POLITECNICO DI TORINO

DIMEAS – DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

MASTER OF SCIENCE
DEGREE IN
AUTOMOTIVE ENGINEERING

Master's degree thesis

VEHICLE CONTROL AND TRAJECTORY PREDICTION USING SUPERVISED NEURAL NETWORK

University Supervisor:
Prof. Andrea Tonoli

Candidate:
Arun Prasath Ganesa
Moorthy Ilangovan
Matr.: S260355

Company Supervisor:
Ing. Francesco Ambrogi
(VI-Grade, Italy)

October 2021

Politecnico
di Torino

Acknowledgements

Firstly, I would like to express my sincere thanks to my company supervisor **Ing. Francesco Ambrogi**, Global Service Manager, VI-Grade, Italy, for giving me an excellent opportunity to carry out my thesis work on this topic. His continuous support throughout the project, along with detailed reviews and suggestions, helped a lot in directing me to complete this thesis work.

I would like to express my gratitude and appreciation to my university supervisor **Prof. Andrea Tonoli** of the Department of Mechanical and Aerospace Engineering (DIMEAS) of the Politecnico di Torino for providing the humongous support from the university end throughout this study.

I would also like to sincerely thank **Dr Stefano Feraco** for his assistance and encouragement right from the beginning of this research work. His valuable suggestions and feedbacks helped me to improve the technical aspects of this research work.

Finally, I would like to thank and dedicate this thesis to my parents, sister, and friends, who always gave me moral support and courage to finish this thesis work.

Contents

List of Figures

List of Tables

Abstract

The research and development in the area of Autonomous vehicles have gained huge attention in recent years, intending to transform it to be a safe, reliable and intelligent solution for transportation. The latest advancement of Artificial Intelligence and Machine learning techniques finds its application in the development of Autonomous vehicles to make it a resilient system. Perception, Localization, Planning and Control are the four-module that makes up the Autonomous vehicle system as a whole. The final parts of planning and control are the most important subsystems where the end control decision has to be made.

This thesis work is a small contribution towards the research on Supervised Neural Network based trajectory prediction and control models for autonomous vehicles. In this study, the performance of the Multi layer Perceptron Neural Network model for both the trajectory prediction and control is presented. In the first part of the thesis, the construction of the Neural Network Model for the lateral and longitudinal control is presented. In the second part, the architecture of the Neural Network model for the optimal trajectory prediction is briefed along with the MATLAB algorithms utilized for its construction. These two separate Neural Network models are constructed based on the simulation data of the VI-CarRealTime Driver model. The built-in Neural Network model is tested in the Simulink interface, where the co-simulation with VI-CarRealTime events are performed.

Key Words: Supervised Learning, Neural Network, Deep Learning, Trajectory Prediction, Autonomous vehicle control.

Chapter 1

Introduction

1.1 Thesis Motivation

The continuing evolution of Automotive technologies to deliver safe transportation has seen a huge revolution with the development of Autonomous Vehicles. Focusing on the benefits of a reliable, intelligent solution, self-autonomy and other environmental gains, the research into Autonomous vehicles has gained much interest in the recent past. The Autonomous driving vehicle is nothing but a system that can sense its surrounding environment and takes the required decision and control actions similar to that of the human driver without their intervention with the help of the core software modules embedded in it. The onboard system architecture of the Autonomous vehicle is comprised of three different modules as environment perception, planning, decision-making and vehicle control. The first perception module acquires the details on the surrounding with the help of the onboard sensors like cameras, radars etc., and it utilises this gathered information to form the grid maps to identify the feasible driving areas. The planning and decision-making modules take in the mapping information from above and combine it with the vehicle states details to compute desired target path and the velocity profile. The last control module takes care of the final control actions on the engine, brake and steering to follow the computed path with the desired speed.

Despite the several advancements in the Autonomous vehicle technology, the full autonomy where complete driver independence could be realised is not yet achieved so far. The recent advancement in the field of machine learning and its adoption in autonomous vehicle development has provided numerous solutions to make it a complete, reliable system. The machine learning algorithms can be deployed to solve several tasks in the case of Autonomous cars ranging from object detection, classification, recognition and movement predictions. Both Deep learning and Reinforcement learning which are part of the machine learning techniques are becoming inevitable choices in solving the several complexities involved in the Autonomous cars, as it offers the vast advantages over the current control modules. The concept of deep learning is based on the Artificial Neural Network composed of several layers of neurons which helps to solve the complex linear and non-

linear problems. Based on the learning strategies, the Artificial Neural Network is classified as supervised, semi-supervised or unsupervised networks.

The motivation of the thesis is to explore deep learning strategy especially supervised neural networks, on the application of the trajectory prediction and control module of the Autonomous vehicle system. The main aim of the work is to study the performance of the proposed strategy and to formulate further technical reformation to improve the robustness of the system for the aforementioned application.

1.2 Scope

The construction of Neural Networks, learning methods, offline prediction, online prediction environment setup. Also, it is within the scope of the thesis to study how these parameters affect the performance of the system and to recommend the best possible solution.

1.3 Limitations

As the proposed strategy is based on the simulation of the vehicle model in the closed tracks, this algorithm is limited for the application in the road tracks without any surrounding participants(i.e. not in the urban environments). The trajectory prediction algorithm proposed here to be a global path planning system for a given road track; however, the real-time trajectory prediction could be the future scope of this project.

1.4 State of the Art

1.4.1 Overview of Neural Network based Vehicle Control:

The Control module in the Autonomous vehicle is a critical system as they are responsible for the final action to be taken. Over the recent years, several advancements have been witnessed on Autonomous vehicle control based on the traditional control system controllers.

The recent development in Machine learning and its application in self-driving cars has gained huge interest. Much research has taken place in the recent past to know the feasibility of the Artificial Intelligence based control modules in Autonomous vehicles. End-end vehicle control

using deep learning technology is one of the latest trends in the development of Autonomous Vehicles. The word 'End-end' comes from the end-to-end (E2E) learning strategy, with which the Deep Neural Network (DNN) composed of several layers will be able to solve complex, non-linear problems. The overview of the previous technical works related to this, which are gathered and analysed during the beginning of the thesis work, are described in this section.

The Deep Learning based Autonomous Vehicle Lateral Control have been studied in the simulation environment in the technical paper [1]. In this, the Convolution Neural Network (CNN) is trained with the input of the camera images and its respective steering angle for the different training tracks. With the image segmentation, lane detection from the input images, the CNN model, after training with few different training tracks, has been able to provide lateral control for the unknown road tracks. In continuation of the above work, the addition of longitudinal control is incorporated by developing a separate CNN model for speed prediction as described here [2]. This model also takes in the road images as input; with the minimal dataset, the longitudinal control prediction was found to be satisfactory for the low-speed simulations. The further improvement in this type of NN control model is described in this paper [3]. In this work, in addition to the camera images as input, the parameters capturing the dynamics of the vehicle are provided as input to the CNN model. The training and the prediction of this NN model have performed well compared to the model with only the road images as input. In addition, the ground truth has also improved after feeding the NN model with the previous-continuous vehicle states information for the lateral and longitudinal control. The feasibility of the trajectory control using the neural network model has been explored in the research work [4]. The supervised and reinforcement learning strategies are tested for tracking the given reference trajectory. The simple Feed-Forward Neural Network and Recurrent Neural Network(RNN), which belongs to the supervised learning scheme, has outperformed the traditional geometric path tracker. The training and testing data's are gathered with the controllers like Pure pursuit controller and Stanley controller in this case. Even though Reinforcement learning shows promising performance comparatively, it requires more work to be used in the actual physical systems. Another case study on the trajectory-based lateral control using Reinforcement learning has been briefed in the technical paper [5]. The combined lateral and longitudinal control using the Deep Neural Network (DNN) is reported in the reference work [6]. The Multi-Layer Perceptron (MLP) and Convolution Neural Network(CNN) schemes are tested for the vehicle control application. Compared to the previous studies, here, only the vehicle

dynamics parameters are feed as input to the NN models along with the trajectory information. Both categories of the NN model performed well with the CNN schema predictions appeared better in the tight curves of the tracks.

With the inference from the above research works, the performance of the Multi-Layer Perceptron Neural Network based Lateral and Longitudinal control model is briefed in this thesis work. Rather than using the visual perception (camera images) as input to the NN model, here the vehicle states feedbacks that captures the lateral and longitudinal dynamics along with the reference path information (Path Curvature, Path Distance-Lateral Offset) are feed as input to the NN model during the training and testing. The architecture of the NN model each for Lateral and Longitudinal control is depicted in Figures 1.1 &1.2

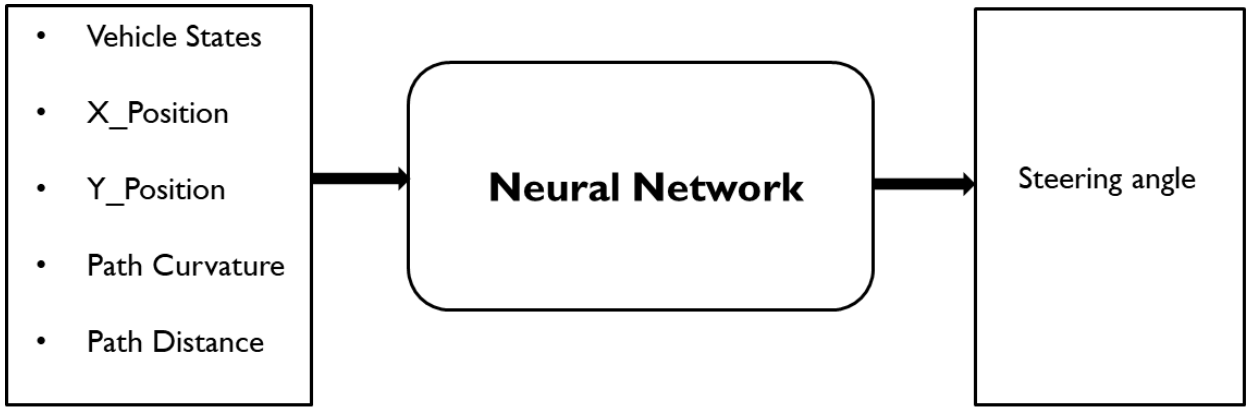


Figure 1.1 Basic layout of NN model - Lateral Control

Vehicle States: Longitudinal velocity, Lateral velocity, Yaw angle, Roll angle, Sideslip angle.

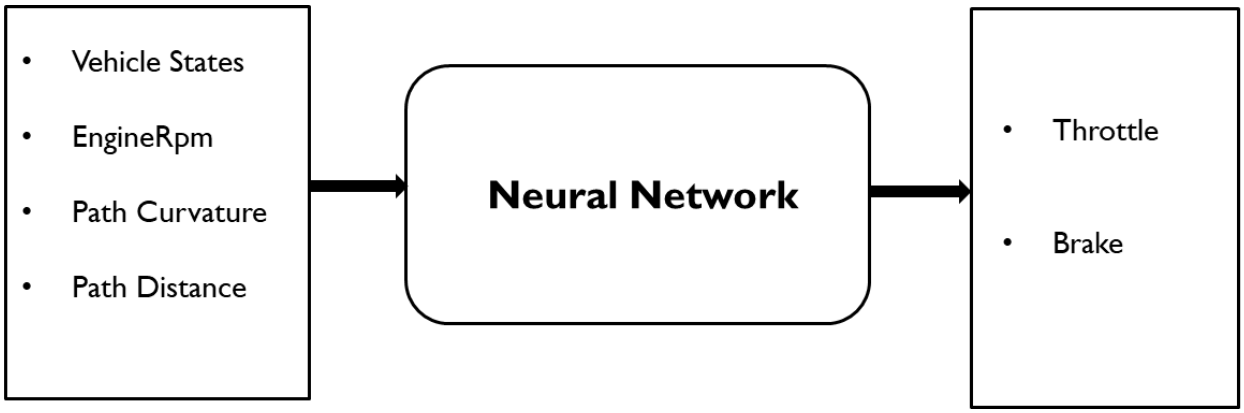


Figure 1.2 Basic layout of NN model - Longitudinal Control

Vehicle States: Longitudinal Acc, Lateral Acc, Pitch angular Acc, Yaw angular Acc, Roll angular Acc.

The NN model architectures are built using the Keras and TensorFlow Python libraries in the Google Colab Python notebook. The simulation data's from the VI-CarRealTime are used for the training and setting up the NN models. All the simulations in VI-CarRealTime are performed using the 14 DOF full vehicle model. At first, the NN model's performance in lateral control for the constant velocity simulations is studied. Then, the NN models are developed separately for the lateral and longitudinal control for the dynamic limit simulation events (Max-performance events). The online testing of the NN model is done in the Simulink environment by performing the co-simulation with VI-CarRealTime events. The built-in NN models are invoked from the Simulink environment through the python interface.

1.4.2 Overview of Neural Network based Trajectory Prediction:

Similar to the control module, Trajectory Planning is also an important sub-system concerning Autonomous vehicles. In addition to providing the collision free-path, this module also holds the significance of various other factors such as performance, handling and comfort. Several studies on the path planning modules based on the A*, RRT algorithms has been done for its application in Autonomous Vehicles. The widespread use of Machine learning and Deep Neural Networks in solving complex problems is being studied in the area of target trajectory prediction and

classification as well. A few of the several works related to this context that are collected during the thesis work are discussed in this section.

When it comes to Machine Learning, all the problems could be formulated either as a classification or a regression one. Likewise, there have been several studies already done on trajectory planning in terms of trajectory prediction and classification. The preliminary study on the trajectory classification using the Deep Neural Network is presented in the paper [7]. In this, the sequential pattern which preserves the order of information of trajectory is chosen to be the feature candidate to make classification among the trajectories. The Convolutional Neural Network and Recurrent Neural Network(LSTM) is deployed for the classification of the GPS based trajectory information to identify the vehicle class as reported in work [8]. Since the GPS trajectory(series of chronological ordered points) cannot be directly fed into the deep neural network models, a representation of the trajectory in terms of vehicle motion features are done as a pre-processing step before training the classification CNN model. However, the proposed classification technique is much less accurate than the classification done by the fixed-point sensors. The application of the Recurrent Neural Network on the future trajectory prediction is briefed in the study [9]. In this, the future position of the surrounding vehicles is predicted using the past observation using Conv-LSTM(Convolutional Long Short Term Memory) Neural Network Model. Also, a feedback scheme is employed to update the position of the target and the surrounding vehicle in the observation sequence after each instant prediction. The study to overcome the problem of identifying the target trajectory based on the machine learning approach is formulated in work [10]. ANN-based system for generating the optimal racing line for the given tracks is explored in this study aimed to reduce the calculation time of several orders compared to the calculation of the target path using the traditional method. This regression task of optimal trajectory line prediction is performed using the feed-forward Neural Network.

