson Kernel Regression model is that there is no time required to retrain the model when new data is collected and inserted into the training data set. This is similar to all instance-based learning methods where the newly acquired data simply needs to be stored alongside the rest of the data until a prediction is to be made.

Conversely, an artificial neural network must be retrained to incorporate newly acquired data. Weights and biases must be adjusted to account for the new data alongside the rest of the training data set. In this thesis, the LSTM network took approximately 1 hour and 40 minutes to train on an Intel(R) Xeon(R) CPU with two cores at 2.20 GHz with 13 GB of DDR4 RAM.

## Required Storage

An implementation of the Nadaraya-Watson Kernel Regression model requires the storage of the entire training data set. This greatly increases the onboard vehicle storage required compared to an implementation of a trained neural network. The required storage will also continue to increase as more data is collected over time to account for additional driving scenarios. 4361 KB were required to save the entire training data set compressed into CSV format.

However, an artificial neural network does not encounter this problem of requiring a greater amount of onboard storage as the size of the training data set increases. The LSTM model is trained offline to determine appropriate values for all associated weights and biases. Implementation of the LSTM network model onto a vehicle does not include the storage of any training data. Therefore, the required storage would only scale if the LSTM network model itself increased in the number trainable parameters by modifying its architecture. Only 3402 KB were required to save the trained LSTM network model used in this thesis.

## Online Query Prediction Time

The average prediction time across all 2575 test points was 0.218 seconds for the Nadaraya-Watson Kernel Regression model. This is a reasonably fast rate for deployment onto a physical vehicle. However, this value would increase if more training data is collected to account for additional driving scenarios. This is due to the closest point selection component within the online portion of the estimation scheme. Algorithm 4, which was implemented to determine the K closest points to a new query, has a linear run time complexity. Therefore, Algorithm 4 and hence the overall online portion of the Nadaraya-Watson Kernel Regression model, would take longer to execute as more training data is stored.

Conversely, the average prediction time was only 0.000544 seconds for the LSTM network model across all 2575 test points. This is a much faster rate that would remain largely consistent as more training data is collected over time.

Overall, the LSTM network model prediction time was considerably lower than predictions from the Nadaraya-Watson Kernel Regression model. The difference could be minimized if further data reduction methods were applied on the training data set. However, the prediction time associated with the Nadaraya-Watson Kernel Regression model would still continue to scale as more data is collected, which is not the case with the LSTM network model.

## Prediction Interpretability

A common problem with artificial neural networks is the lack of interpretability with predictions on new queries [55]. An artificial neural network is an approximation of a target function after all the weights and biases have been tuned. However, the contributions of each individual neuron to the final prediction on a trained network is often unclear, even if the predictions are highly accurate. This can be problematic in safety-critical applications, such as autonomous vehicles, as it is desirable to understand the exact reasoning behind each prediction generated by an artificial neural network.

Conversely, predictions generated by the Nadaraya-Watson Kernel Regression method can be interpreted more easily. Data points within a local neighbourhood around each query can be specifically identified and traced back to the particular training data set of origin. The final prediction is interpreted as a weighting between these data points with larger weights associated with closer data points.

A summary of the comparison between the LSTM network and Nadaraya-Watson Kernel Regression model results are displayed in Table 5.10.

Feature	LSTM Network	Nadaraya-Watson Kernel Regression
Accuracy	2321/2575 = 90.1%	With Drastically Different Driving Maneuvers: 2149/2575 = 83.5%
		Without Drastically Different Driving Maneuvers: 1979/2143 = 92.3%
Training Time	1 hour, 39 minutes, 18 seconds	0 seconds
Required Storage	3402 KB [Model Size (Architecture + Weights)]	4361 KB [Data Set Size]
Average Prediction Time (Across all 2575 Queries)	0.000544 seconds	0.218 seconds

Table 5.10: Comparison of Methods Summary

# Chapter 6

## Conclusions and Future Work

### 6.1 Conclusions

Two different machine learning-based methods were investigated throughout this thesis for the purpose of road surface estimation. The first method was to create an approximation of the target function globally using an artificial neural network. Specifically, LSTM units were used to capture temporal relationships within the training data and to mitigate the vanishing gradient problem common with deep neural networks. The second method was to create local approximations of the target function around each query using Nadaraya-Watson Kernel Regression, which is an instance-based learning method.

Both methods were first applied to estimate friction on an inverted pendulum system. A numerical value representing the friction force between the cart and the track is estimated through posing the task as a regression problem. It was found that both methods were capable of estimating friction forces reasonably well. The prediction time was longer using Nadaraya-Watson Kernel Regression, and the storage required for implementation was also larger. However, there was no training time required, similar to all instance-based learning methods.

Next, the road surface estimation task was posed as a classification problem for a vehicle driving on different road surfaces. Each road surface was classified as either Ice, Wet, or Dry. Measurement data from sensors commonly available on commercial vehicles and physically relevant towards tire-road friction was gathered using CarSim. Numerous simulations for different driving maneuvers and road conditions were created for data collection.

It was found that the LSTM network model was more accurate at making predictions involving driving maneuvers that were drastically different from every maneuver within the training data set. In particular, testing data involving the Brake In Turn or Accelerate and Brake driving maneuvers were classified correctly far more often with the LSTM network model. In addition, the prediction accuracy was slightly higher for the LSTM network model on other driving maneuvers that were not present within the training data set. However, the Nadaraya-Watson Kernel Regression model generally performed better on testing data involving driving maneuvers that were present in the training data set. Overall, the LSTM network and Nadaraya-Watson Kernel Regression models classified 90.14% and 83.46% of all queries correctly throughout the entire testing data set, respectively. However, the accuracy of the Nadaraya-Watson Kernel Regression model improves to 92.34% after excluding the Brake In Turn as well as Accelerate and Brake driving maneuvers.

The developed LSTM network and Nadarya-Watson Kernel Regression models in this thesis were both capable of classifying road conditions to a reasonable degree of accuracy. Different benefits and drawbacks were characteristic of each method. The LSTM network was able to make accurate predictions on queries drastically different from the training data, required minimal storage for implementation, and had relatively short prediction times. However, it must be retrained to incorporate new data which could be a lengthy process, and lacked interpretability in generated predictions. Conversely, the Nadaraya-Watson Kernel Regression model was unable to make accurate predictions on queries drastically different from the training data, required more storage for implementation onto a physical vehicle, and had longer prediction times. However, no training was required to incorporate newly collected data into the model, and predictions were more easily interpretable.

Overall, both artificial neural networks and instance-based learning methods can be sufficient for road surface estimation. In general, different performance requirements may be used to dictate the more appropriate choice for each specific application. For example, Nadaraya-Watson Kernel Regression may be appropriate if the operational design domain is constrained, it is feasible to both collect and store data regarding nearly all scenarios, and the prediction time requirement is lenient. However, an artificial neural network may be the more appropriate choice if the operational design domain is too large to collect data regarding all plausible scenarios, or if the environment is continually changing. It is also more appropriate if the storage constraints or prediction time requirements are more strict.

## 6.2 Future Work

In general, data-driven models typically benefit from the collection of a larger quantity of high quality data. Further patterns could be captured within the LSTM network model to improve classification accuracy on unseen driving maneuvers. In addition, the instance-based Nadaraya-Watson Kernel Regression model becomes more accurate when similar data points have been observed within the training data set. Therefore, collecting additional data would improve the results presented in the thesis.

Next, the trained LSTM network and Nadaraya-Watson Kernel Regression models have not been experimentally implemented on a physical vehicle. Real-world data collected from physical sensors are much noisier than simulation data. Additional tuning may be required in both models to continue generating road surface classifications with a reasonable accuracy. In addition, further complications that may arise with vehicle integration may be worth investigating.

One of the largest limitations imposed onto the work of this thesis was to only use sensor measurements commonly available across commercial vehicles. This constrained potential work with respect to feature engineering. However, machine learning-based models for road surface classification may be further improved if additional features were used such as tire forces, slip angles, and camber angles. It may be worth investigating potential accuracy improvements from using additional features. However, results from this study could not be directly implemented onto current commercial vehicles.

Additional feature selection methods could be applied to potentially narrow down the 15 features selected in this study. One potential method is to calculate the Pearson correlation between each of the 15 features and the output variable of the friction coefficient. However, the Pearson correlation coefficient only represents the strength of a linear relationship between the variables, and would not identify nonlinear relationships. The expression to evaluate the Pearson correlation coefficient for sample data  $(x_i, y_i)$  is

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}, \quad (6.1)$$

where  $\bar{x}$  and  $\bar{y}$  represent the arithmetic mean, respectively.

Finally, both the LSTM network and Nadaraya-Watson Kernel Regression models could be extended to make predictions for additional labels. For example, road surface classifications could be extended to include snow, sand, grass, or other less common road surface conditions. Furthermore, additional data could be collected to incorporate road grade and bank angles as well. Training data could also be collected for other vehicles beyond the C-Class Hatchback to develop more robust road surface classification models.