Based on the study of the above research works, the Multi-Layer Perceptron Neural Network based optimal trajectory prediction model is presented in this work. The NN model formulated here will predict the optimal trajectory position in terms (X, Y) coordinates by taking the input trajectory details of boundary coordinates and path curvature. Although the model developed here as a global trajectory prediction for a given road track, this could also be utilized as a rapid real-time trajectory

planning in Autonomous Vehicles. The architecture of the NN model for the trajectory prediction is depicted in Figure 1.3.

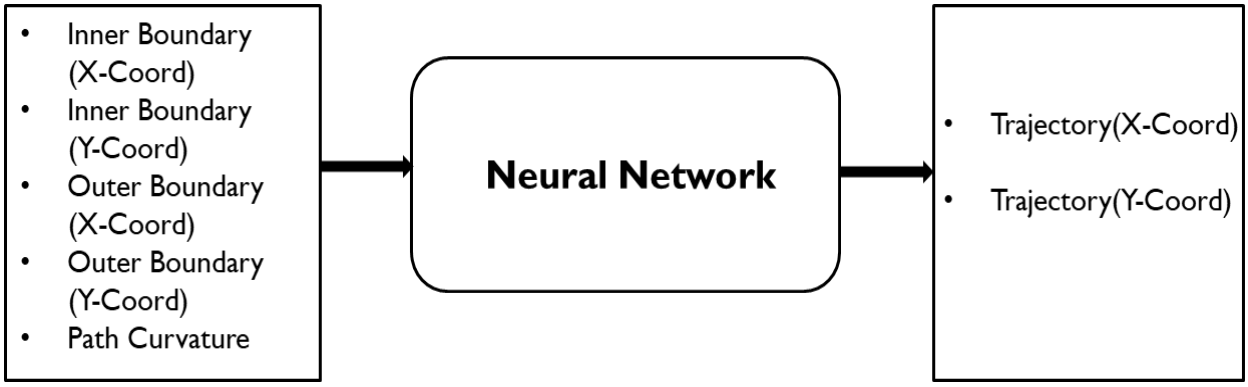


Figure 1.3 Basic layout of NN model - Trajectory prediction

Similar to the above discussed NN model for the control, this model was also built using the Keras and TensorFlow Python libraries in the Google Colab Python notebook. Different road tracks from the VI-CarRealTime database and also from TUMFTM-(2020)(open source racetracks database developed by the TU Munich team) are used for preparing the training and testing datasets. The optimal trajectory here is chosen to be the fastest trajectory(one with the minimal lap-time) which is generated for a given track by the path-builder script(a python script developed by VI-grade to generate different trajectories to the given track centre line based on the corner-cutting value). For the selected road tracks, their corresponding track features are extracted with the help of the MATLAB algorithm. With the information on the optimal trajectory and its respective track parameters, the above NN model architecture is trained to predict the optimal trajectory for any given unseen road track.

1.5 Thesis Outline

The thesis is organized as follows:

- *Chapter 2*: This module presents the general description of the Machine Learning techniques and the type of the learning strategies. Further, the theory behind the ANN in general, model parameters, training parameters and optimization algorithm which are used in the thesis work are also discussed in this chapter

- *Chapter 3*: This module present the description of the VI-CarRealTime vehicle model, the theory behind the VI-Driver model, the Simulation setups utilised in the thesis and the details on the Simulink co-simulation.

- *Chapter 4*: This module presents the methodology involved in the construction of the Neural Network for vehicle control. The Keras and TensorFlow framework, data collection and preparation, training procedure and the online validation setup are also covered in this chapter

- *Chapter 5*: This module presents the methodology involved in the construction of the Neural Network for trajectory prediction. The MATLAB Algorithm for boundary and optimal trajectory point generation along with the data collection, training and validation procedure of the model are also discussed in this chapter.

- *Chapter 6*: This module presents the results of the performance evaluation on two built-in Neural Network models based on different validation procedures.

- *Chapter 7:* This module presents the conclusion and future scope of the thesis work.

Chapter 2

Theory

2.1 Machine Learning

Machine Learning (ML) is the subset of Artificial Intelligence that trains a system how to learn. It is an efficient method of data analysis that aids to build analytical models based on the idea that the system can learn from data, identify patterns and makes a decision with less human intervention. In accordance with the learning strategy, the ML approaches are broadly classified into Supervised, Semi-supervised, Unsupervised and Reinforcement Learning.

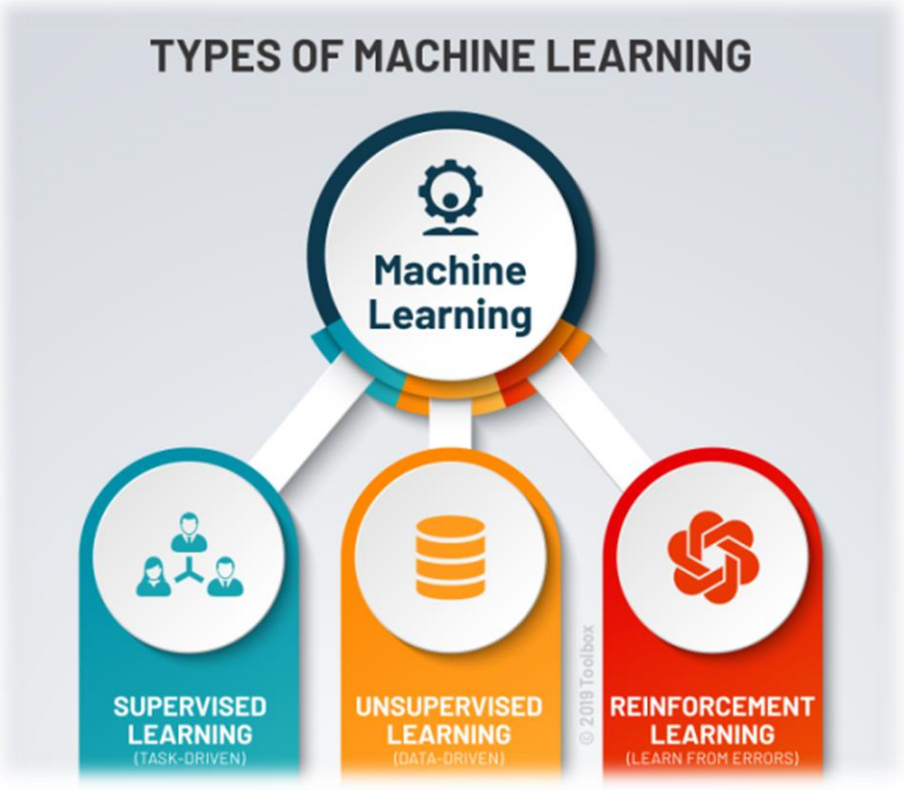


Figure 2.1 Types of Learning Strategies

- **Supervised learning**: In this learning algorithm, the mathematical model is built based on the training data containing both the input and desired outputs. The training dataset could contain one or more inputs and desired output termed as a supervisory signal. This supervised algorithm can be adopted for active learning, classification and regression problems.
- **Unsupervised learning**: In contrary to the supervised algorithm, unsupervised learning models are provided with the training data, which contains only the inputs, with which it finds structure in the data like clustering or grouping. In general, the algorithm learns from the data which has not been labelled or classified by identifying the commonalities in the dataset and reacts to new data based on the presence or absence of such common patterns.
- **Semi-supervised learning**: This type of learning algorithm falls between the supervised and unsupervised categories. In this, only a few parts of the training data will be labelled(with desired output), while the remaining will be unlabelled(without desired output).
- **Reinforcement learning**: This is a branch of machine learning which concerns how agents need to take actions in the environment such that they maximize its cumulative reward. This type of learning strategy is best suited for the disciplines such as control theory, multi-agent systems, genetic algorithms etc.

2.2 Artificial Neural Networks

Artificial Neural Networks, more commonly termed as Neural Networks, are computing systems inspired by the biological neural topology that constitutes the human brain. They were first proposed by McCulloch and Pitts in 1943, in which they proposed a linear combiner with multiple binary input and a single output. With the proper data feeding, this proposed system was able to solve the simple Boolean functions but was not able to learn from the experience. Later the learning scheme was developed, which introduced the first scheme of Neural Network called Perceptron. The Perceptrons are comprised of variable weights, which allowed them to learn from the data.

In general, the structure of a Neural Network is composed of interconnected computation entities called Artificial neurons or nodes in its layers, as shown in figure 2.2.

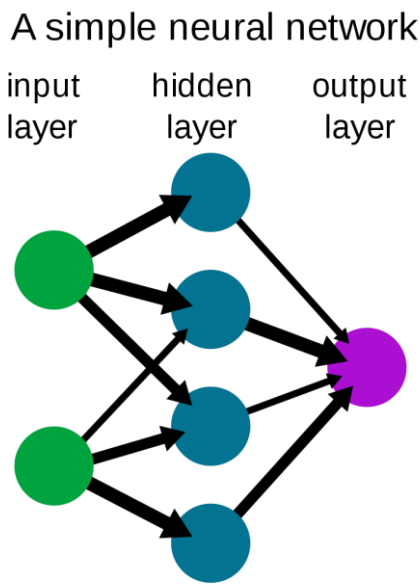


Figure 2.2 Simple network structure

The layers of artificial neurons try to create a mapping function between the input and output, same as that of the brain reaction (output) when it senses something (input). The Neural Networks are a typical application of supervised learning where the input-output mapping is performed by minimising the residual prediction error through forward and backwards pass over several input-output pairs. All the machine learning algorithms are centred around data. The data which are employed to build the network model are categorised as training, validation and testing datasets. Firstly, the training dataset is the one based on which the model will be trained, and hence this should be the largest and cover diverse data to make the trained model robust enough. The validation set is used while training the network to assure that the model generalizes well on the new unseen data. The test data set is utilised for the final evaluation of the neural network once it has finished its training. Its main purpose is to check the performance of the model on the dataset, which has not been used in any part of the training.

2.3 Feed-Forward Neural Networks

The Feedforward neural network (FFNN) is the simplest form of ANN where the connection between the units do not form a cycle. The name feedforward comes from the fact that the

information flows only forward in the network from the input nodes to the output nodes through hidden nodes (if present).

Feedforward neural network is primarily used in supervised learning applications. This network tries to compute the function *f* on the fixed size input *x* such that $f(x) = y$ for the training pairs (x,y). The typical representation of the feedforward network can be given by the formula

$$f(x) = f_1 (f_2 (\ldots f_{n-1} (f_n(x))))$$

where f_i is a layer in the network consisting of *n* layers.

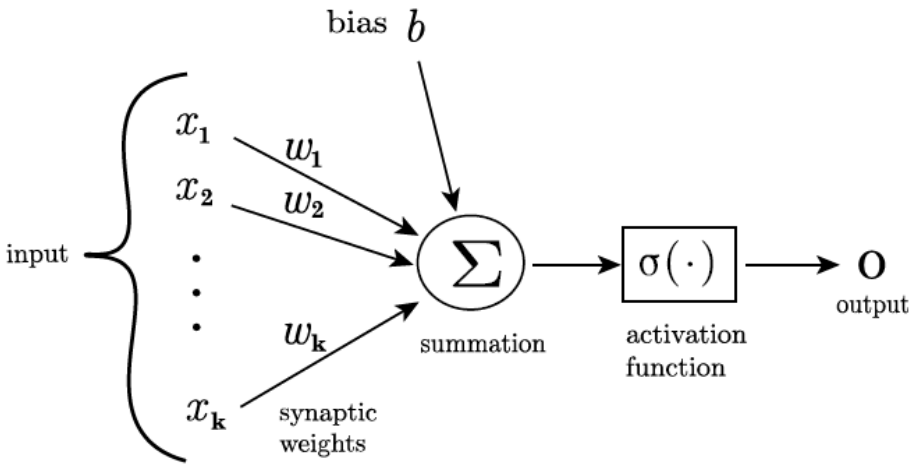


Figure 2.3 Network with its weights and biases

The FFNN with several neurons is shown in figure 2.3. The neurons *j* in the hidden layers gets their input $x_1, x_2, \ldots\ldots, x_k$ from the input layers, which are then multiplied with the respective weights $w_{j1}, w_{j2}, \ldots, w_{jk}$. These weight products are added together with the bias b_j. This summation is then passed through an activation function $\sigma(.)$, which results in the output o_j. The below equation represents the complete math behind the output of each neuron.

$$o_j = \sigma(\Sigma w_{ji} x_{ji} + b_j)$$

The same computation takes place in the neurons in the next layer with this output of the previous layer as its input.

The input layer size depends on the input features decided to be fed to the network, e.g., if the input data is an image, each input will correspond to one pixel. Likewise, the size of the output layer depends on the number of the desired target chosen. Based on the nature of the problem, the type of activation function varies; the usual procedure is to have the same activation function throughout the network except for the output layer, which produces the values in the range of the desired target value *y*. The important feature of the activation function is to add non-linearity into the linear model.

The performance of the network hugely depends on its architecture, such as the number of layers, size of layers and hyperparameters. However, there is no possible way available now to know beforehand about the effectiveness of the chosen architecture. Currently, performance is usually evaluated by trial and error of different architectural settings, which is generally a time-consuming process.

2.4 Activation Function

As stated in the above section, the activation function takes the input value from the nodes and maps it into a specific range based on the chosen function. The output from this is then fed as input to the next nodes in the network. There are several activation functions available to be used for training the Neural Network; the most common functions are discussed below.

2.4.1. ReLU Activation Function

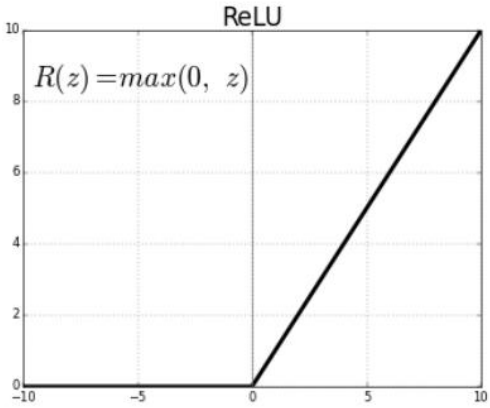


Figure 2.4 ReLu Activation function range

The Rectified Linear Unit, most often termed as ReLU, is the most used function in the present generation in most deep learning applications.

The range of the ReLU activation function is 0 to infinity. It is half rectified (from the bottom), as shown in the figure. f(z) is zero when z is negative, and it is equal to z when z is greater than or equal to zero. The major disadvantage with ReLU is it cannot handle negative value, meaning with the negative value given, the function returns zero, which in turn affects mapping of the value which are less than zero.