# References

- [1] V. Kishore Ayyadevara. *Neural networks with Keras cookbook : over 70 recipes leveraging deep learning techniques across image, text, audio, and game bots*. Packt Publishing Ltd, Birmingham, 1st edition, 2019.
- [2] Karl Berntorp. Online bayesian tire-friction learning by gaussian-process state-space models. *IFAC-PapersOnLine*, 53(2):13939–13944, 2020. 21st IFAC World Congress.
- [3] Sue Ann Campbell, Stephanie Crawford, and Kirsten Morris. Friction and the inverted pendulum stabilization problem. *ASME Journal of dynamic systems, measurement, and control*, 130(5):0545021–0545027, 2008.
- [4] Langat Reuben Cheruiyot. Local linear regression estimator on the boundary correction in nonparametric regression estimation. *Journal of Statistical Theory and Applications*, 19:460–471, 2020.
- [5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals, and systems*, 2(4):303–314, 1989.
- [6] Xiaotong Dong, Yi Jiang, Zhou Zhong, Wei Zeng, and Wei Liu. An improved rollover index based on bp neural network for hydropneumatic suspension vehicles. *Mathematical problems in engineering*, 2018:1–15, 2018.
- [7] Weijiang Feng, Naiyang Guan, Yuan Li, Xiang Zhang, and Zhigang Luo. Audio visual speech recognition with multimodal recurrent neural networks. In *2017 International Joint Conference On Neural Networks (IJCNN)*, IEEE International Joint Conference on Neural Networks (IJCNN), pages 681–688, New York, 2017. IEEE.
- [8] Eduardo García-Portugués. *Notes for Predictive Modeling*. 2022. Version 5.9.9. ISBN 978-84-09-29679-8. <https://bookdown.org/egarpor/PM-UC3M/>.

- [9] João Gonçalves and Jorge Ambrósio. Road vehicle modeling requirements for optimization of ride and handling. *Multibody System Dynamics*, 13:3–23, 02 2005.
- [10] Ian. Goodfellow. *Deep learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 2016.
- [11] Torben Graber, Stefan Lupberger, Michael Unterreiner, and Dieter Schramm. A hybrid approach to side-slip angle estimation with recurrent neural networks and kinematic vehicle models. *IEEE transactions on intelligent vehicles*, 4(1):39–47, 2019.
- [12] Fredrik Gustafsson. Slip-based tire-road friction estimation. *Automatica (Oxford)*, 33(6):1087–1099, 1997.
- [13] Javier García Guzmán, Lisardo Prieto González, Jonatan Pajares Redondo, Mat Max Montalvo Martínez, and María Jesús L. Boada. Real-time vehicle roll angle estimation based on neural networks in iot low-cost devices. *Sensors (Basel, Switzerland)*, 18(7):2188–, 2018.
- [14] Amin Habibnejad Korayem. *State and Parameter Estimation of Vehicle-Trailer Systems*. PhD thesis, MME Department, University of Waterloo, 2021.
- [15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [16] Bin Huang, Xiang Fu, Sen Wu, and Song Huang. Calculation algorithm of tire-road friction coefficient based on limited-memory adaptive extended kalman filter. *Mathematical problems in engineering*, 2019:1–14, 2019.
- [17] Meysam Khaleghian, Omid Ghasemalizadeh, Saied Taheri, and Gerardo Flintsch. A combination of intelligent tire and vehicle dynamic based algorithm to estimate the tire-road friction. *SAE International Journal of Passenger Cars - Mechanical Systems*, 12, 04 2019.
- [18] Dae Jung Kim, Jin Sung Kim, Seung-Hi Lee, and Chung Choo Chung. A comparative study of estimating road surface condition using support vector machine and deep neural networ. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1066–1071. IEEE, 2019.
- [19] Min-Hyun Kim, Jongchan Park, and Seibum Choi. Road type identification ahead of the tire using d-cnn and reflected ultrasonic signals. *International journal of automotive technology*, 22(1):47–54, 2021.

- [20] Moad Kissai. *Fundamentals on Vehicle and Tyre Modelling*, pages 1–59. Springer International Publishing, Cham, 2022.
- [21] Raphael Linus Levien. *From Spiral to Spline: Optimal Techniques in Interactive Curve Design*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2009.
- [22] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameeet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.
- [23] Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current Opinion in Neurobiology*, 55:82–89, 2019.
- [24] Jadranko Matusko, Ivan Petrovic, and Nedjeljko Peric. Neural network based tire/road friction force estimation. *Engineering applications of artificial intelligence*, 21(3):442–456, 2008.
- [25] William F. Milliken. *Race car vehicle dynamics*. Society of Automotive Engineers, Warrendale, PA, 1995.
- [26] Tom Mitchell. *Machine learning*. McGraw-Hill international editions. McGraw-Hill Publishing Company, New York, 1997.
- [27] Michael A. Nielsen. How the backpropagation algorithm works. In *Neural Networks and Deep Learning*. Determination Press, 2015.
- [28] Tommaso Novi, Renzo Capitani, and Claudio Annicchiarico. An integrated artificial neural network–unscented kalman filter vehicle sideslip angle estimation based on inertial measurement unit measurements. *Proceedings of the Institution of Mechanical Engineers. Part D, Journal of automobile engineering*, 233(7):1864–1878, 2019.
- [29] Levon Nurbekyan, Wanzhou Lei, and Yunan Yang. Efficient natural gradient descent methods for large-scale optimization problems, 2022.
- [30] Tom O’Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. <https://github.com/keras-team/keras-tuner>, 2019.
- [31] Ghazaleh Panahandeh, Erik Ek, and Nasser Mohammadiha. Road friction estimation for connected vehicles using supervised machine learning. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1262–1267. IEEE, 2017.

- [32] W.R Pasterkamp and H.B Pacejka. Optimal design of neural networks for estimation of tyre/road friction. *Vehicle system dynamics*, 29(sup1):312–321, 1998.
- [33] R Rajamani, G Phanomchoeng, D Piyabongkarn, and J. Y Lew. Algorithms for real-time estimation of individual wheel tire-road friction coefficients. *IEEE/ASME transactions on mechatronics*, 17(6):1183–1195, 2012.
- [34] A Rakotomamonjy, R Le Riche, D Gualandris, and Z Harchaoui. A comparison of statistical learning approaches for engine torque estimation. *Control engineering practice*, 16(1):43–55, 2008.
- [35] Enrico Regolin and Antonella Ferrara. Svm classification and Kalman filter based estimation of the tire-road friction curve. *IFAC-PapersOnLine*, 50(1):3382–3387, 2017. 20th IFAC World Congress.
- [36] Alexandre M Ribeiro, Alexandra Moutinho, André R Fioravanti, and Ely C de Paiva. Estimation of tire–road friction for road vehicles: a time delay neural network approach. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 42(1):1–12, 2020.
- [37] Sohini Roychowdhury, Minming Zhao, Andreas Wallin, Niklas Ohlsson, and Mats Jonasson. Machine learning models for road surface and friction estimation using front-camera images. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.
- [38] Eldar Sabanovič, Vidas Žuraulis, Olegas Prentkovskis, and Viktor Skrickij. Identification of road-surface type using deep neural networks for friction coefficient estimation. *Sensors (Basel, Switzerland)*, 20(3):612–, 2020.
- [39] Stefania Santini, Nicola Albarella, Vincenzo Maria Arricale, Renato Brancati, and Aleksandr Sakhnevych. On-board road friction estimation technique for autonomous driving vehicle-following maneuvers. *Applied sciences*, 11(5):1–27, 2021.
- [40] Anton Maximilian Schafer and Hans-Gorg Zimmermann. Recurrent neural networks are universal approximators. *International journal of neural systems*, 17(4):253–263, 2007.
- [41] Liang Shao, Cornelia Lex, Andreas Hackl, and Arno Eichberger. Road friction estimation using recursive total least squares. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 533–538. IEEE, 2016.

- [42] Mohd Ibrahim Shapiai, Zuwairie Ibrahim, Marzuki Khalid, Lee Wen Jau, and Vladimir Pavlovich. A non-linear function approximation from small samples based on Nadaraya-Watson kernel regression. In *2010 2nd International Conference on Computational Intelligence, Communication Systems and Networks*, pages 28–32. IEEE, 2010.
- [43] Jaekwan Shin and Ikjin Lee. Reliability analysis and reliability-based design optimization of roadway horizontal curves using a first-order reliability method. *Engineering Optimization*, 47:622–641, 03 2015.
- [44] Seungmok Song, Kyushik Min, Jongwon Park, Hayoung Kim, and Kunsoo Huh. Estimating the maximum road friction coefficient with uncertainty using deep learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3156–3161. IEEE, 2018.
- [45] Tao Song, Hongliang Zhou, and Haifeng Liu. Road friction coefficient estimation based on bp neural network. In *2017 36th Chinese Control Conference (CCC)*, pages 9491–9496. Technical Committee on Control Theory, CAA, 2017.
- [46] Stephen J. Wright Suvrit Sra, Sebastian Nowozin. *Optimization for machine learning*. Neural information processing series. MIT Press, Cambridge, Mass, 2012.
- [47] Kashvi Taunk, Sanjukta De, Srishti Verma, and Aleena Swetapadma. A brief review of nearest neighbor algorithm for learning and classification. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 1255–1260, 2019.
- [48] Sina Torabi, Mattias Wahde, and Pitoyo Hartono. Road grade and vehicle mass estimation for heavy-duty vehicles using feedforward neural networks. In *2019 4th International Conference on Intelligent Transportation Engineering (ICITE)*, pages 316–321. IEEE, 2019.
- [49] Trang Thi Kieu Tran, Sayed M. Bateni, Seo Jin Ki, and Hamidreza Vosoughifar. A review of neural networks for air temperature forecasting. *Water*, 13(9), 2021.
- [50] Hamit Taner Unal and Fatih Basciftci. Evolutionary design of neural network architectures: a review of three decades of research. *The Artificial intelligence review*, 55(3):1723–1802, 2021.

- [51] Jorge Villagra, Brigitte d’Andréa Novel, Michel Fliess, and Hugues Mounier. A diagnosis-based approach for tire–road forces and maximum friction estimation. *Control engineering practice*, 19(2):174–184, 2011.
- [52] Chris C Ward and Karl Iagnemma. Speed-independent vibration-based terrain classification for passenger vehicles. *Vehicle system dynamics*, 47(9):1095–1113, 2009.
- [53] Nan Xu, Yanjun Huang, Hassan Askari, and Zepeng Tang. Tire slip angle estimation based on the intelligent tire technology. *IEEE transactions on vehicular technology*, 70(3):2239–2249, 2021.
- [54] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing (Amsterdam)*, 415:295–316, 2020.
- [55] Yu Zhang, Peter Tino, Ales Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions On Emerging Topics In Computational Intelligence*, 5(5):726–742, 2021.

# APPENDICES

# Appendix A

## Driving Maneuvers

Driving maneuvers listed throughout Chapter 5 are described in further detail throughout Appendix A.

### Double Lane Change

A double lane change consists of two main steering actions. The first is to steer towards the left such that the vehicle switches lanes. The second is to steer back towards the right to re-enter the original lane in which it started. A plot of the lateral offset of a vehicle from the original lane is displayed in Figure A.1.

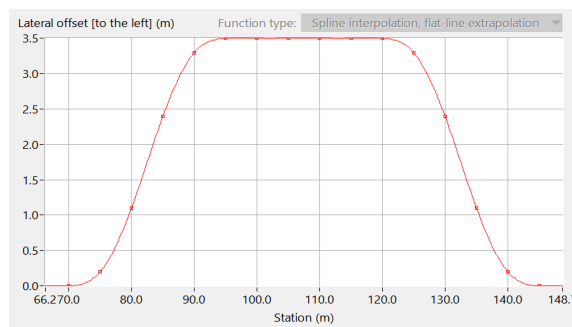


Figure A.1: Double Lane Change Driving Maneuver



## Sine Sweep Steer

This driving maneuver consists of an aggressive steering input on a flat test pad. The steering input follows a sinusoidal pattern with a decreasing amplitude over time. An example of the steering input command is displayed in Figure A.2.

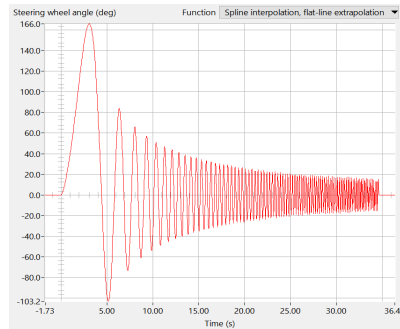


Figure A.2: Sine Sweep Steer Driving Maneuver

## Rollover Steer

This is another aggressive driving maneuver that consists of two main steering inputs. First, the steering wheel is turned left and held in that position for a brief amount of time. Next, the steering wheel is aggressively turned to the right and held in that position until the end of the driving maneuver. An example plot of the steering wheel plotted against time is displayed in Figure A.3.

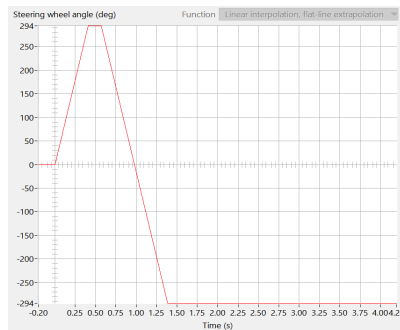


Figure A.3: Rollover Steer Driving Maneuver

## S Turn with Clothoids

A clothoid, also commonly referred to as an Euler Spiral, is a curve in which the curvature varies linearly along the arc length. More formally, clothoids are two-parameter splines that achieve  $G^2$  continuity [21]. Clothoids are often used in railroad transitions and several other applications such as local path planning for an autonomous vehicle. An S-shaped road that consists of two clothoids is generated as the path for the vehicle to follow under this driving maneuver is displayed in Figure A.4.

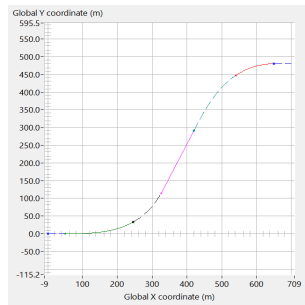


Figure A.4: S Turn with Clothoids Driving Maneuver

## Straight Driving

This is a very basic maneuver that simply consists of a vehicle starting from rest and driving along a straight road with no steering input. The specified speed is maintained once it has been reached. Figure A.5 demonstrates that no lateral deviation from the straight road occurs throughout this driving maneuver.

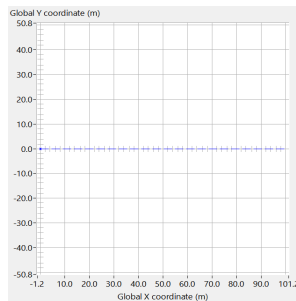


Figure A.5: Straight Driving Maneuver

## Side Drift

This driving maneuver is similar to the Double Lane Change. However, the main difference is that there is no period of straight driving once the first lane change is complete. Instead, the second lane change back into the original lane is performed immediately after the first lane change. Overall, this maneuver appears as if the vehicle is drifting to the side briefly. Figure A.6 displays the lateral offset from the original lane.

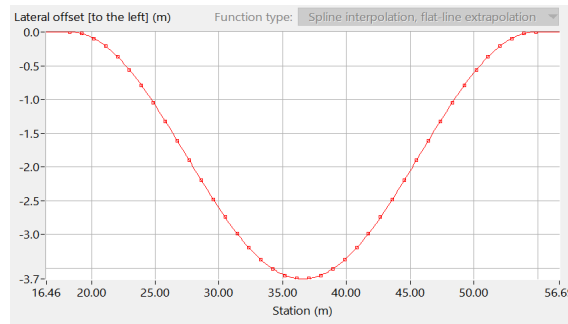


Figure A.6: Single Lane Change Maneuver

## Increasing Steer

This driving maneuver begins with no steering input similar to the Straight Driving maneuver. However, the steering wheel is turned aggressively to the left after a certain period of time. An example plot of the steering wheel angle vs. time is displayed in Figure A.7.

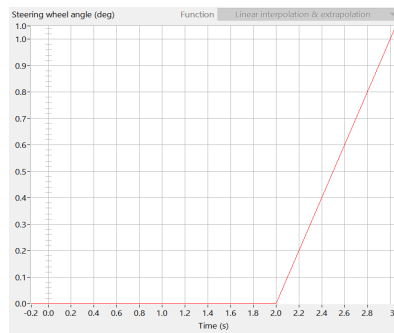


Figure A.7: Increasing Steer Driving Maneuver

## Increasing Speed

The vehicle starts from rest during this driving maneuver. Afterwards, the vehicle slowly begins to accelerate up until the maximum throttle is reached. A plot of the throttle command vs. time is demonstrated in Figure A.8.

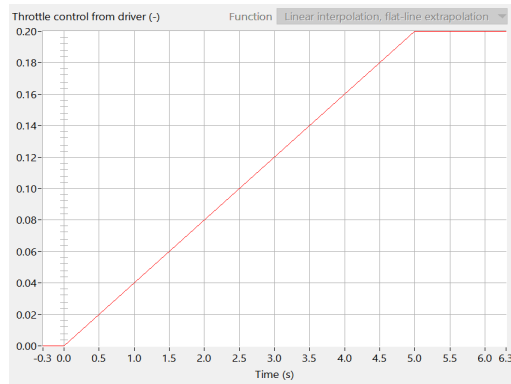


Figure A.8: Increasing Speed Driving Maneuver

## Brake In Turn

The vehicle starts at the specified target speed during this driving maneuver. Afterwards, the vehicle applies the brakes and adjusts the steering wheel at the same time. A plot of the brake pedal command vs. time is demonstrated in Figure A.9.

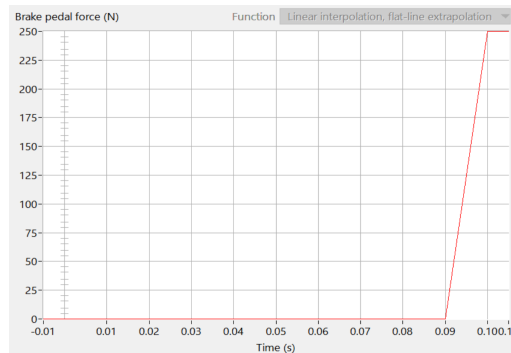


Figure A.9: Brake In Turn Driving Maneuver

## Accelerate and Brake

The vehicle initially starts at rest during this driving maneuver. The throttle is then pressed and linearly increased in position for about 10 seconds. The brakes are then applied after about 15 seconds, from the start of the maneuver, until the vehicle comes to rest once again. A plot of the throttle and brake commands vs. time are displayed in Figure A.10 and Figure A.11 respectively.

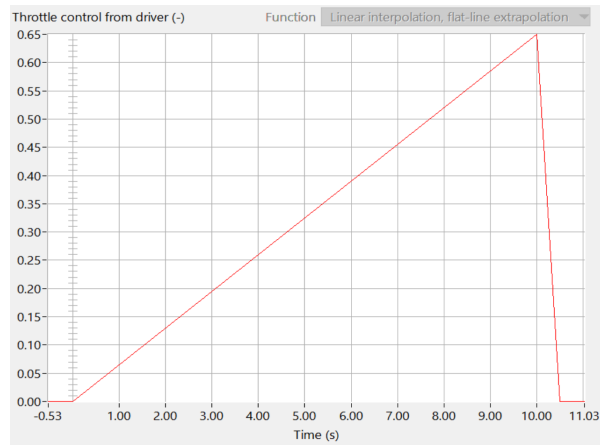


Figure A.10: Throttle Command Throughout Accelerate and Brake Driving Maneuver

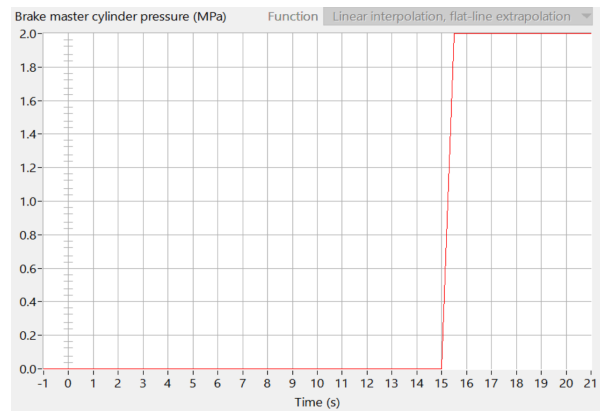


Figure A.11: Brake Command Throughout Accelerate and Brake Driving Maneuver

# Appendix B

## Measurement Data from Test Data Set

The measurement data used as input data to generate predictions from both the LSTM network and Nadaraya-Watson Kernel Regression models are displayed throughout this section of the Appendix.