2.4.2 Sigmoid Activation Function

As the range of the sigmoid function is 0 to 1, it is predominantly used in the models where it needs to predict the probability as the output. The shape of the function looks like an S-shape

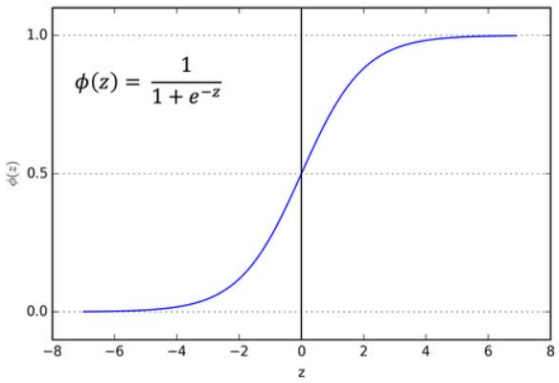


Figure 2.5 Sigmoid Activation function range

The sigmoid function is also one of the most widely used activation functions, especially for the classification problem. The major drawback of this function is the vanishing gradient problem; when the function reaches its horizontal part of the curve, the gradient will be very small, and the network will struggle to learn.

2.4.3 ELU Activation Function

It is the extension of the Rectified Linear Unit function to operate with the negative values with the function as $f(x) = a(e^x-1)$ when x is less than zero.

2.5 Loss Function

The loss functions are key components in the ML algorithms which relates to model accuracy. The main purpose of the loss function is to evaluate how well the model is trained for the given dataset. Based on the value of the loss during the network training, the model's prediction accuracy can be determined (i.e.). The loss value will be very less when the predictions are good, and it will be of larger value when the predictions are totally off.

Depending on the type of problem that needs to be handled, the loss function needs to be chosen. The most common loss function for the regression problems is Mean Squared Error (MSE) which is described below. In this thesis work, all the NN models are evaluated based on the MSE loss function.

The MSE works by computing the sum of the squared distance between the predicted value and the expected value. The MSE function works well but is heavily affected by the outliers, and this has to do with the square of the error. So when the error grows larger than 1, then MSE will grow exponentially.

2.6 Optimization Algorithm

During the backpropagation, while training the network based on the optimization algorithm, the weights are updated to minimise the loss function J(Θ), where Θ is the model parameters. Gradient descent is the most commonly used optimization algorithm in deep learning model construction. To reduce the loss, during each training step, the parameters are updated in the opposite way of the gradient of the loss function. The learning rate has to be multiplied by the gradient to know how much weight needs to be updated during each step.

Among the variants of gradient descents, batch gradient descent is more effective as it computes the gradients for the batch of training samples. Thus only a small subset of the training data needs to be in the memory at a time, and hence the fluctuations are reduced. The gradient descent is often time-consuming to identify the correct learning rate. The learning rate should neither be so high or very small, thus finding the optimal point is hard and time-consuming. During this update process, there is a possibility to end up in a sub-optimal state where it might be hard to get out gradient

descent. To overcome these disadvantages, other optimizers have been developed; among them, the Adam is the most widely used optimizer, and it is also be used in all the NN models presented in this thesis work.

2.6.1 Adam Optimization

Identifying the good value of the learning rate is very important to have accurate training results. The Adam optimization algorithm computes adaptive learning rates, which are preferable for deep and complex networks.

The intuition behind adapting the learning rate to the individual parameter is that the history of the parameter with the small magnitude gradient should be updated with more impact than those with the larger ones so that the training process will progress more evenly along all the parameters axes. The Adam, which is an acronym for adaptive moment estimation, make use of the past gradients and also its squares, much like momentum.

Adam Configuration Parameters [11]

- Alpha: Referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
- beta1: The exponential decay rate for the first moment estimates (e.g. 0.9).
- beta2: The exponential decay rate for the second-moment estimates (e.g. 0.999).
- epsilon: Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8)

Specifically, the algorithm calculates an exponential moving average of the gradient and its squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

The initial value of the moving averages and beta1 and beta2 values close to 1.0 (recommended) result in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates.

2.7 Network

In section 2.3, the theory behind the general feed-forward neural network is described; however, there are several variants of neural network topologies available at present which could be employed depending upon the application. Multilayer Perceptron(MLP) topologies have been used in this work as it is the best fit for the regression problems.

2.7.1 Multilayer Perceptron

Multilayer Perceptron is the class of feed-forward neural network often applied to supervised learning problems. They train on a set of input-output pairs and learn to model the correlation (or dependencies) between those inputs and outputs. Training involves adjusting the parameters or the weights and biases of the model to minimize error. Backpropagation is used to make those weight, and bias adjustments relative to the error, and the error itself can be measured in a variety of ways, including by root mean squared error (MSE).

In the forward pass, the signal flow moves from the input layer through the hidden layers to the output layer, and the decision of the output layer is measured against the ground truth labels.

In the backward pass, using backpropagation and the chain rule of calculus, partial derivatives of the error function concerning the various weights and biases are back-propagated through the MLP. That act of differentiation gives us a gradient, or a landscape of error, along which the parameters may be adjusted as they move the MLP one step closer to the error minimum. This can be done with any gradient-based optimisation algorithm such as Adam optimizer. The network

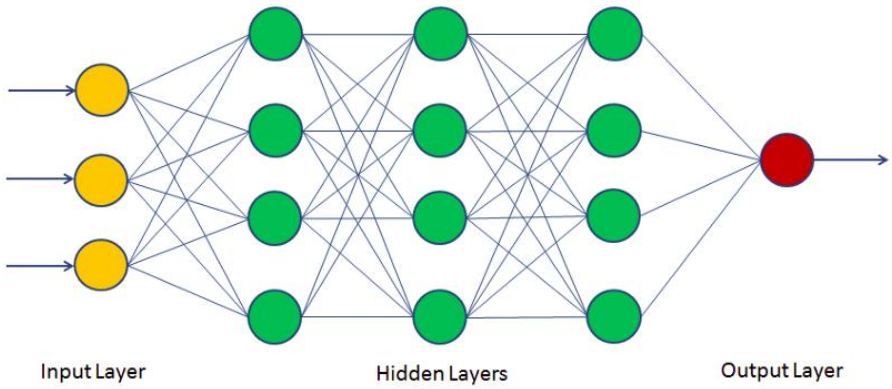


Figure 2.6 Network Architecture of MLP

keeps playing that game of tennis until the error can go no lower. This state is known as convergence.

2.7.2 Training a Network

The training process of a supervised neural network starts by initializing the parameters to a small random value. These parameters are then updated to reduce the difference in distance between the output of the network and ground truth, thus optimizing the problem. This distance is computed based on the L2-norm as the squared distance between the output and the ground truth values. This distance is formulated as scalar-valued error or loss, denoted by *L,* which need to be optimized with the gradient descent optimization as briefed in section 2.6.

The learning rate is set to scale the training update; however, too large a learning rate value may cause the loss to diverge, and too small leads to very slow convergence [12]. This can be regularly updated by calculating the gradient of the loss with respect to all the parameters. To compute this, the gradient of all the parameters between the last network layer and the specific parameters of interest have to be calculated as they are dependent on each other. By using the chain rule of calculus, this is done by results in error being propagated backwards through the network, and this process is usually termed as backpropagation.

This process of backpropagation will be repeated many times over the training dataset while training the supervised neural network. As per the theory, the network first passes the data forward in the network and then performs the backpropagation and updates the parameter. In every update step, the network tried to reduce the loss between the output and ground truth, eventually reaches a convergence at one point to satisfy the optimal performance.

2.7.3 Network Regularisation

The main objective of building a Neural Network is to have a final model that performs well on both the training data and also on the new data on which model should be able to make a good prediction. The degree to which the model is regularised measures its prediction accuracy on the new unseen data not been used while training the network. This makes the necessity to improve the training of the model, which may in turn results in too much learning with respect to the training

data set, resulting in an overfitting problem. On the contrary, too little learning during the training results in an underfitting model. In either of the cases, the model is not generalised enough to predict the new data. Based on this phenomenon, the model could be grouped as following [13]

- **Underfit Model**: A model that fails to sufficiently learn the problem and performs poorly on a training dataset, and does not perform well on an unseen data sample.
- **Overfit Model**: A model that learns the training dataset too well, performing well on the training dataset but does not perform well on a new data sample.
- **Good Fit Model**: A model that suitably learns the training dataset and generalizes well to the holdout dataset.

An underfitting phenomenon could be addressed by increasing the capacity of the network (i.e.) increasing the complexity of the model by changing its structure, such as adding more nodes or layers so that the good mapping of input to output takes place.

Mostly while training the network, it is more prone to reach the overfitting phenomenon. It is easily diagnosed by monitoring the training performance of the model on the training and validation dataset, especially by observing the learning curve, which shows an overfitting pattern. There are several strategies available to reduce the overfitting problem, below are the two techniques utilized in this work while training the NN model.

Dropout

One of the most widely used procedures in most of the NN architecture is to overcome the overfitting model on training data. Depending on the dropout factor, this layer will deactivate the neurons during the forward pass making the data bypass these nodes, reducing the capacity of the network by a fraction. This technique will randomly turn off neurons during each iteration forcing the model to regularize well.

Early Stopping

This technique prevents the network from overfitting by stopping the training process once the loss related to the validation set starts to increase. The patience level can be set to monitor the validation loss during the training; once the loss keeps increasing till the threshold iterations, the training will be terminated to prevent the model from overfitting.

Chapter 3

VI-CarRealTime

VI-CarRealTime is a virtual modelling and simulation environment targeted to a simplified four-wheeled vehicle model. In this chapter, the vehicle model used in this thesis work will be briefed along with a short description of the tool and set-up which has been utilized in the course of this project.

3.1 Vehicle Model

The Mathematical vehicle model is a multi-body system that could be well suited to study dynamic behaviours. The vehicle model, in general, is a complex system composed of various sub-systems like vehicle body, wheel and tires, steering, suspension, power train, brakes. In this thesis work, we used the VI-CarRealTime vehicle model for both data collection and validation purposes. Few of the important vehicle model details [14] are briefed in the following subtopics.

3.1.1 Coordinate System

The reference coordinates used in the VI-CRT vehicle model, as shown in figure 3.1, is in accordance with the standards:

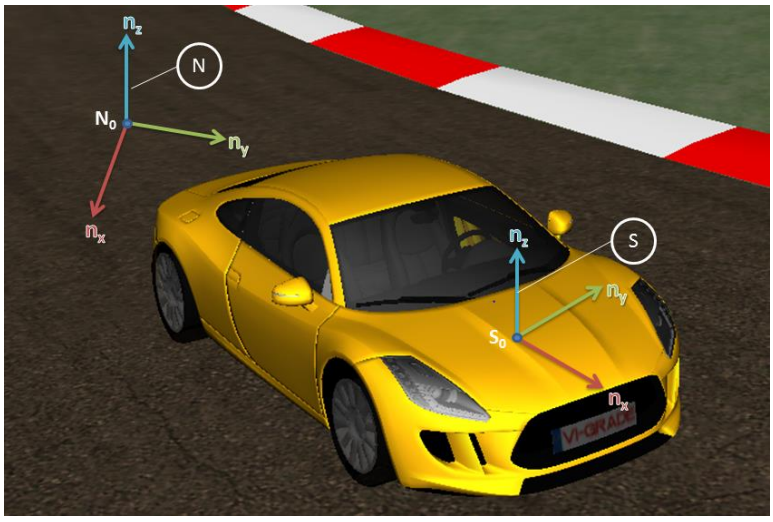


Figure 3.1 Reference coordinate system

ISO 8855, Road Vehicles- Vehicle dynamics and road-holding ability- Vocabulary.1991 and SAE Recommended Practice J670f, Vehicle Dynamics Terminology.

Global Reference Frame

The root body of the tree representing the rigid-body system is the global frame. The body is a Newtonian or inertial frame, which means that it does not accelerate in translation and does not rotate.

The global inertial frame of reference, N (N denotes Newtonian), for VI-CarRealTime models has unit vectors n_x, n_y, and n_z, and origin point N_0 as shown in the picture below. In design condition, Global Reference Frame coincides with Vehicle Reference System.

Vehicle Reference Frame

The vehicle model is positioned concerning the global frame. The origin of the Vehicle Reference System S0 is located at Z=0 of Global Reference Frame and half front vehicle track.

Gyro Reference Frame

The gyro reference system (gyro marker) is located on the body coinciding to sensor point and is oriented with the Z-axis coincident with global Z during the simulation with the X-axis along the vehicle movement direction, so it follows the vehicle displacement, follows the yaw but not the pitch of the vehicle.

3.1.2 VI-CRT Vehicle Model

In this thesis work, the goal is to build a Neural Network vehicle controller which mimics the behaviour of the exiting control system based VI-driver model with respect to the lateral and longitudinal dynamics. For this purpose, a simplified vehicle model with 14 degrees of freedom is used, which includes five rigid parts such as vehicle chassis(sprung mass) and four-wheel parts unsprung mass, as depicted in figure 3.2.

The 14 DOF includes 6 DOFs of vehicle chassis, while wheel parts have 2 DOFs (each for describing the motion concerning the vehicle body and for the wheel spin). The main assumption are suspension does not have linkage or bushings, and the steering system does not have the part

for the steering wheel or rack. The kinematic and compliance data of the steering and suspension are described by the look-up tables using the conceptual approach.

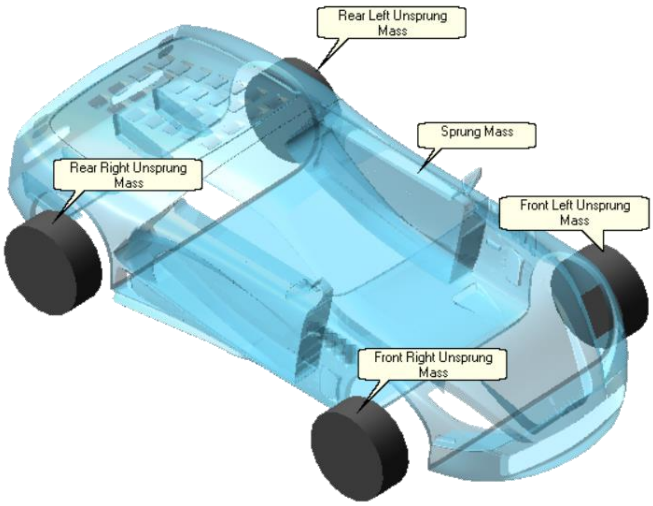


Figure 3.2 Vehicle model Sprung and Unsprung mass

The Vehicle model intends to accurately predict overall vehicle behaviour for cornering, braking, and acceleration-performance studies for four-wheeled vehicles with independent-front and independent-rear suspensions. The simplified model is described in terms of commands and functions that use an internal development environment working as a symbolic manipulator tailored for deriving multibody equations and a code generator.

3.1.3 VI-Driver Theory

VI-Driver is a control system designed to drive the vehicle model simply and efficiently to fulfilling the requirements such as being capable of driving on both limit and sub-limit

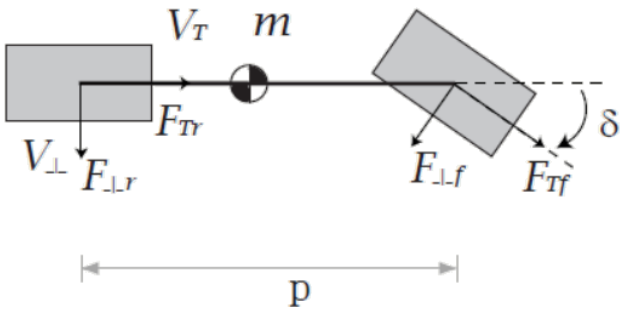


Figure 3.3 Classical bicycle model

manoeuvres, simple to tune and self-adapting and robust enough to adopt a wide range of vehicle characteristics.