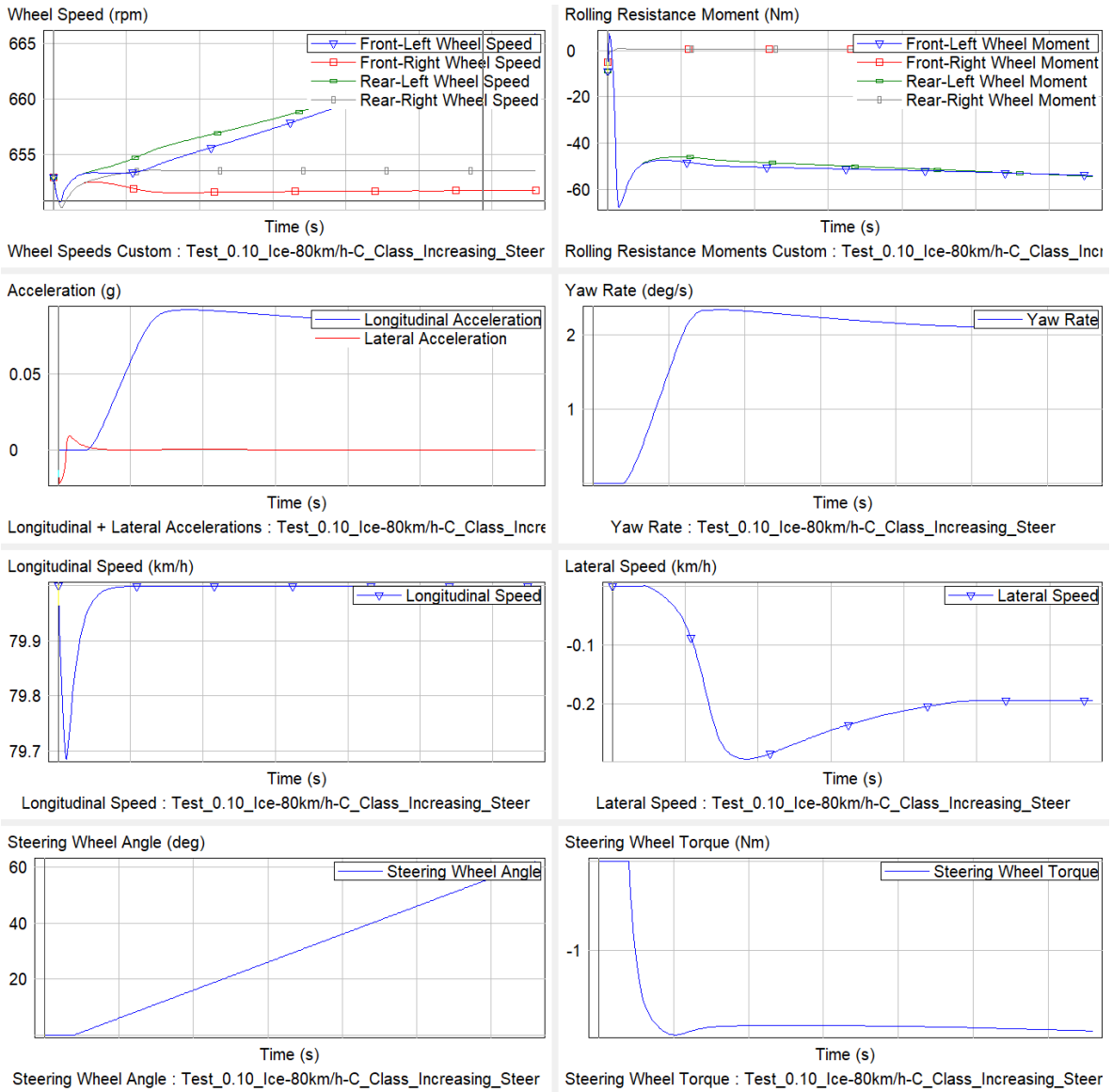


Figure B.1: Measurement Data for 80 km/h Increasing Steer (Ice) Simulation

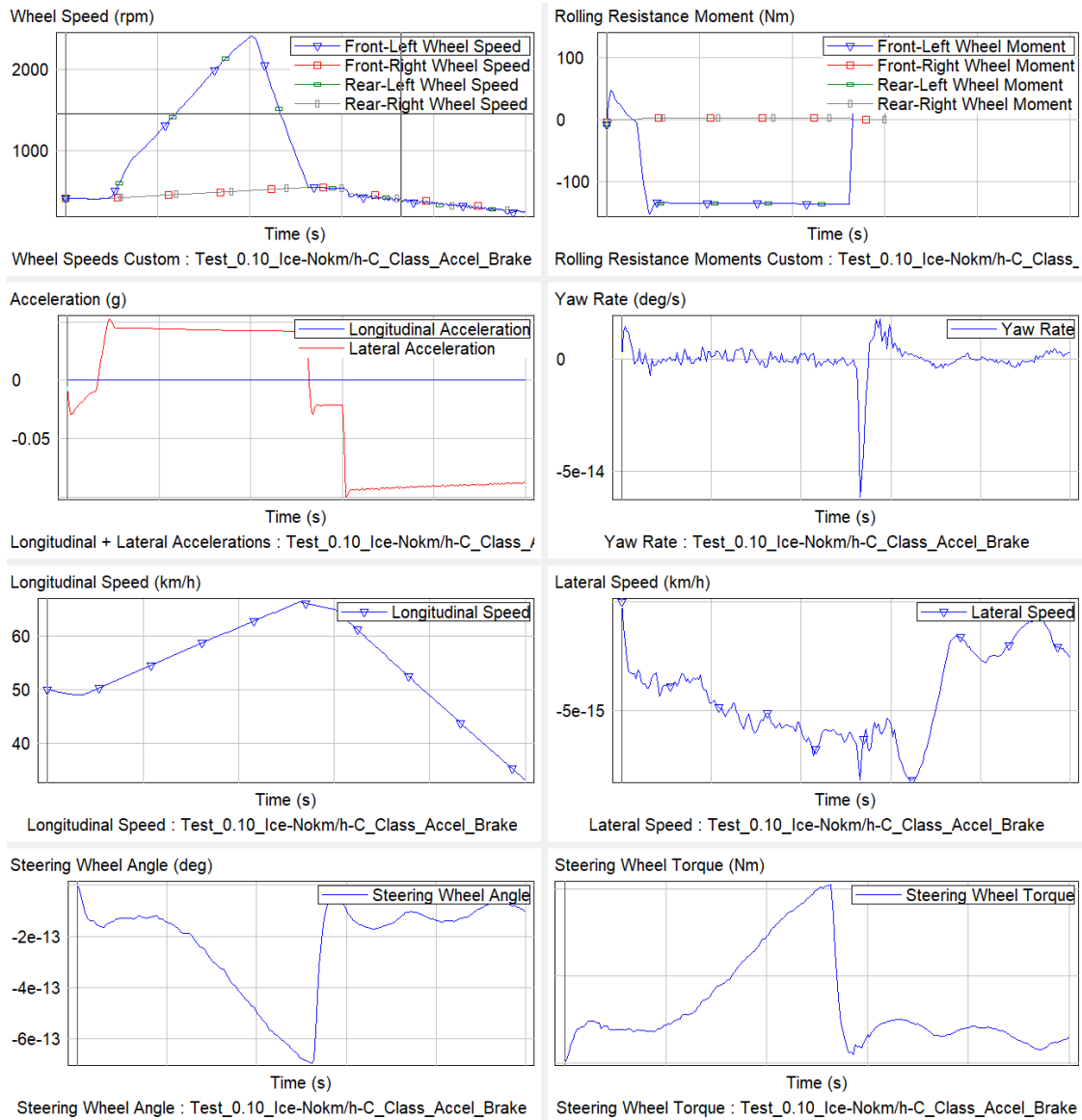


Figure B.2: Measurement Data for Accelerate and Brake (Ice) Simulation



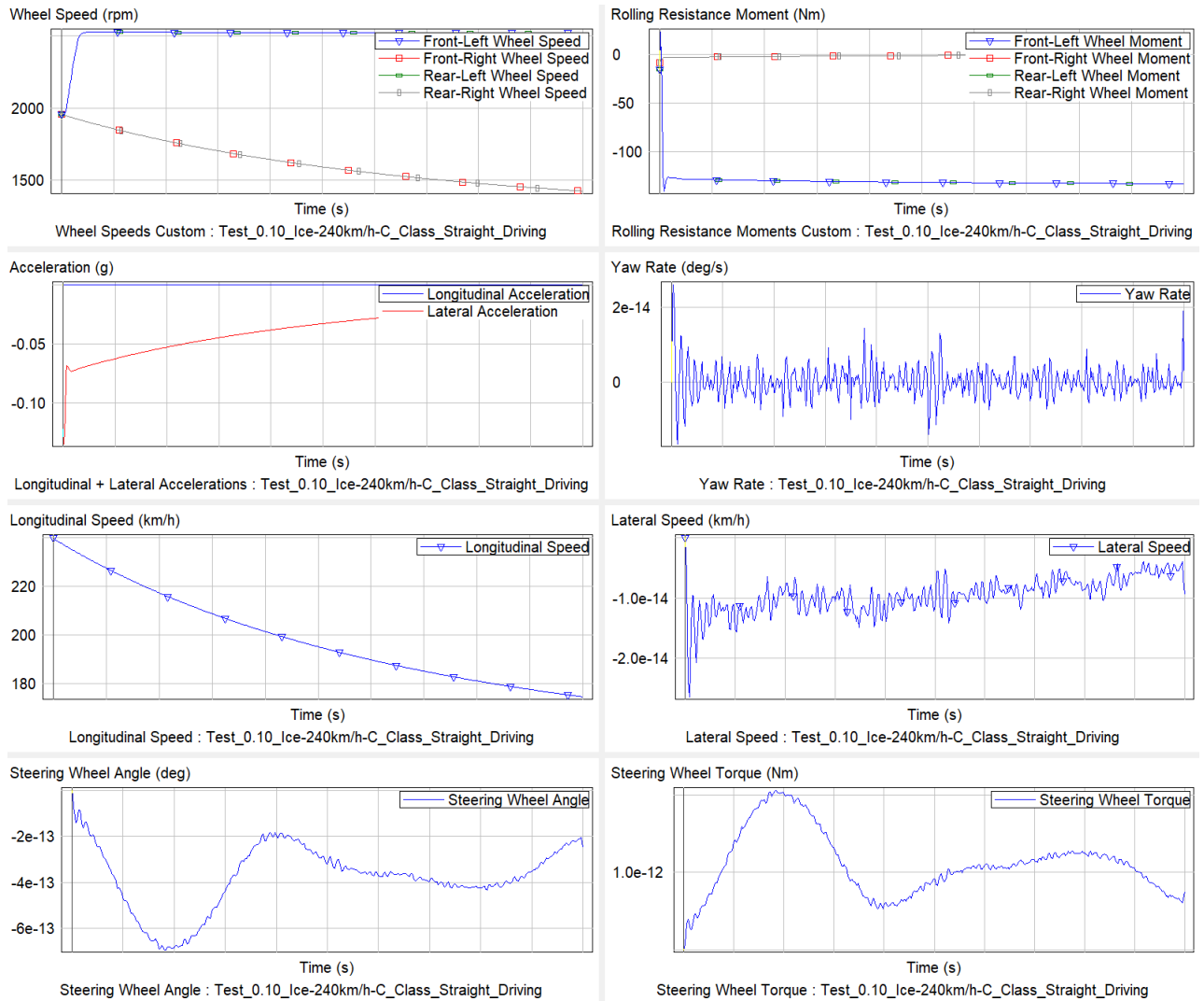