This system is based on the model-based predictive control for which the classical bicycle model is used as described below. The lateral controller uses a model base predictive control technique, calculating at each instant in time the required control action.

For the vehicle model shown above, given a target curve that represents the trajectory that must be followed, it is possible to define a connecting contour (compliant to the differential flatness principle) obtained by resolving the equation below (assuming a proper order polynomial is used for interpolation):

The vehicle moving frame is used as reference (Frenet Frame), and the connecting contour constraints taken into account are:

1. contour initial position (compatible with the vehicle position)

2. contour initial orientation (compatible with the vehicle speed and heading)

3. contour final position (the reference driver line evaluated at the preview distance)

4. contour final orientation (smoothly joined with the reference driver line)

Those four constraints require at least a polynomial of order four: a cubic polynomial has been chosen, and its coefficients calculated using the relations shown in the picture above.

Using the differential flatness property, the connecting contour is used as a dynamic vehicle trajectory to be applied as a control action. The same property allows inverting that trajectory and calculating the appropriate steering control action.

This procedure is repeated at each instant in time, following the Model Predictive Control Paradigm. Given the vehicle speed v, the side slip angle, the preview time pt the preview distance D (computed as v*pt), the principle used can offer a very good approximation of the required steering angle necessary to bring back the vehicle to the target path, when a tracking error occurs.

A final stage has been implemented, which compensates for the unmodeled lateral dynamics. The compensation, called "yaw rate controller", uses the reference path curvature and the actual vehicle

yaw rate and corrects the steering action with the yaw rate error to bring the instantaneous vehicle curvature as close as possible to the reference path curvature

As for longitudinal dynamics, a model-based feedforward /feedback scheme is used to compute the needed torques to accelerate or brake the vehicle based on the target speed profile coming from the stationary prediction.

Because of unmodeled dynamics, disturbances and numerical integration noise, the simulated vehicle almost always diverges from the global optimal trajectory target. The combination of the control actions takes the vehicle back on it in some sub-optimal fashion: the motion control acts on the steering, the throttle and/or the brake to reduce the longitudinal speed tracking error to acceptable limits. The controller is not acting in a combined way though, the longitudinal and the lateral controller algorithms are separated: any longitudinal controller action would certainly influence the lateral dynamics of the car, especially at the limit.

Nevertheless, the simplicity and the natural robustness of the lateral controller invokes correct actions, and it has been observed to behave very realistically in all the cases where the target trajectory is not properly followed due to excessive side slip. Those cases include limit over and understeer situations, which typically occur to a real driver when trying to reach the limit vehicle performance.

3.2 Simulation Events

The build mode in the VI-CRT interface allows the creation of the vehicle model that need to be used in the simulation and other changes required to be introduced into any of the vehicle subsystems. The vehicle model can be analysed by the virtual simulation based on the setup of the driving event in the test mode ranging from simple manoeuvres to the more complex customised events. The simulation events used in this work to gather the data values are File driven events and the Max Performance events.

3.2.1 File Driven Event

In the first part of the vehicle control Neural Network, the constant velocity model for lateral control is built and studied, for which the file-driven simulation event data have been used. The file driven events simulate the vehicle model based on the VDF file, which typically

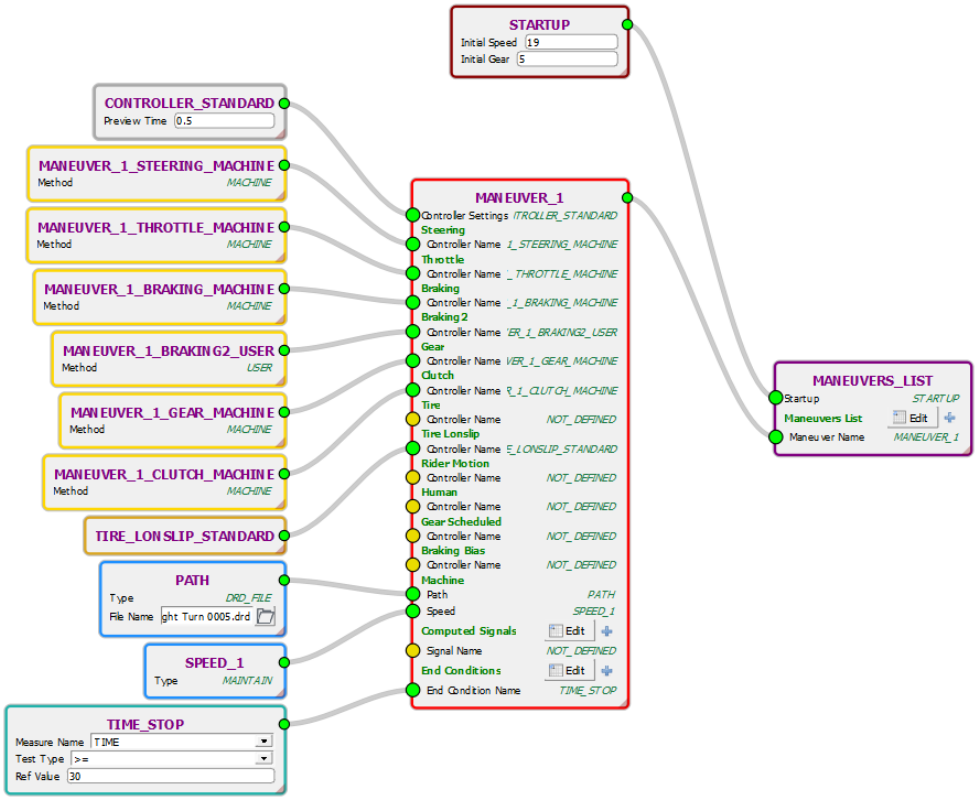


Figure 3.4 VDF file settings

store driver controls tables of databases. These events are set up and modified with the help of the Event builder interface. The Event builder space for the constant velocity event for the sample road track is shown in figure 3.4

All the VI-Driver controller settings and the manoeuvre set-up could be configured in this space. In addition, the reference path of the road track selected for the simulation event could be fed in the Path tool box as a DRD file which will be interpreted as target trajectory by the driver controller. This reference path input provides the path related output channels during the

simulation, such as path curvature and path distance (lateral offset between the actual vehicle point to the reference). Apart from the driver control setting, the road track in which the simulation has to run is given as input with the help of a road data file which could be created and modified with the help of the VI-Road interface.

3.2.1 Max Performance Event

Once the constant velocity model for the lateral control is studied, Neural Network architectures are reconfigured to suit the dynamic events based on the Max Performance simulation setup in VI-CarRealTime.

Max Performance event is used to define dynamic limit velocity profile on a given track. The event uses a specific static solver (VI-SpeedGen) to compute a velocity profile, and then it uses the dynamic solver to verify if the computed speed profile is feasible. If the velocity path is unfeasible (maximum path distance exceeded, zero velocity etc.), the solver turns back to the static and computes a new velocity profile, modifying only the portion nearby the unfeasible point. The process is iterated until a feasible velocity profile is found.

The vehicle model used for static prediction has no suspensions and inherits all properties from the full model. The effect of aero forces is considered, and the effect of suspension jounce is taken into account by the presence of ride height maps.

3.3 MATLAB/Simulink Co-Simulation

The VI-CarRealTime environment can interact with the MATLAB/Simulink tool using a specific interface. The vehicle model can be connected to MATLAB so that all the MATLAB tools can be used to modify the model complexity or add any additional control modules which are developed in MATLAB/Simulink or external systems. This interface aids to run the co-simulation to analyse the test result.

The VI-CRT model is imported to the Simulink as an S-function representing the vehicle plant, as shown below in figure 3.5.

Figure 3.5 Passive Vehicle S fucntion - Simulink

This S-function receives the details of the vehicle parameter along with the simulation event details through the XML file (generated when the simulation is run in VI-CRT environment), which is feed as an input file to the Simulink solver.

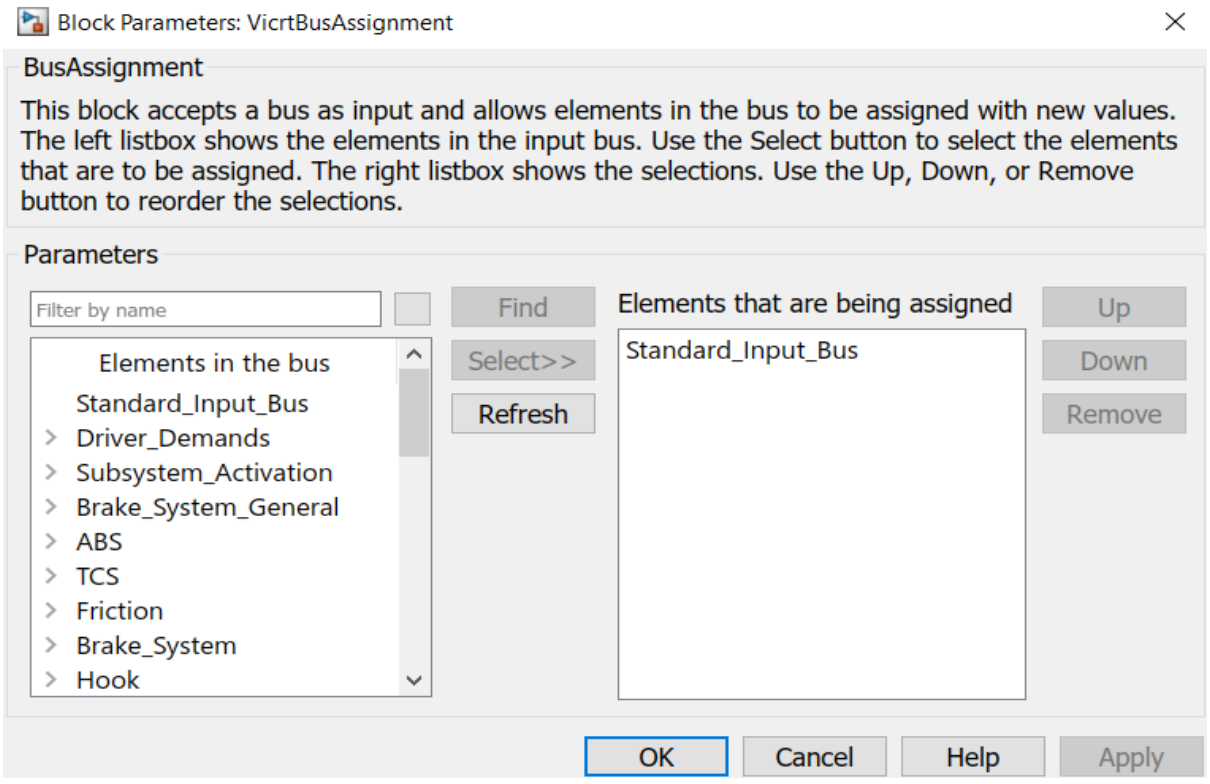


Figure 3.6 Input and output channel selection pan

The input and output ports of the vehicle model during co-simulation can be chosen from lists of available channels as shown above in figure 3.6, so the resulting overall interface is a unique Simulink block that can be connected to other blocks through its ports.

To start a co-simulation, the standard approach on the VI-CarRealTime platform need to be followed to run a standalone simulation, setting the vehicle model and the simulation event. After configuring the desire solver settings (and providing the XML file as input), MATLAB S-function communicates with the defined VI-CarRealTime event through the co-simulation from the Simulink environment.

Chapter 4

Methodology - Vehicle Control

4.1 Overview

In this chapter, the description of the tools, methods and workflow used during this thesis work concerning the NN model for vehicle control is explained. In general, the Neural Network model for vehicle control has been implemented iteratively, starting with the simpler system and has been extended further.

As a first step, the Neural Network model, which predicts the lateral control for the low-speed constant velocity event, is created. After evaluating its performance and the selection of the input features of the network along with its mapping tendency to the output(steering angle in this case), this simple model is then extended. With increasing complexity, the lateral and longitudinal control prediction Neural Network model is then built for the dynamic limit conditions. Both the networks are set up based on the offline training strategy. The term offline training refers to training the Neural Network with the dataset, which is gathered and prepared in advance. With this dataset, the model is created by optimizing the training strategies and network hyperparameter. This method is contrary to the online training where the NN is trained by the real-time data generated from the simulator environment (for instance, through client-server connections). To evaluate the performance of the models created, both the offline and online prediction methodology is followed. The online validation setup is done with the VI-CarRealTime environment in the Simulink, where the NN model created is loaded and called from the python interface. The layout of this validation environment is discussed in the final section of this chapter.

4.2 NN-framework

The Keras and TensorFlow libraries are commonly used to train and implement the development of the Neural Network model using the supervised learning method. Compared to other deep learning frameworks, the Keras/TensorFlow frameworks are most widely used for deep neural networks. A brief description of the Keras and TensorFlow is provided in the below section.

4.2.1 TensorFlow

TensorFlow is one of the top numerical platforms that provide the basis for deep learning research and development. It is a Python library for fast numerical computations, which was created and maintained by Google and released under the Apache 2.0 open-source [15]. It is a foundation library that can be used across a range of tasks but has a particular focus on training and inference of deep neural networks directly or by using wrapper libraries that simplify the process built on top of TensorFlow. The API is nominally for the Python programming language, although there is access to the underlying C++ API.

4.2.2 Keras

Keras is a minimalist Python library for deep learning that can run on top of TensorFlow [15]. Though TensorFlow suits the best for numerical computations, it can be difficult to use directly creating the deep learning models. Hence Keras is developed to make deep learning models as fast and easy as possible for research and development. It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs, and CPUs were given the underlying frameworks.

In this project work, all the network architectures are predominantly created with the help of the Keras library on the top of TensorFlow. The main focus of Keras is the idea of a model. The model here denotes the sequence of layers, the main type utilised in this work is the Sequential model, which is a linear stack layer. Once the Sequential layer is initialised, depending upon the non-linearity of the problem, the additional layer stacks could be added by specifying the number of neurons for each layer in the network. After the layers definition, the model needs to be compiled so that it makes use of the underlying framework to optimize the computation to be performed. During this compile part, the loss function and optimizer, along with its performance parameters,

should be specified. Once compiled, the model must be fit to the trained dataset. This can be done by dividing the dataset in batches or as a whole by mentioning the number of epochs. The epochs refer to the number of times the model will be trained to fit the dataset; for instance, if we give batch size as 50 and epoch as 50 when we try to fit the model for a given dataset, it will split the entire dataset in a batch of 50, and one epoch here refers to one cycle when all the training data (all the batches of 50) has been passed through the model. Once it trains the model for the given number of epochs, the model predictions ability could be assessed using. The brief layout of deep learning model construction in Keras is shown below.

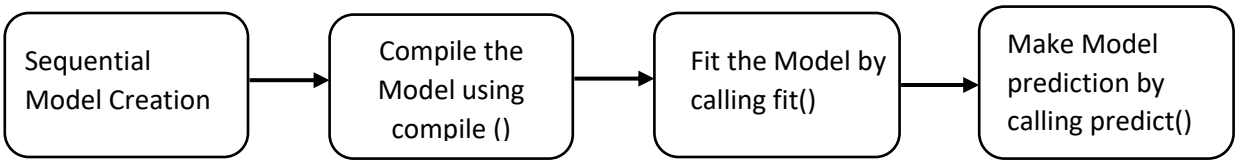


Figure 4.1 Steps in Keras model creation

The TensorFlow/ Keras definition and model creation procedure as described above is done in the Google Colab python notebook. It is an open-source Google's interactive environment that allows to write and execute the Python codes in the browser with all the configurations in-built, which eliminate the requirement of configuring the necessary python packages and libraries.

4.2 System Setup

Table 1 System Configuration

Environment	
Python Distribution	3.8.8
Anaconda Distribution	4.10.0
Machine Learning Tools	
TensorFlow	2.3.0
Keras	2.4.3
Pandas	1.1.3
NumPy	1.20.1
h5py	2.10.0

4.3 Dataset Preparation

4.3.1 Data Collection

Primarily all the data required for the Neural Network model construction is extracted from the VI-CarRealTime simulations. As described in section 3.2, the simulation events are configured in the VI-CRT event builder space for the constant velocity condition, and longitudinal speed is kept a 10 m/s. As our main goal is to develop a NN model whose behaviour is a replica of the VI Driver controller, a wide range of different driving conditions should be included in the training data set. Due to this reason starting for the simple manoeuvres like straight, single turns, multiple turns, more complex road tracks are selected on which the constant velocity simulations are performed. Around 15 road tracks simulation data are used, which includes few tracks from the VI-CarRealTime road database and the rest of the tracks are created in the VI-Road interface based on the centreline coordinates from the TUMFTM – race track database (an open-source GitHub repository created and maintained by TUM – Institute of Automotive Technology) [16]. A few of the road tracks used for the training data preparation is shown below.

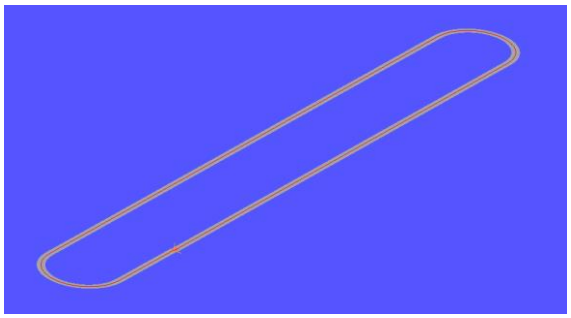


Figure 4.2.1 Track : VI-Road Example

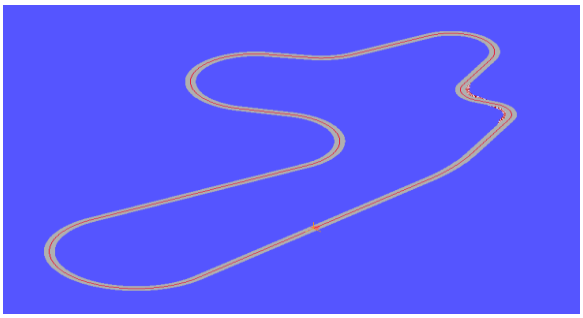


Figure 4.2.2 Track : VI-Track

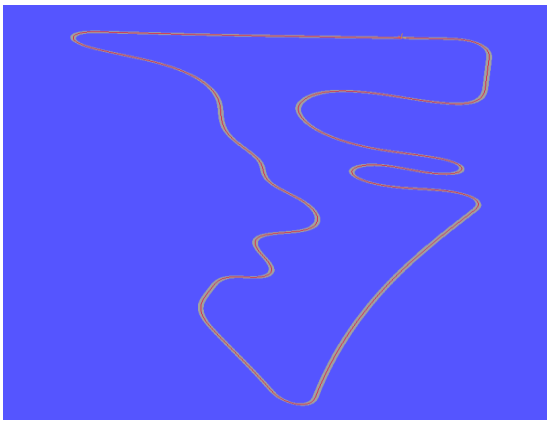


Figure 4.2.3 Track : Austin

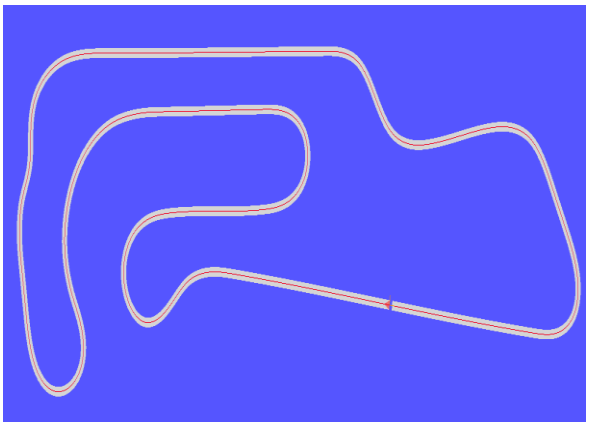


Figure 4.2.4 Track : Oschersleben

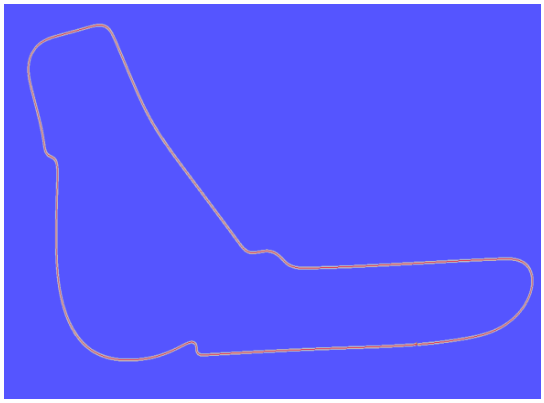


Figure 4.2.5 Track : Budapest

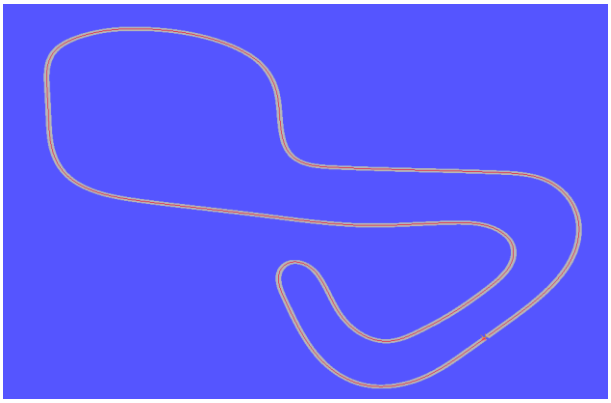


Figure 4.2.6 Track : Brand hatch

All the road tracks are configured with the same track parameters, such as constant friction throughout, no bank angle, and the starting point aligns with the centreline. After performing simulation with each road track in VI-CRT test mode, the data recorded during the event (i.e.) all the output channels could be post-processed and exported in the CSV format in the review mode of the VI-CRT interface. The data recorded during each simulation are compiled together to form a single CSV report which eventually becomes the training data set for the NN construction.

4.3.2 Feature Selection

The CSV data report records a wide range of the output channel data values resulting from the simulation. To achieve the lateral and longitudinal control, the input features are selected considering the parameters which affect the dynamics of the vehicle and also based on the controller setup of the VI-Driver.

Considering the theory of the simplified full vehicle model, its lateral dynamics are described by the following differential equation when assumed for the constant velocity (dV/dt= 0).

$$m(\dot{v} + Vr) - msh\ddot{\phi} = Y_v v + Y_r r + Y_\phi \phi + Y_\delta \delta + F_{ye} \quad 4.1$$

$$J_z \dot{r} - J_{xz}\ddot{\phi} = N_v v + N_\phi \phi + N_\delta \delta + M_{Ze} \quad 4.2$$

$$J_z\dot{r} - J_{xz}\ddot{\phi} = N_v v + N_\phi \phi + N_\delta \delta + M_{Ze} \quad 4.3$$

Where = v - component of the velocity in the y-direction, r - yaw rate, ϕ – roll angle are the degrees of freedom and m - total mass of the vehicle, ms - suspended mass, Jz - the moment of inertia of the whole vehicle about the z-axis, Jx - the moment of inertia of the whole vehicle about the roll axis, Jxz - the mixed moment of inertia in the xz plane, V - longitudinal velocity of the vehicle.

The above three differential equation indicates the three DOF Lateral velocity (v), Yaw rate (r), roll angle (ϕ). The coefficients indicate the factor of side-slipe and camber stiffness of the front and rear axles while representing the factor of the self-aligning moment and roll stiffness.

With the consideration of the above representation of the full vehicle model and the theory behind the VI driver model, the vehicle state feedback channels which describe the DOF related to the lateral and longitudinal dynamics are selected to be the input feature for the Neural Network model. In addition to the vehicle states, the parameters which capture the information about the path, such as curvature and lateral offset concerning the reference trajectory, are also selected to be the input feature. These path features are computed by the driver controller, whose values are obtained from the output channels Path curvature and the Path distance. The details of the selected input features along with the target variables are listed in the below table.

Table 2 Selected input and output features of the Network

Selected Data Channel Measurements	
Vehicle_States.lateral_vel_wrt_road	Lateral Velocity
Vehicle_States.lateral_acc_wrt_road	Lateral Acceleration
Vehicle_States.longitudinal_vel_wrt_road	Longitudinal Velocity
Vehicle_States.longitudinal_acc_wrt_road	Longitudinal Acceleration
Vehicle_States.roll_angle	Roll angle
Vehicle_States.roll_angular_acc_wrt_road	Roll angular acceleration
Vehicle_States.side_slip	Side slip angle

Vehicle_States.yaw_angle	Yaw angle
Vehicle_States.yaw_angular_acc_wrt_road	Yaw angular acceleration
Engine.engine_rpm	Engine Rpm
Driving_machine_monitor.path_x	Target position X with respect to the reference trajectory
Driving_machine_monitor.path_y	Target position Y with respect to the reference trajectory
Gyro.gyro_X	Actual position X with respect to Gyro reference
Gyro.gyro_Y	Actual position Y with respect to Gyro reference
Driving_machine_monitor.path_curvature	Path curvature of with respect to the reference trajectory
Driving_machine_monitor.path_distance	Lateral offset with respect to the reference trajectory
Driver_demands.steering	Steering angle
Driver_demands.throttle	Throttle
Driver_demands.brake	Brake

With the same vehicle state feedbacks used for the lateral control prediction, the longitudinal control output of throttle and brake demands are not satisfactory. Hence a longitudinal control is implemented separately with input features slightly varied compared to the network dedicated for the lateral control. The CSV report, which contains the data values of all the selected measurements of the data channels, is uploaded to the Google Colab space using a Pandas library as a Data frame.

4.3.3 Data Pre-Processing

Data pre-processing is one of the most important steps in the creation of machine learning models to transform the data values to a suitable form which helps the model to learn efficiently from the given data set. There are several data pre-processing techniques available in general, and its choice is based on the type of dataset being handled and also its intended application. A few of the data pre-processing steps which are followed for the data preparation in this work are briefed in this section.

Data Scaling

Data scaling is the most widely used pre-processing procedure, especially in regression problems. Basically, during the supervised learning (mapping of input to the desired output as per the training data) of the deep neural network, the weights of the model are initialized to small random values and updated via an optimization algorithm in response to estimates of an error on the training dataset. For the use of small weights in the model and the use of error between predictions and expected values, the scale of inputs and outputs used to train the model is an important factor. Unscaled input variables can result in a slow or unstable learning process, whereas unscaled target variables on regression problems can result in exploding gradients, causing the learning process to fail.

In addition, the differences in the scales across input variables may increase the difficulty of the problem being modelled [17]. An example of this is that large input values (e.g. a spread of hundreds or thousands of units) can result in a model that learns large weight values. A model with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error.

Data normalization and standardisation are the two normally adopted data-scaling procedures during the preparation of the data set. A good rule of thumb is that input variables should be small values, probably in the range of 0-1 or standardized with a zero mean and a standard deviation of one [17]. In our case, since our input features cannot be standardised, the normalization procedure is adopted to rescale the values in the range of [0,1].

This normalization is performed using the scikit-learn (machine learning library for the data analysis) object MinMaxScaler. The scalar object performs rescaling by performing the below transformation on all the data values.

$$y = (x - \min)/(max - \min) \tag{4.4}$$

Where max, min is specific to each feature that is subjected for the data-scaling.

Data Augmentation

In general, training the deep neural network models with more data can provide a good result in terms of model accuracy. Data augmentation is a technique by which the data variation can be introduced, which improves the model ability to generalize from the provided learning data.

Data balancing is a necessary factor for eliminating the model to be biased around a certain prediction range. For example, in the case of the steering angle prediction, if the training data set has more data representing the left turn, this makes the model to be biased towards the left steering angle. In our case, the target data such as steering angle, throttle and brake are checked for balance. The overbalancing for the steering angle is adopted, as shown below. When visualised the histogram of the steering angle in the training dataset, it is evident that it has a normal distribution, but the steering value around zero is very high compared to other values. To avoid the model being

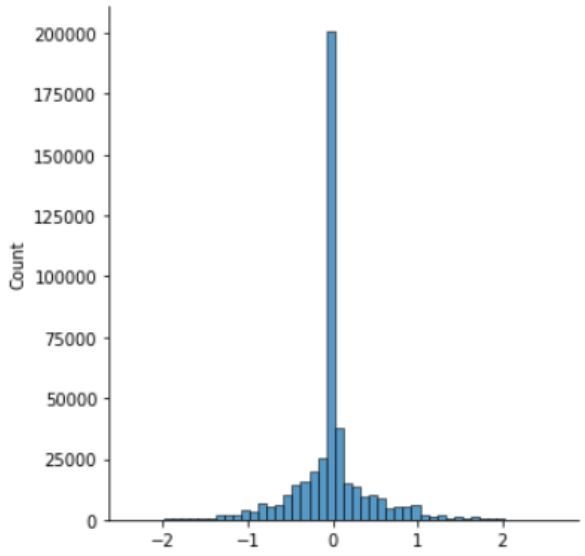


Figure 4.3 Histogram before Data Augmentation - Steering Values

biased towards the straight, data augmentation is applied to create a more balanced dataset without removing the data samples(overbalancing). The augmentation is as follows:

- Steering values greater than 0.01 and less than 1 are multiplied 5 times
- Steering values greater than 1 and less than 2 are multiplied 3 times
- Steering values less than 0.01 and greater than 1 are multiplied 5 times
- Steering values less than 1 and greater than 2 are multiplied 5 times

The histogram plot after the data augmentation is as below.