Figure B.3: Measurement Data for 240 km/h Straight Driving (Ice) Simulation

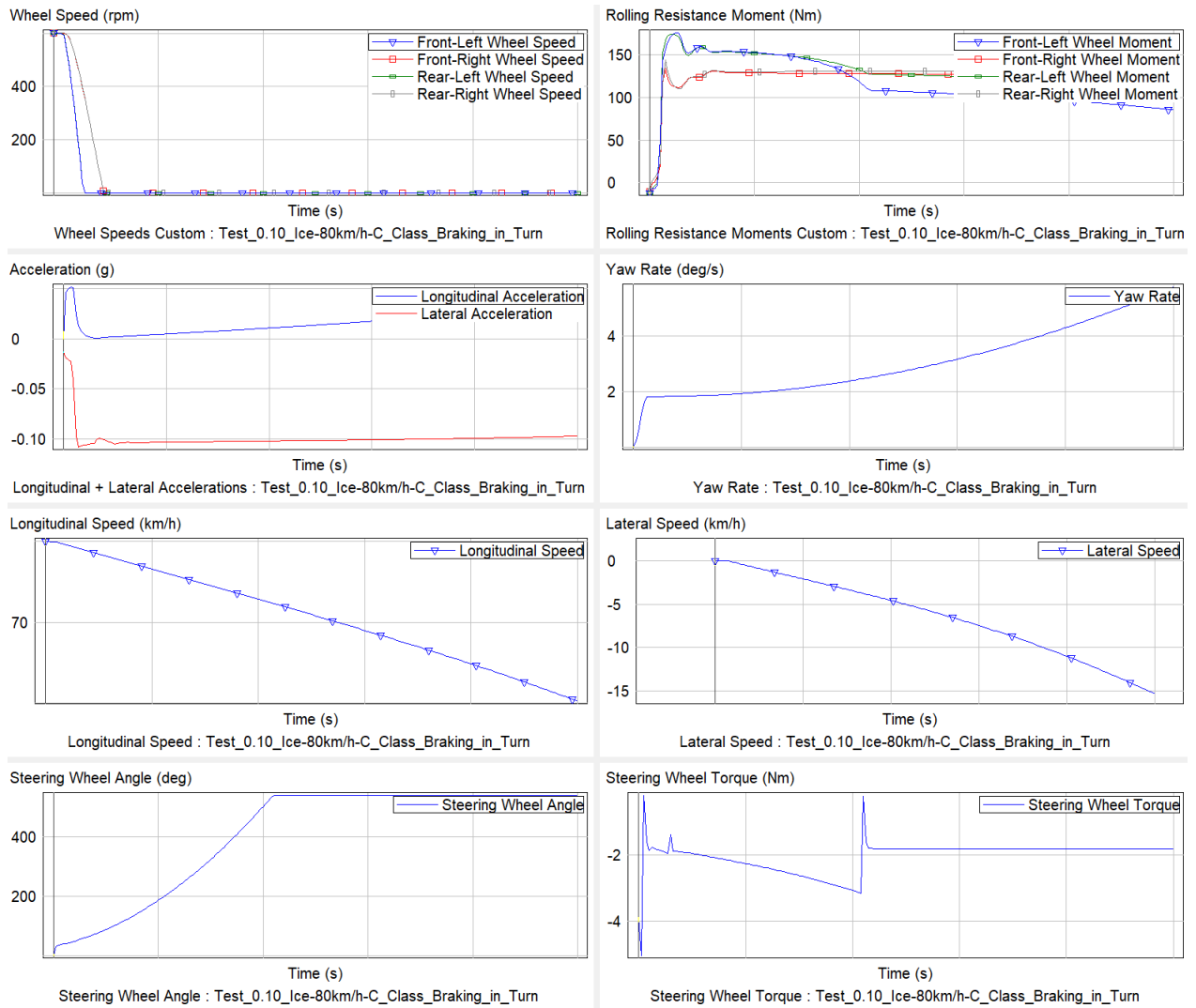


Figure B.4: Measurement Data for 80 km/h Brake In Turn (Ice) Simulation

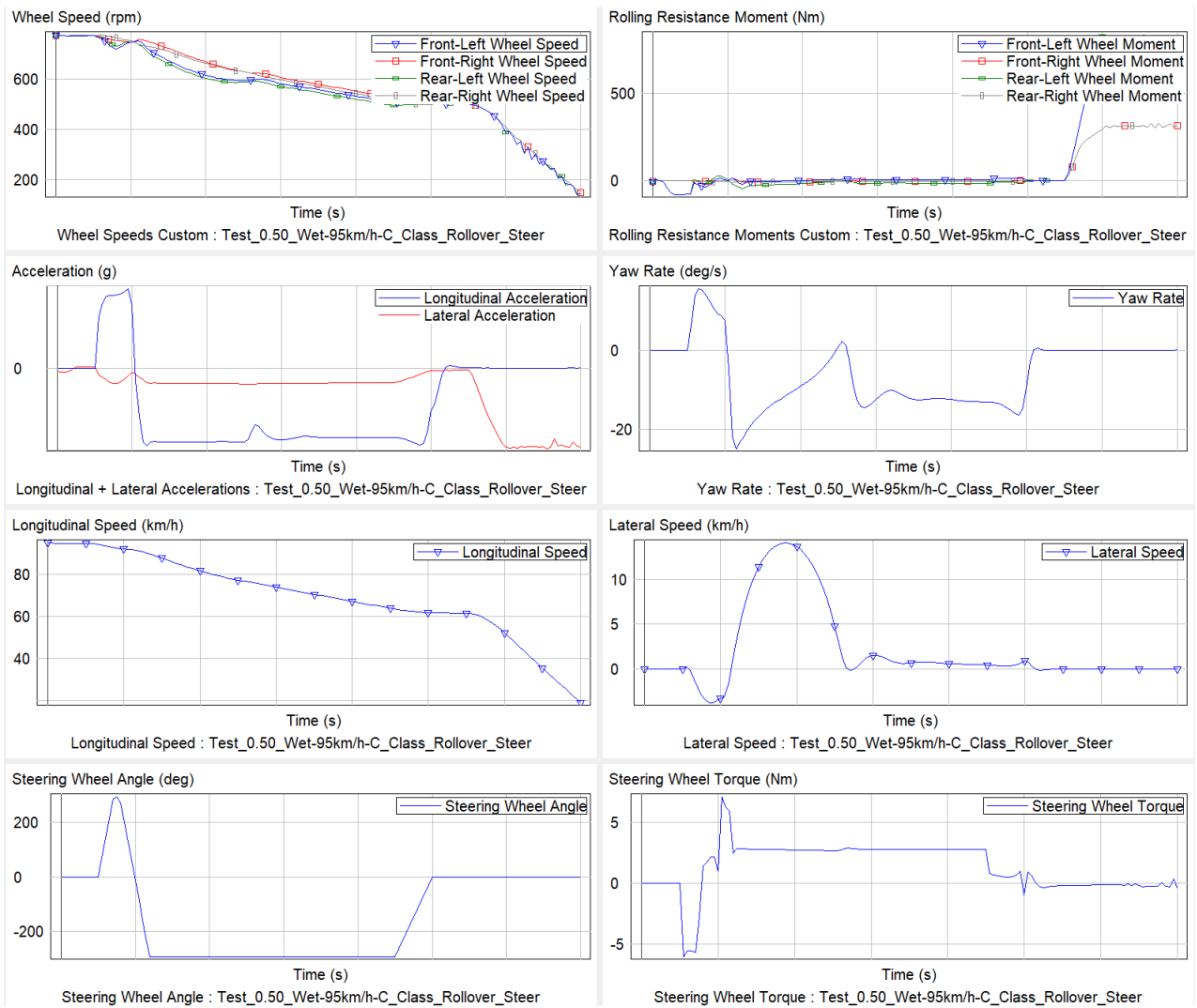


Figure B.5: Measurement Data for 85 km/h Rollover Steer (Wet) Simulation