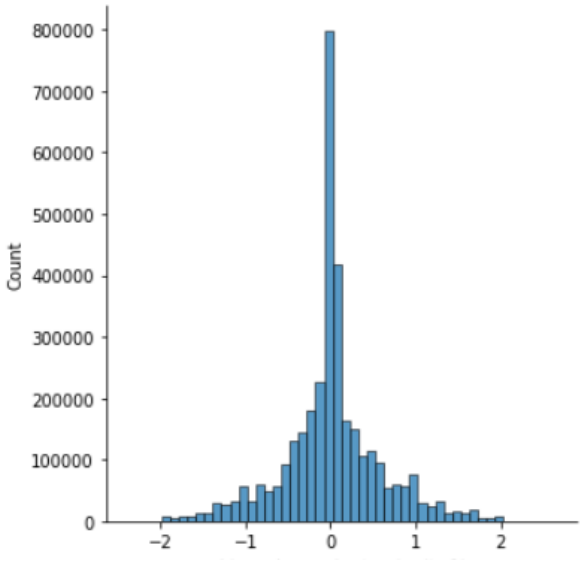


Figure 4.4 Histogram after Data Augmentation - Steering Values

4.4 Training

To achieve efficient training, several configurations of the Multi-Layer Perceptron network are tested by trial and error. During the exploration, a grid search algorithm is applied to check on the different combinations of the hyperparameters. This is done with the purpose to find the optimal hyperparameters setting for the given training dataset.

4.4.1 Network Architecture

The main goal here is to feed the dynamic vehicle parameters as input to the network and get the output of vehicle control demands steering, throttle and brake. The architecture of three different networks based on control output is discussed in this section

Constant Velocity & Max Performance – Lateral Control

For the task of achieving the lateral control for the constant velocity conditions, the architecture of the multi-input and single output MLP network is selected. The input features here are vehicle states such as lateral velocity, longitudinal velocity, side slip angle, yaw angle, roll and the current vehicle position(Gyroscope measurement), along with the reference trajectory details such as target vehicle position, path curvature and path distance. On the other side, the output and target is the lateral control (i.e. the steering angle). Figure 4.5 shows a simplified layout of the network input, hidden and output layers, along with the input and output nodes.

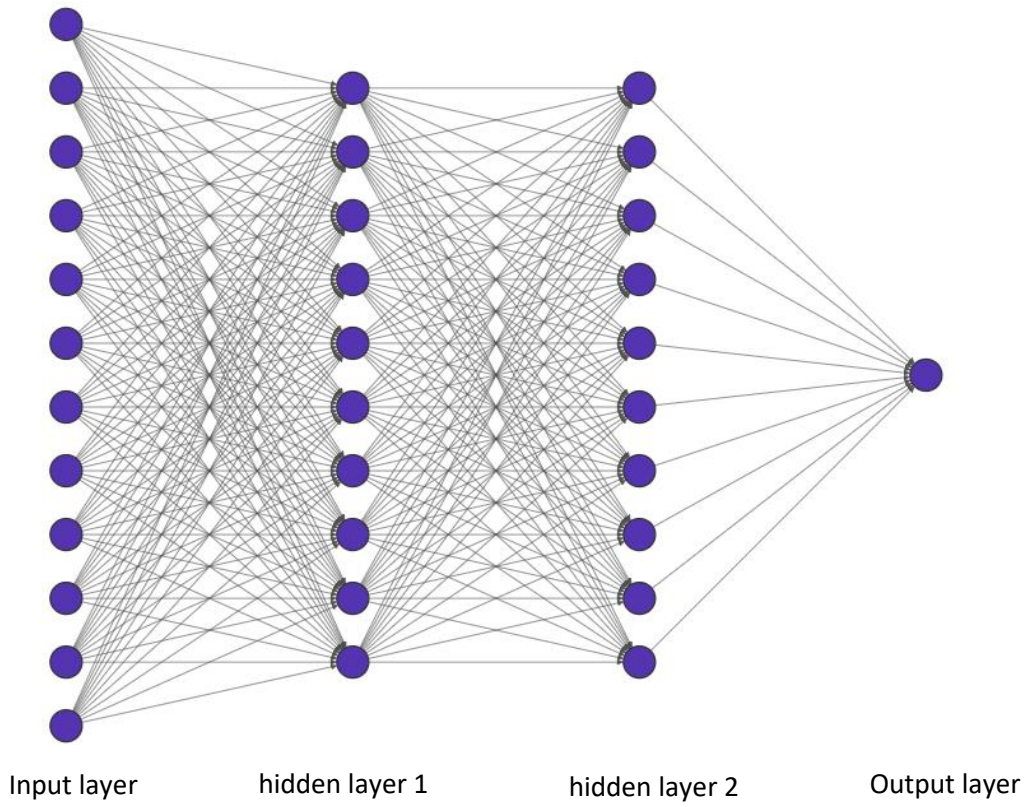


Figure 4.5 NN Architecture Diagram - Lateral Control

After several experimentations and with the grid search result, the configuration with an input layer, two hidden layers and an output layer with the nodes 300, 200,100, 1 in each layer showed the better result. The Keras model summary with the number of computational parameters involved in the network is given below.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 300)               3600
_________________________________________________________________
dense_13 (Dense)             (None, 200)               60200
_________________________________________________________________
dense_14 (Dense)             (None, 100)               20100
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 101
=================================================================
Total params: 84,001
Trainable params: 84,001
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.6 Model Summary - Lateral Control

The dataset consists of 450406 data samples which are divided into training samples of '315284' and validation samples of '135121'. The batch size is selected as 50, and the number of epochs is selected as 100; this splits the entire training dataset into a batch of 50 samples. When this entire split passes through the network, this cycle represents one epoch. While training, the model will be validated with the validation set whose loss, along with the training loss, will be given as output for each epoch. The Mean Squared Error (MSE) and Mean Absolute Error (MAE) are the two-loss functions tested; the use of MSE showed a better result while fitting the model with the training data. The residual error is optimized by the Adam optimizer with the learning rate 10e-03; this worked well with the combination of the Relu activation function in all the layers except the output layer for which linear activation function is used. To prevent the training procedure from resulting in the overfitted model, the early stopping call back is introduced, which monitors the validation loss metric for the patients level of 10 epoch; if the validation loss keeps increases continuously for 10 cycles, the training procedure will be terminated. With the call back triggered, the training is completed before 100 epochs which helps the model to generalize well with the data provided. The summary of the architecture, along with its hyper- hyperparameter's, is mentioned in the below table 2 and 3.

Table 3 Configuration and Hyperparameters - Constant velocity Lateral control

Parameters	Value
Learning rate	10 e-04
Optimizer	Adam
Loss Function	MSE
Activation function	Relu
Number of Epochs	100
Batch Size	50
Top Layers	300, 200, 100, 1

The architecture of the MLP model for the prediction of the lateral control in the case of dynamic limit condition is almost the same as that constant velocity model, with few exceptions in the value of the hyperparameters chosen for training the network. The summary of the model hyperparameter's is described in the below table.

Table 4 Configuration and Hyperparameters - MaxPerformance Lateral control

Parameters	Value
Learning rate	10 e-03
Optimizer	Adam
Loss Function	MSE
Activation function	Relu
Number of Epochs	100
Batch Size	400
Top Layers	300, 200, 100, 1

Max Performance – Longitudinal Control

The MLP configuration for the longitudinal control is implemented with the multi-input and multi-output architecture Keras model as it is required to predict targets such as throttle and brake demand. Compared to the lateral control model, here, different input features are fed to the network, which has good relation with respect to the longitudinal control target parameters. The input features here are vehicle states such as lateral acceleration, longitudinal acceleration, side slip angle, yaw angular acceleration, roll angular acceleration and the current vehicle position (Gyroscope measurement), along with the reference trajectory details such as target vehicle positions, path curvature and path distance. The output side of the network contains the two target variables, which are, in this case, is throttle and brake. The architecture figure 4.7 shows a simplified layout of the network input, hidden and output layers, along with the input and out nodes.

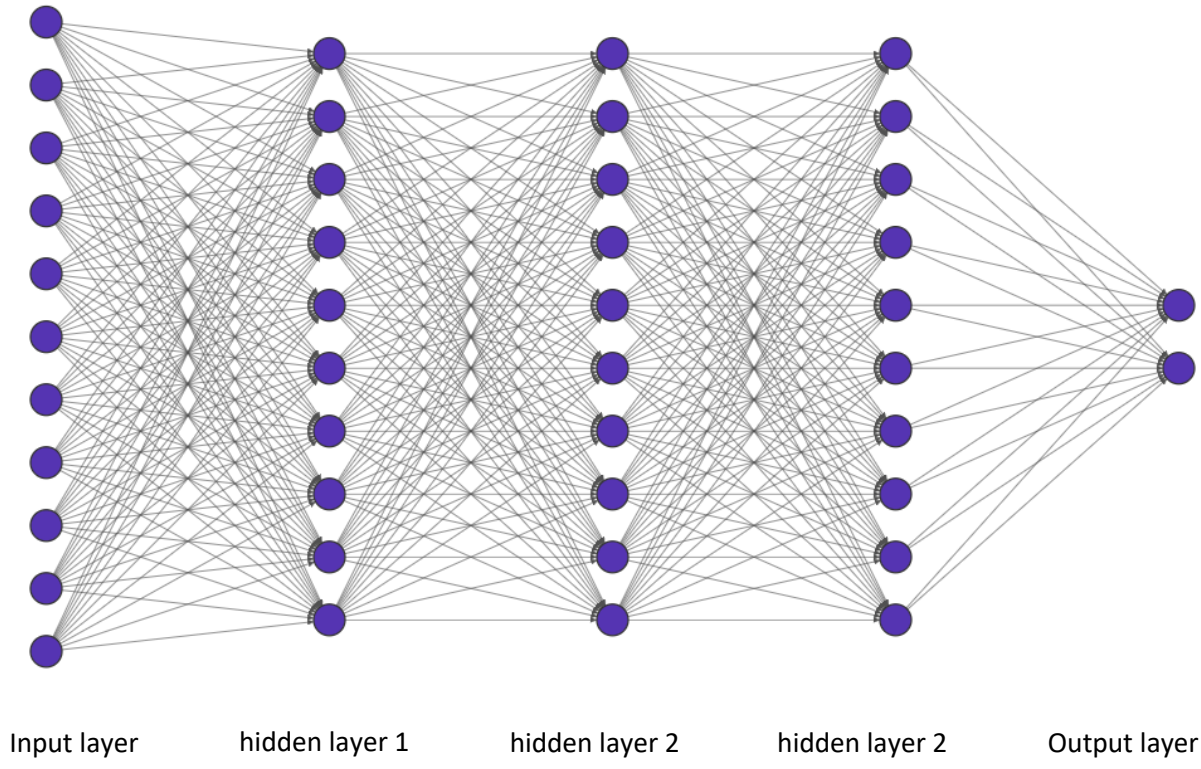


Figure 4.7 NN Architecture Diagram - Longitudinal Control

Here three hidden layers with a greater number of nodes are used compared to the lateral control model as it involves high non-linearity to make the multi-output prediction. The Keras model summary of this MLP model, along with the hyperparameters setting used for the network training procedure, is shown below in the figure and table.

```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_127 (Dense)            (None, 300)               2700
_________________________________________________________________
dense_128 (Dense)            (None, 200)               60200
_________________________________________________________________
dense_129 (Dense)            (None, 150)               30150
_________________________________________________________________
dense_130 (Dense)            (None, 100)               15100
_________________________________________________________________
dense_131 (Dense)            (None, 2)                 202
=================================================================
Total params: 108,352
Trainable params: 108,352
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.8 Model Summary - Longitudinal Control

Table 5 Configuration and Hyperparameters - MaxPerformance Longitudinal control

Parameters	**Value**
Learning rate	10 e-02
Optimizer	Adam
Loss Function	MSE
Activation function	Relu
Number of Epochs	50
Batch Size	400
Top Layers	300, 200, 150, 100,1

4.4 Online Validation Setup

In addition to the offline prediction evaluation of the built-in NN model architectures, the real-time validation setup is established with the help of the Simulink interface. Though this integration of the NN model with the VI-CarRealTime can also be done with the help of plugin API's, for the sake of simplicity, this is done in the MATLAB/Simulink environment. As described in the co-simulation in section 3.3, the co-simulation of VI-CarRealTime events can be performed by importing the vehicle S plant in the Simulink space, which will get its simulation details and vehicle setup through the XML file generated from VI-CRT. The main idea here is to integrate the prediction output from the NN model to the vehicle model as an input while performing the co-simulation in the Simulink interface. The basic layout of this integration process is represented by the below block, figure 4.9.

Figure 4.9 Integration Environment Layout

To invoke and receive the prediction output from the Neural Network model, calling python from the MATLAB procedure is adopted. For this purpose, a python script is utilized, in which the built-in Keras model, which is saved from the Google Colab notebook as .h5 format, is loaded. This python script is invoked from the Simulink by passing the vehicle state feedback (retrieved from the output channel of vehicle s function) from the vehicle model during the co-simulation. This vehicle state feedbacks are feed as input to the NN model and makes its prediction for each integration step and returns it to the Simulink. The returned prediction value is connected to the input channel of the vehicle model. With this integration environment, the Keras model prediction could be validated for different VI simulation events. The architecture of the integration environment of the Simulink with the built-in Keras model is shown in the below layout figure 4.10.

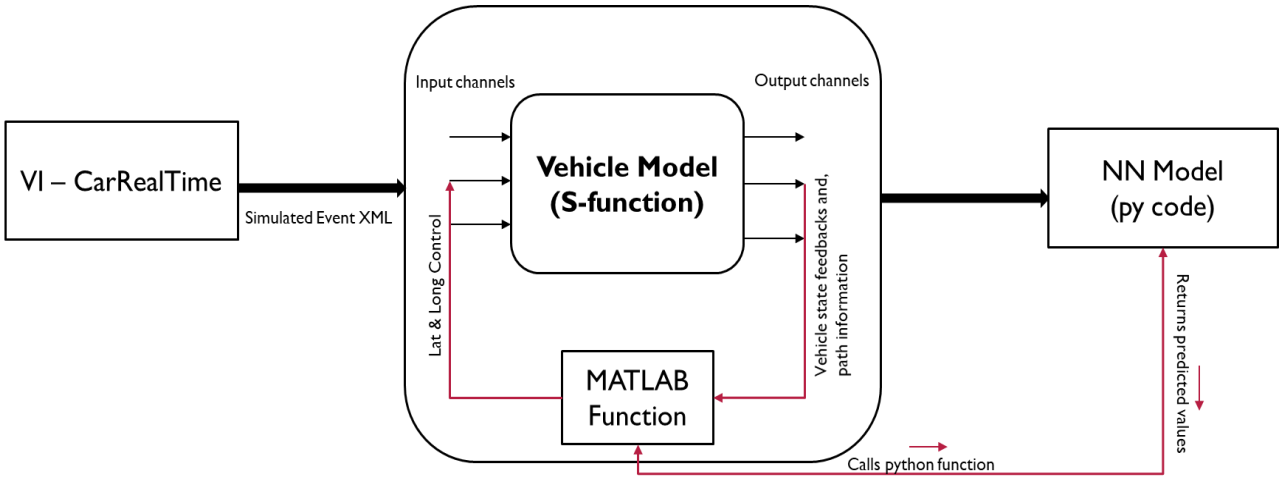


Figure 4.10 Simulink co-simulation setup for NN validation

The procedure of calling python from MATLAB is implemented in the MATLAB function block, as shown above. During the co-simulation, the VI-Driver controller's control (lateral and longitudinal control) are cut-off. However, it will still be active to provide the information related to the path from the output channels during this co-simulation process. These steering, throttle and brake controller (to cut off) settings are modified in the VDF event file.

Chapter 5

Methodology – Trajectory Prediction

5.1 Overview

This chapter covers the description of the algorithms and methodology adopted for building a Neural Network Model for optimal trajectory prediction. Like the above methodology for vehicle control, an iterative approach is followed here as well, starting from the system for a simple solution followed by the complex one.

The main aim behind the construction of this model is to find a target trajectory to be provided for the vehicle control model. In this project, the global trajectory planning using the NN model is designed. At first, the research plan was to select the target trajectory from the set of trajectory candidates which are generated based on the VI-Path Builder script (a python script that generates a different trajectory based on the given centreline of the track by taking the factors of corner-cutting) for a given road track.

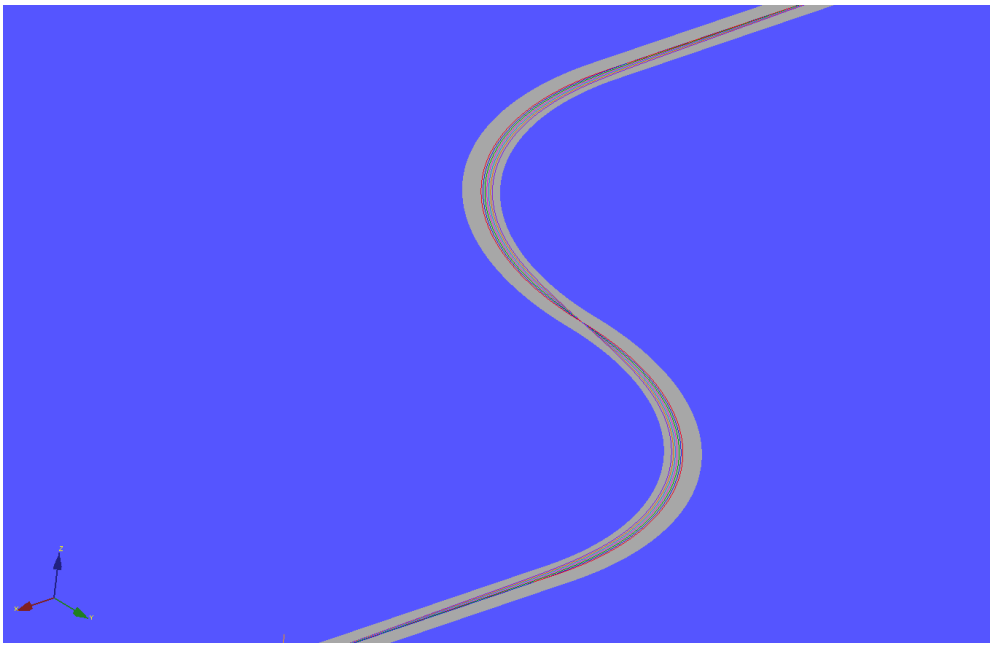


Figure 5.1 Different trajectories generated based on Corner-cutting

The different trajectories based on the corner-cutting value ranging from 0 to 1 for the simple 'S' curve track is shown in the above figure 5.1. However, when tried to formulate this as a classification problem to select a target trajectory from the set of different trajectories based on the feature candidates, such as tracking accuracy of the trajectory, the result from the NN model is not satisfactory. Hence as an alternative and to reduce the complexity, this global trajectory planning is deduced as a regression task that involves predicting the optimal trajectory points based on the road track parameters.

5.2 System Design

As the main goal here is to predict the optimal trajectory points, the target trajectory from each road track are selected, and the Neural Network is trained to make the optimal trajectory prediction for any given road track. The system definition involves taking the input parameters as track details, such as the information on the inner and outer boundary along with the curvature of the track at each point. These selected input features, along with output which is selected here to be the trajectory point in terms of X and Y coordinates, are provided to the NN architecture model during the training process.

At first, the model is built with desired trajectory candidate as a centreline for each track (i.e. trained with the centreline of the multiple tracks as target trajectory to predict centreline for any given track), once its performance is studied and the selection of the network hyperparameters is evaluated, this model is extended to make the prediction of optimal trajectory which is the fastest trajectory in our case. The fastest trajectory refers to the one selected from the above trajectory set, which shows as minimum lap time for the respective road track. The supervised learning procedure is applied to the NN model, which is fed with the track features of multiple road tracks with the aim of getting the global optimal trajectory prediction.

5.2.1 Road Boundary Generation

As described in the system definition, the model is designed to take the road boundary information as input features; the inner and outer boundaries of the different road track used for the training of the Neural Network is generated using a MATLAB algorithm.

Basically, this algorithm uses the centre line of each road track as a reference to generate the inner and outer boundaries. With the track width as constant at 10m, this algorithm generates boundaries using the general triangulation law, which involves finding the right and left point with reference to the centre point by calculating the distance between its succeeding centre point using the distance formula.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5.1}$$

Where (x_1, y_1) – centre point coordinate, (x_2, y_2) - succeeding point coordinate.

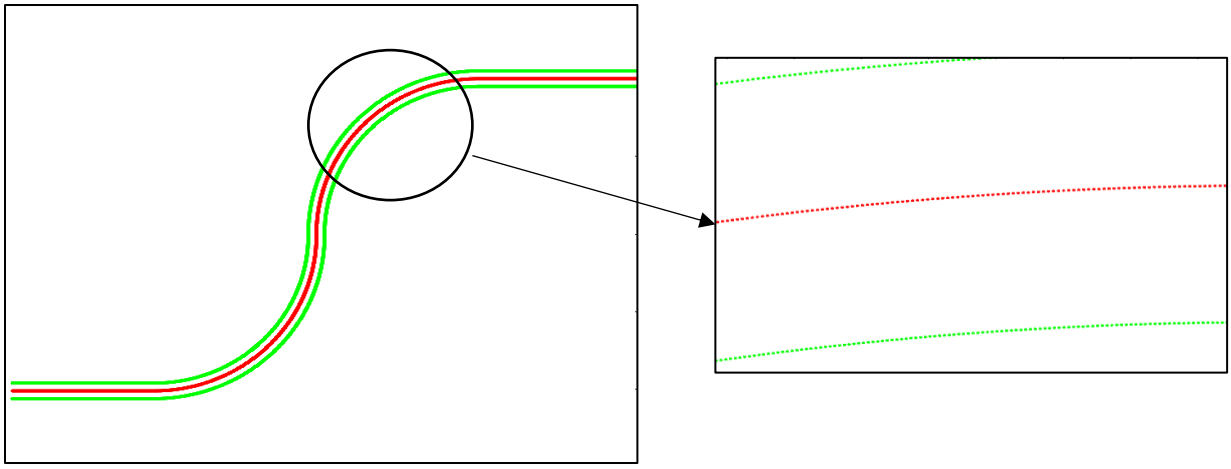


Figure 5.2 Inner and outer Boundaries plot