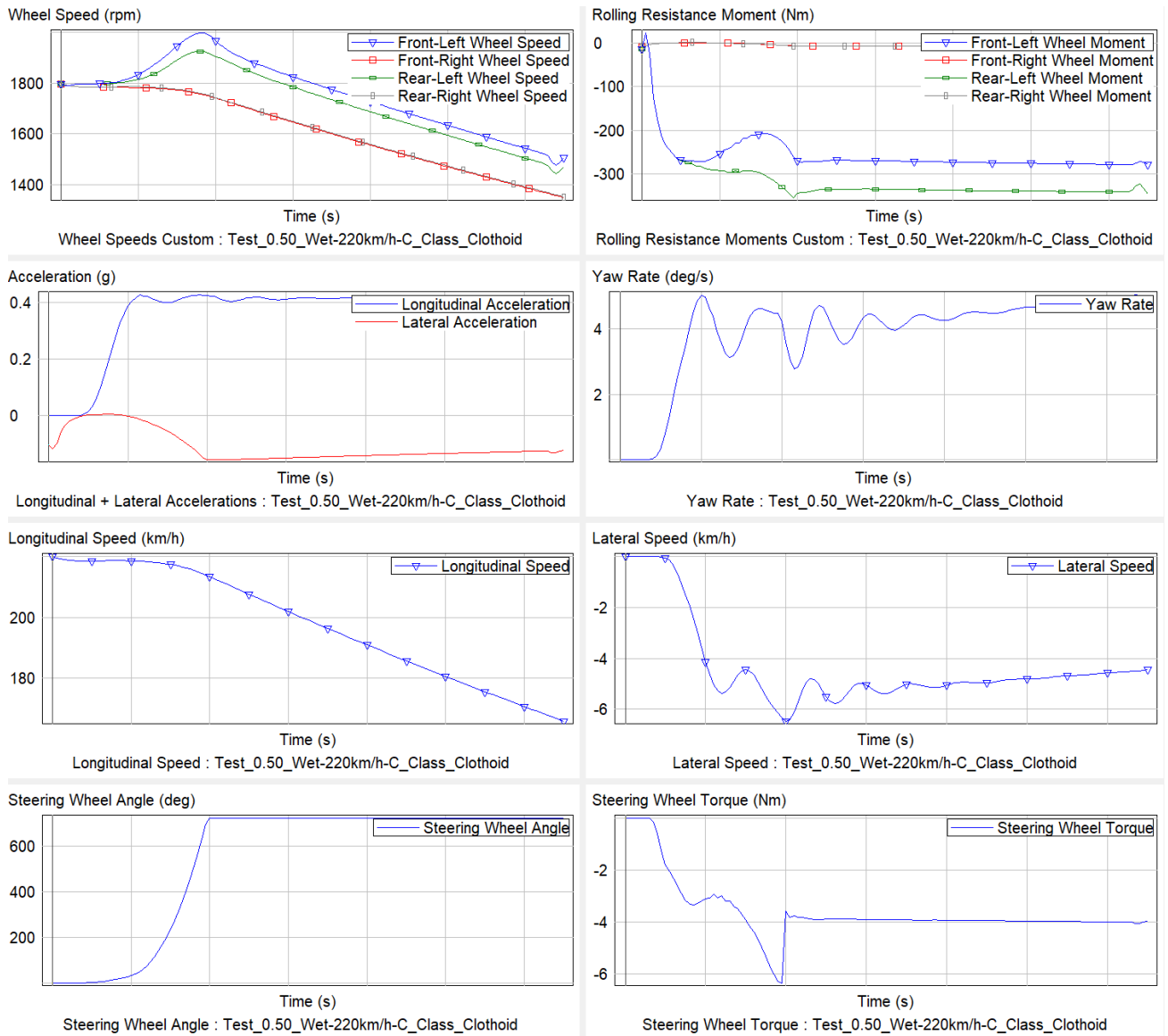


Figure B.6: Measurement Data for 220 km/h S Turn with Clothoid (Wet) Simulation

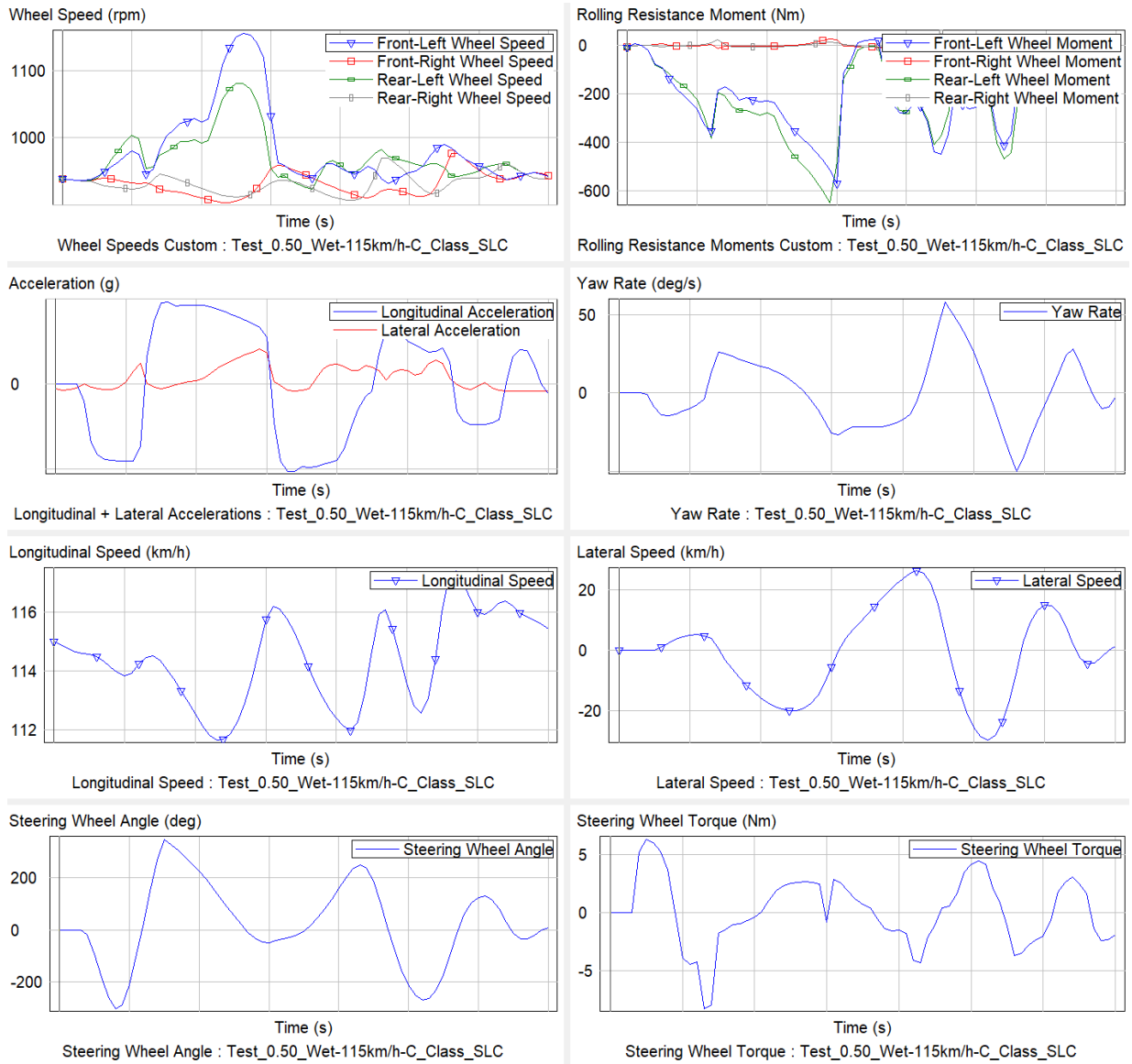


Figure B.7: Measurement Data for 115 km/h Side Drift (Wet) Simulation

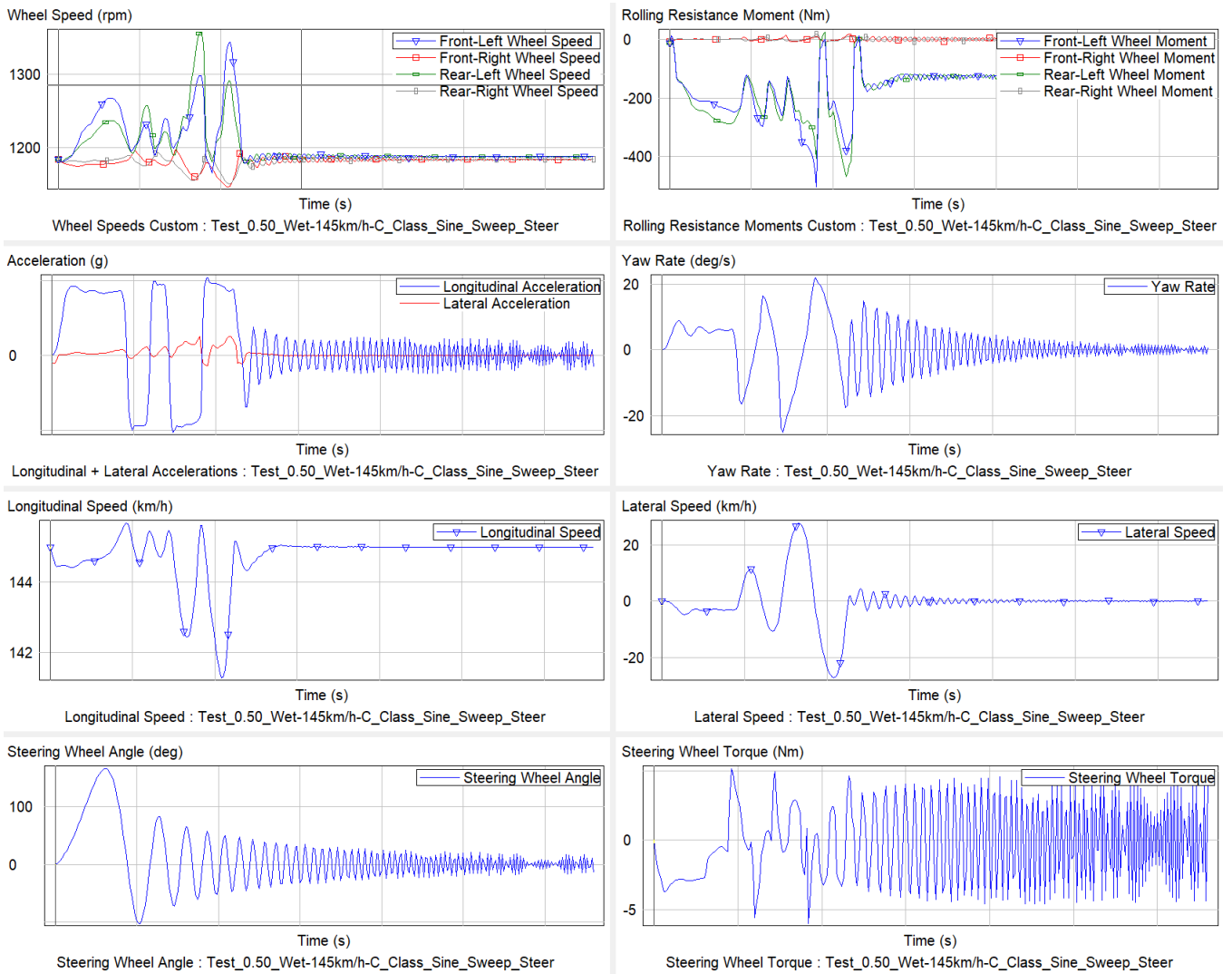


Figure B.8: Measurement Data for 145 km/h Sine Sweep Steer (Wet) Simulation

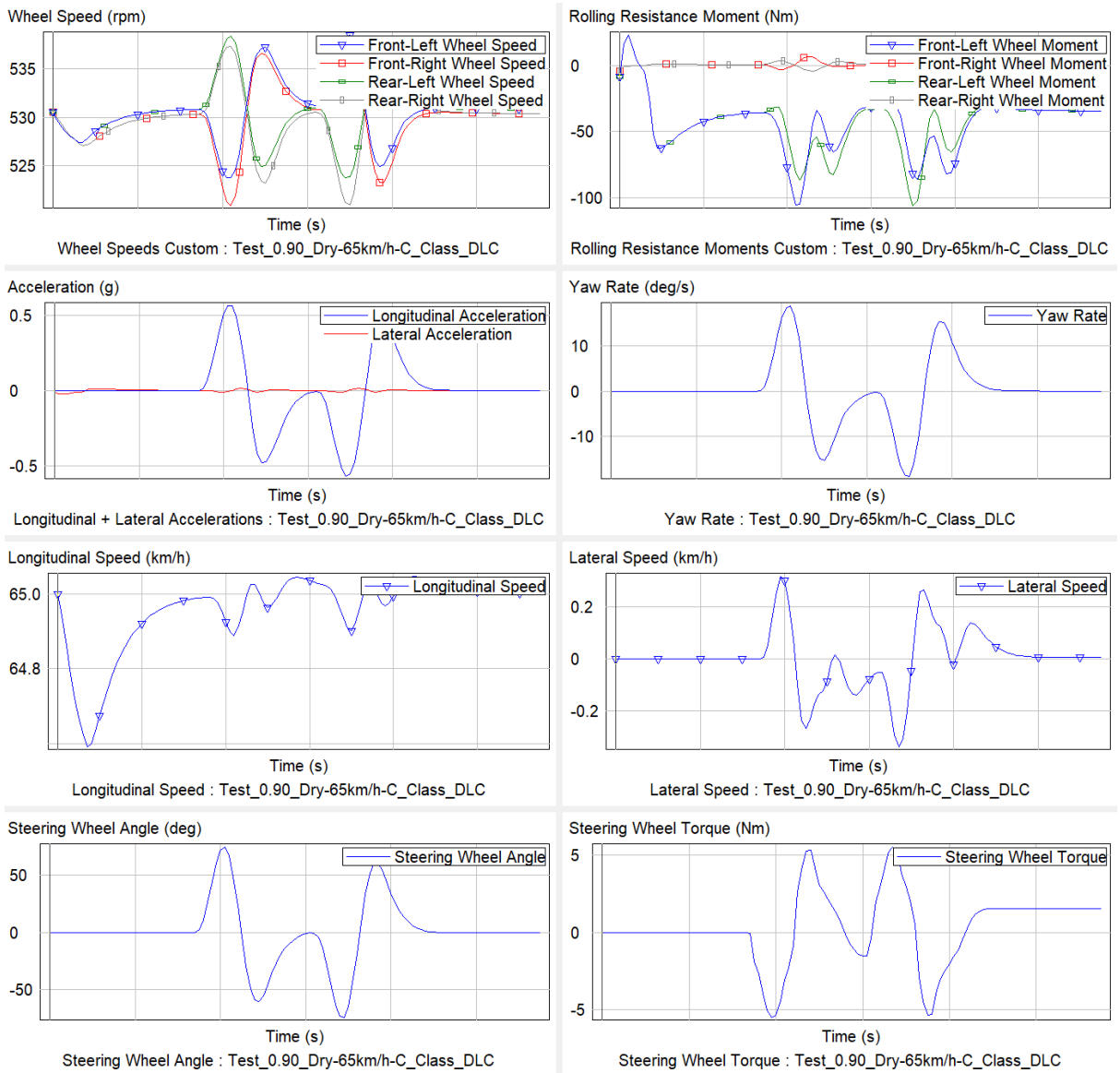


Figure B.9: Measurement Data for 65 km/h Double Lane Change (Dry) Simulation

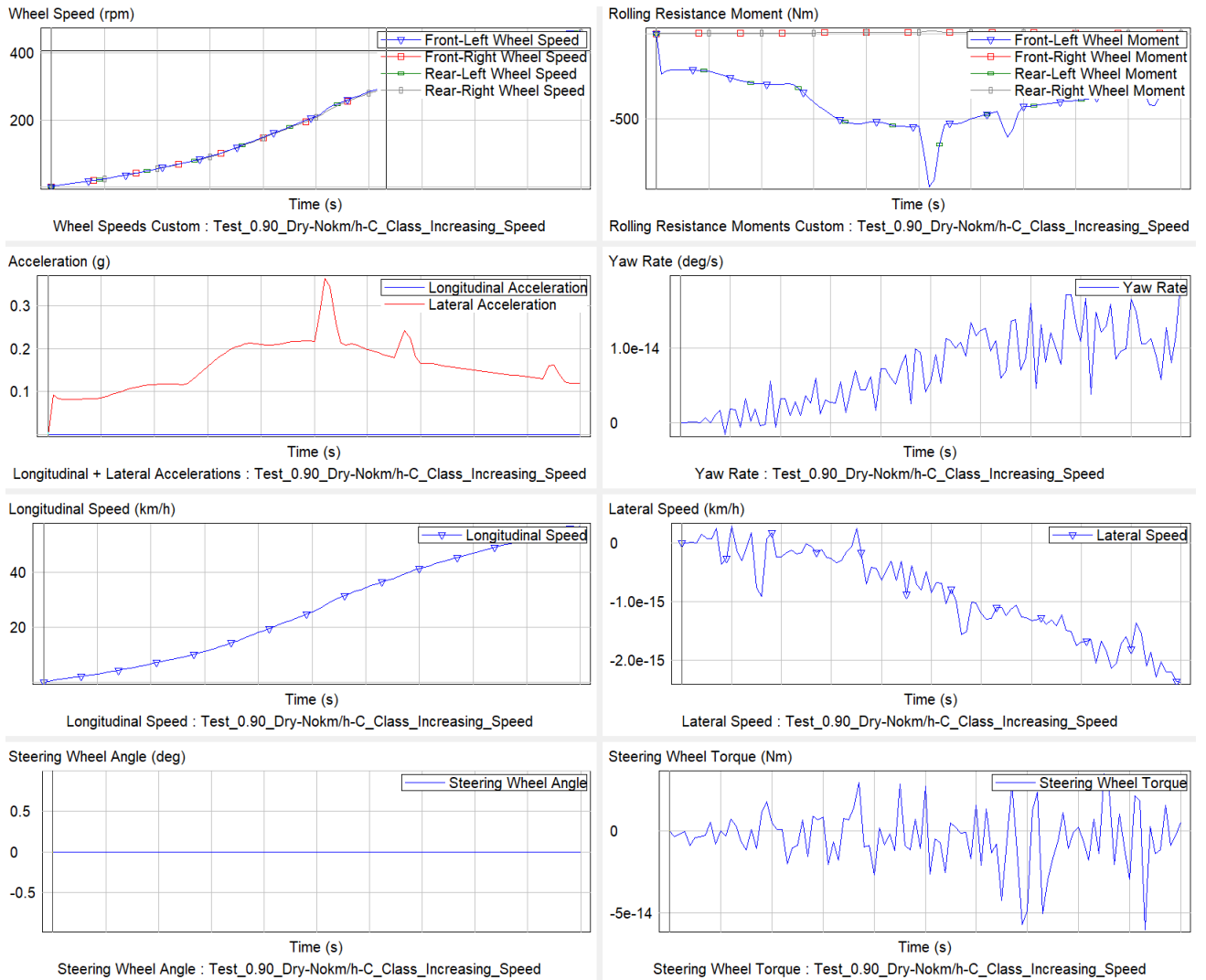


Figure B.10: Measurement Data for Increasing Speed (Dry) Simulation



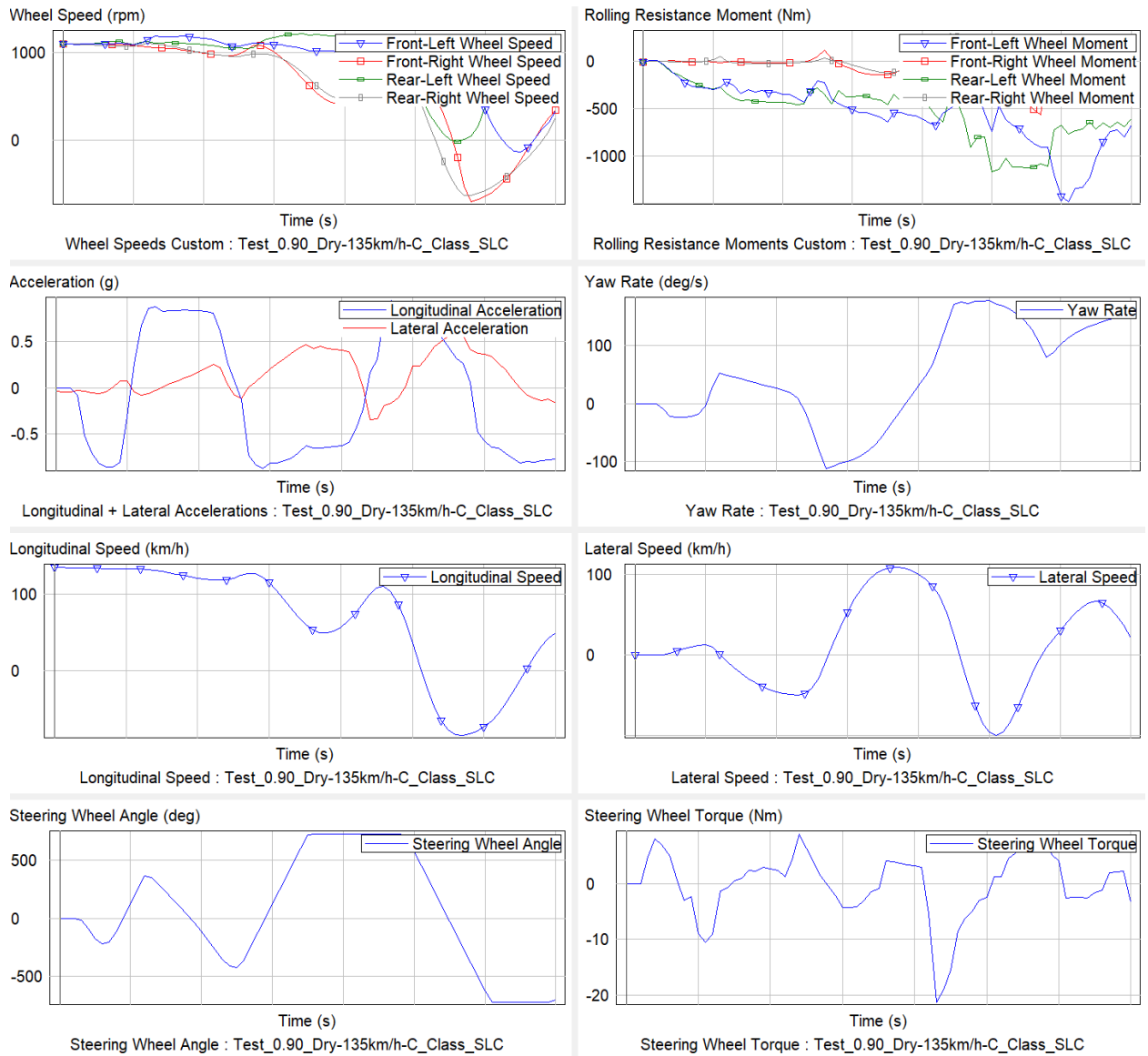


Figure B.11: Measurement Data for 135 km/h Side Drift (Dry) Simulation

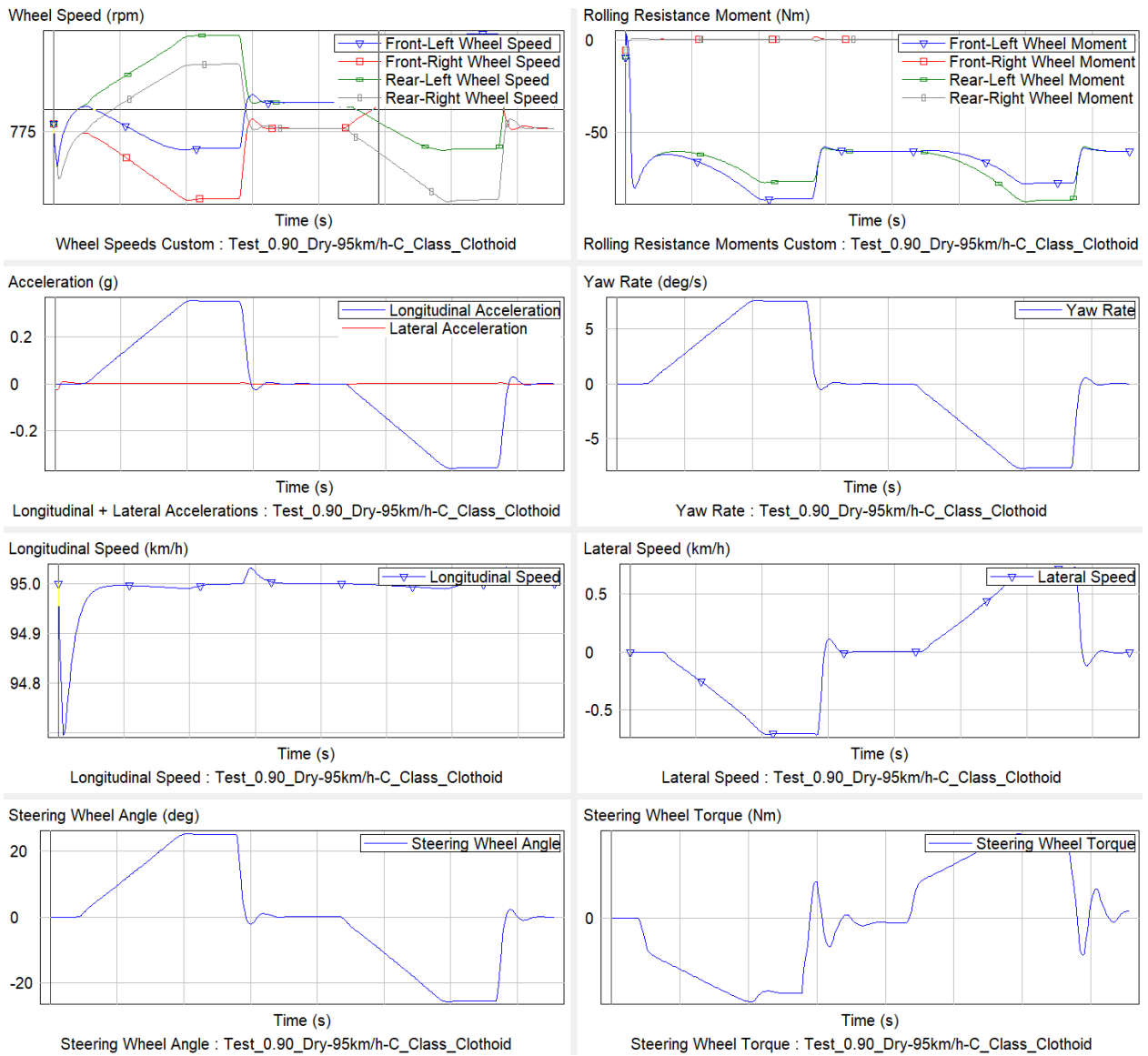


Figure B.12: Measurement Data for 95 km/h S Turn with Clothoid (Dry) Simulation