The left and right boundaries generated for the simple S curve is shown above in figure 5.2. The representation of the inner and outer boundaries, which are generated based on the centre-line along with the optimal trajectory of the particular track, is shown in figure 5.3.

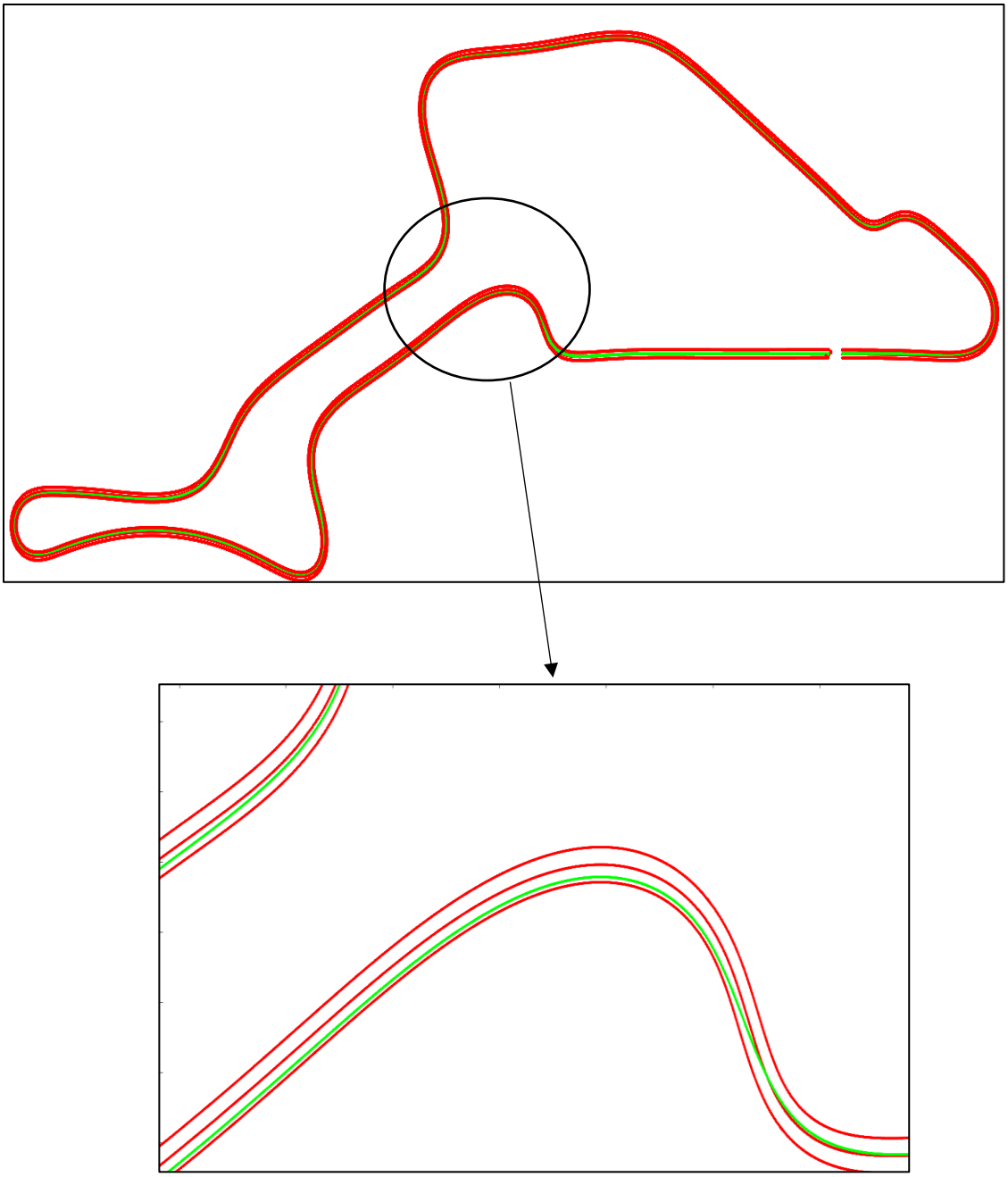


Figure 5.3 Tack's optimal trajectory line along with the road boundaries

Here the green line is the optimal trajectory (fastest) chosen for this respective track which corresponds to the corner-cutting value of 1(maximum). Since the algorithm above generates the inner and outer boundaries based on the centre line, the boundary coordinates are aligned here only to the centre point coordinates. Concerning the optimal trajectory point coordinates, there existed a misalignment with respect to the left and right coordinate points as its track total distance and curvature has variation compared to the centre-line. Hence to find the optimal trajectory point in

line with its boundaries, the normal lines to the centre points, which intersect with the left and right boundaries, are plotted as shown in figure 5.4.

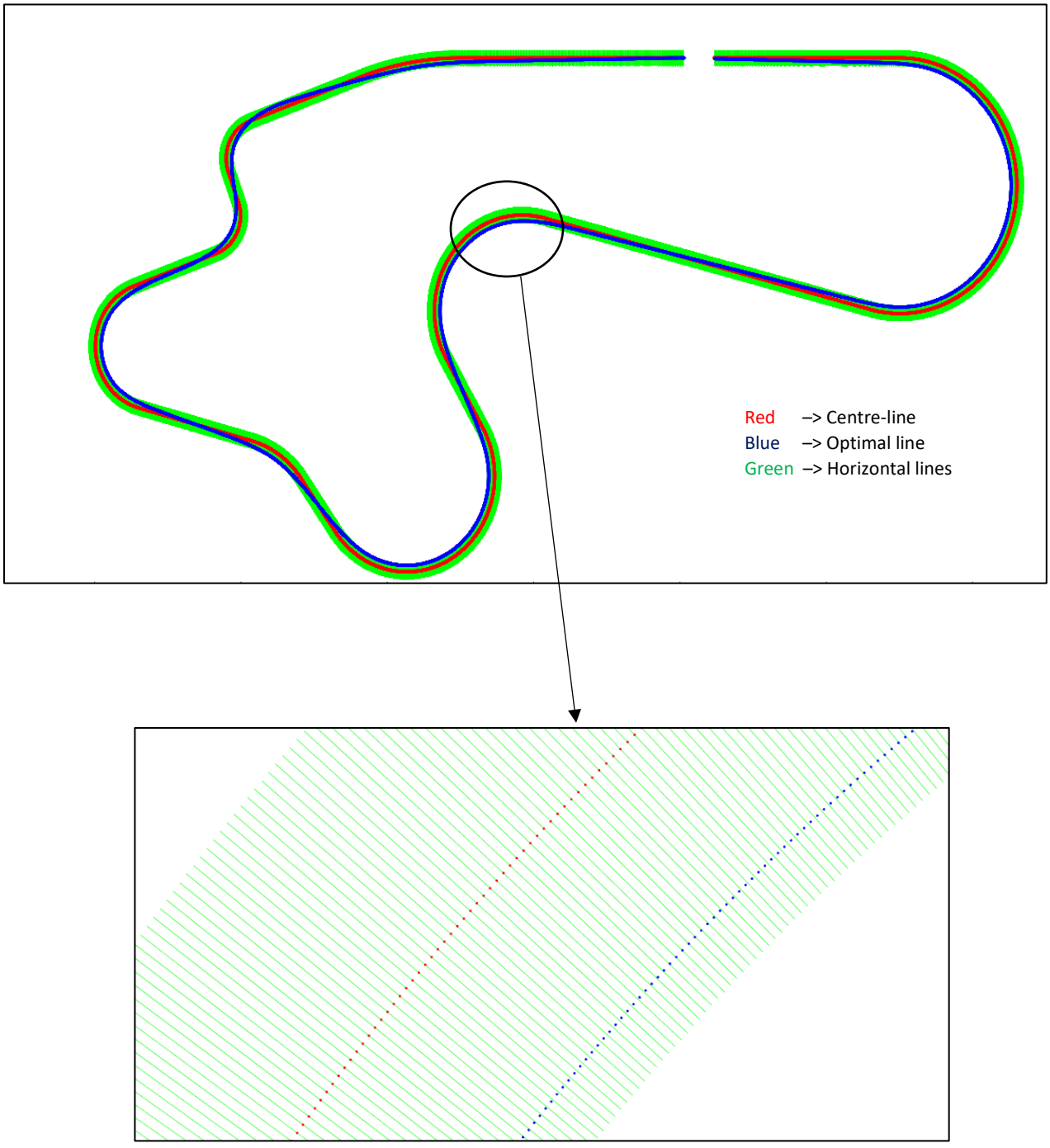


Figure 5.4 Optimal trajectory point's misalignment with the centre points

This misalignment is solved by finding the intersection point of the optimal line (blue line shown in the below figure) with each normal line with the help of the separate curve intersection MATLAB algorithm [18]. This algorithm is slightly modified to suit our purpose of finding the intersection point.

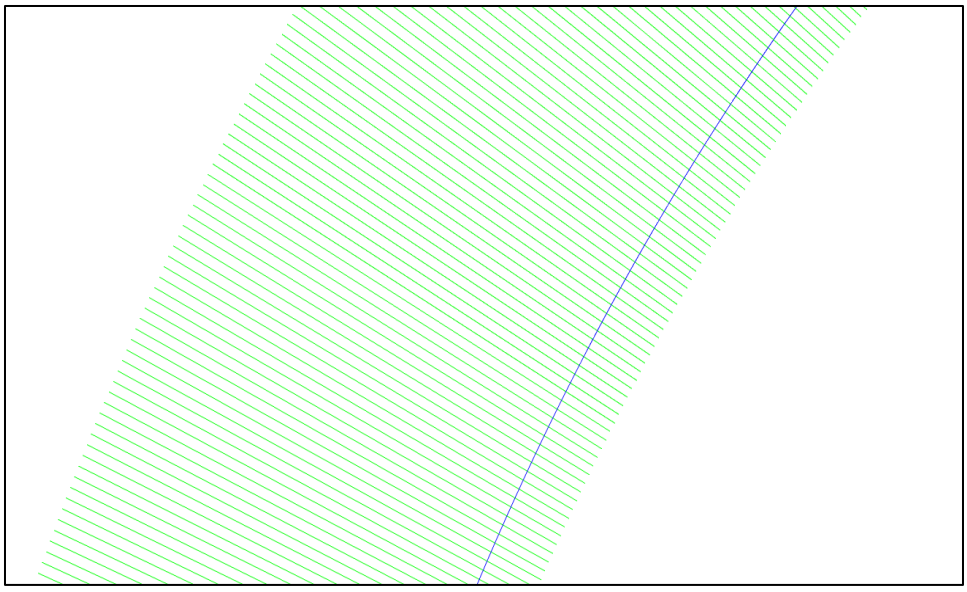


Figure 5.5 Optimal trajectory line (blue)

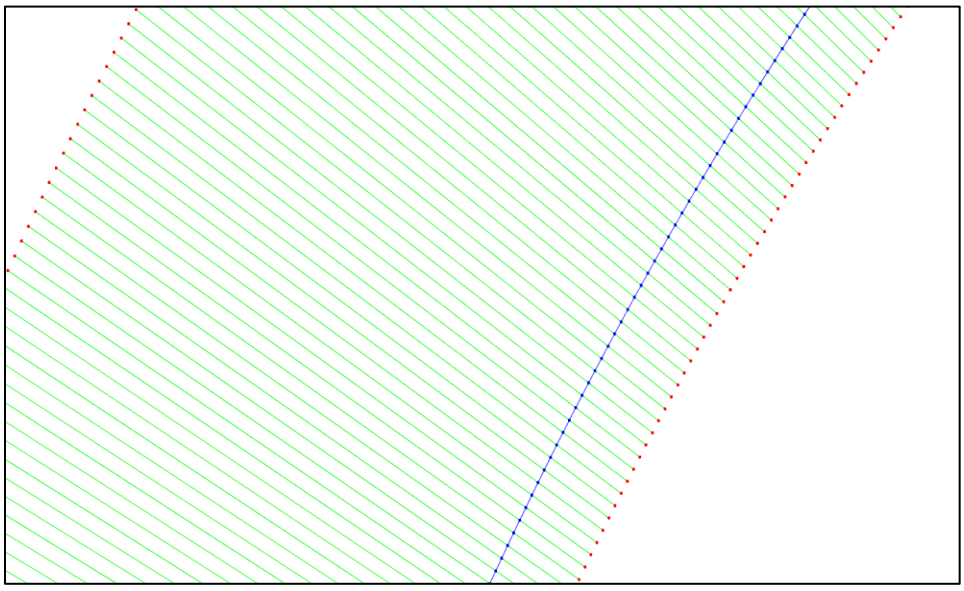


Figure 5.6 Optimal trajectory point in line with boundary coordinates

These optimal trajectory points in accordance with the track boundaries are gathered for the different road tracks used for the training dataset preparation.

5.2.2 Sliding Window Approach

This approach is based on the notion of the human driver look ahead perception on the tracks to identify the desired trajectory line. Normally, the driver line at any point of the track is influenced by the preceding as well as the upcoming track information (for instance, curvature), which results in the current vehicle position. Thus to enhance the performance of the Neural Network model in trajectory prediction, bidirectional 'Foresight' - both fore and aft of the current position are adopted here, thereby enabling the network to learn both how the vehicle arrived at the current position and where the trajectory heads next.

For this purpose, the road track is fragmented into several segments (windows), which considers the forward and backward trajectory points to the current vehicle position. The single window of the approach is shown in figure 5.7

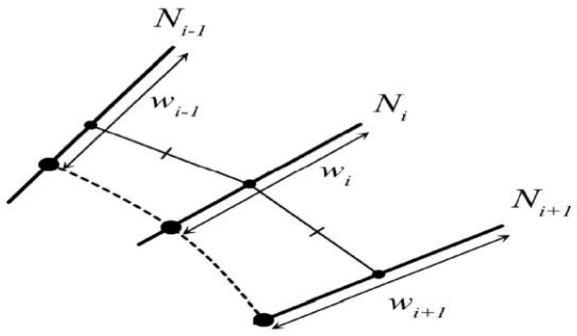


Figure 5.7 Single Sliding Window

Where w_i – current trajectory point, w_{i+1} - forward trajectory point, w_{i-1} - backward trajectory point.

This approach allows the entire tracks to be split it several overlapping sections with a fixed number of trajectory points which enables the supervised learning procedure to be more efficient. Also, the network learns from a larger number of windows of racetracks, enabling it to generalise across behaviour from multiple circuits with similar features. The sliding window approach vastly

increases the quantity of data provided by each circuit, thereby reducing the number of circuits required in the training dataset.

5.3 Dataset Preparation

All the data's required for the trajectory prediction model is extracted from the trajectory information resulting out of the VI-CarRealTime simulations. As our primary aim is to build a model that predicts the fastest trajectory for any given track, the model is first formed to predict the centreline for any track. For this purpose, the simulations of the different road tracks are performed in the VI-CRT. The trajectory (centre-line) information, along with the curvature details of each track, is extracted from the output channel. With centre-line information (in terms of X and Y coordinates) as a reference, the inner and outer boundaries of the respective road track are extracted with the help of the MATLAB algorithm explained in section 5.2.1.

Similarly, the optimal trajectory information resulting from the simulation with the trajectory corresponding to the minimum lap-time is also mapped according to the above-generated road boundaries with the help of the algorithm discussed in section 5.2.1. With the information on the desired trajectory coordinates and its respective inner and outer boundary coordinates and curvature details, the training dataset is prepared for supervised learning. Similar to the previous NN model, around 12 road tracks simulation data are used, which includes few tracks from the VI-CarRealTime road database, and the rest of the tracks are created in the VI-Road interface based on the centreline coordinates from the TUMFTM – race track database (an open-source GitHub repository created and maintained by TUM – Institute of Automotive Technology).

Figure 5.8.1 Track : VI - Track

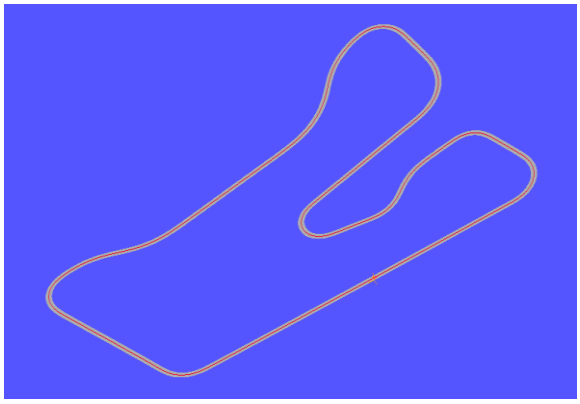


Figure 5.8.2 Track : Nürburgring

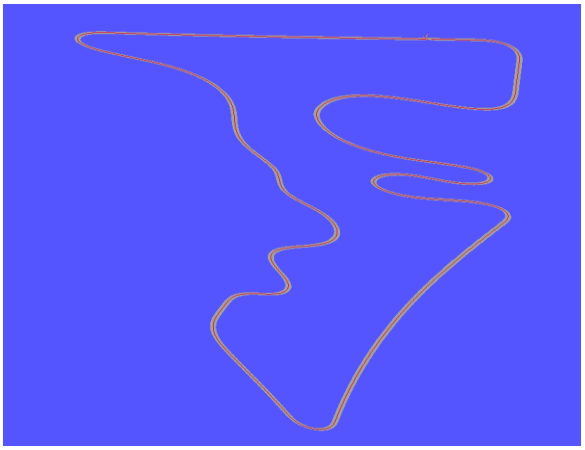


Figure 5.8.3 Track : Austin

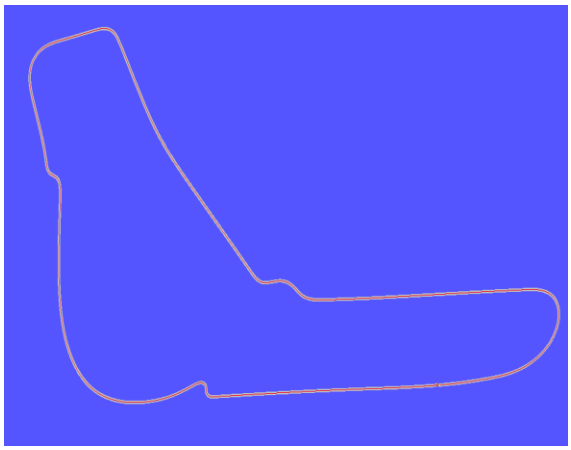


Figure 5.8.4 Track : Budapest

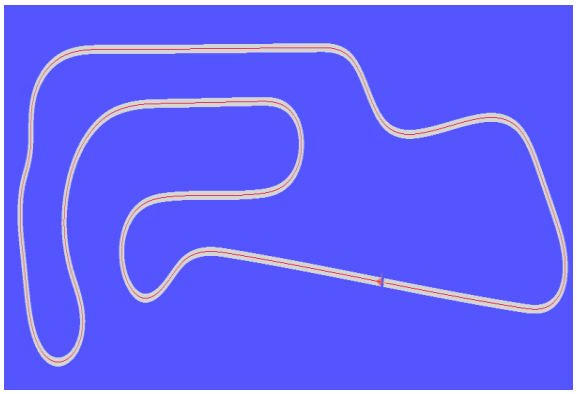


Figure 5.8.5 Track : Oschersleben

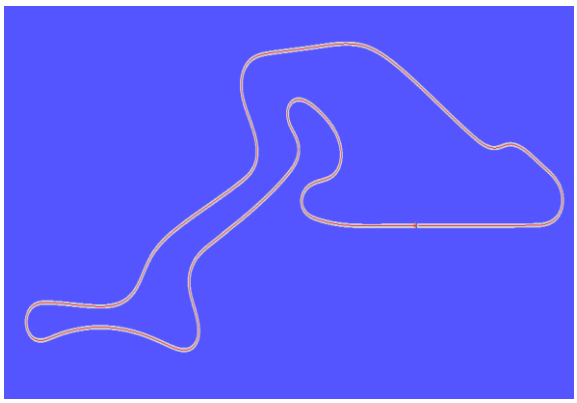


Figure 5.8.6 Track : Nürburgring

After importing the CSV report with data gathered in the Colab notebook, the pandas library is used for data sliding to implement the sliding window approach discussed in section 5.2.2. To transform the dataset into several overlapping sections, each column values are shifted forward and backwards to aid the bi-direction foresight in the prediction. The sampling amount (i.e. the number of trajectory points considered in the single window) affects the prediction accuracy; the higher the sampling rate more the accuracy on the trajectory prediction. In order to reduce the complexity and minimize the dimension of the input feature to the network, the sampling frequency of one is adopted here, which includes only one step before and after the data point in the single window.

Listed below are the features selected and the augmented features present in the data frame used for training

Table 6 Selected input and output features of the Network

Features in Data frame	
Path_X	Trajectory X-Coordinate
Path_Y	Trajectory Y-Coordinate
Path_Curvature	Trajectory Curvature
Xin	Inner Boundary X-Coordinate
Yin	Inner Boundary Y-Coordinate
Xout	Outer Boundary X-Coordinate
Yout	Outer Boundary Y-Coordinate
X-1, X+1	Prior and after data shift of X-Coordinate
Y-1, Y+1	Prior and after data shift of Y-Coordinate
Xin-1, Xin+1	Prior and after data shift of inner boundary X-Coordinate
Yin-1, Yin+1	Prior and after data shift of inner boundary Y-Coordinate
Xout-1, Xout+1	Prior and after data shift of outer boundary X-Coordinate
Yout-1, Yout+1	Prior and after data shift of outer boundary Y-Coordinate
C-1, C+1	Prior and after data shift of Curvature

5.4 Network Architecture

As this MLP model is formed in the same methodology as the previous NN model, the framework details and other standard model settings are not covered again in this section.

The main goal of this model is to predict the trajectory points in terms of X and Y coordinates by taking the input of the track details such as the boundary information and the path curvature. As the problem here is formulated to be a regression one, the Neural Network model selected here is also a Multilayer Perceptron since it is best suited for the regression task and the property that this network needs to have minimum computation time for generating the trajectory prediction.

The MLP architecture required for this task will have a multi-input and multi-output structure. The network contains the 15 input features which capture the details on the road track information, such as the inner and outer boundary coordinates and the path curvature, along with its data

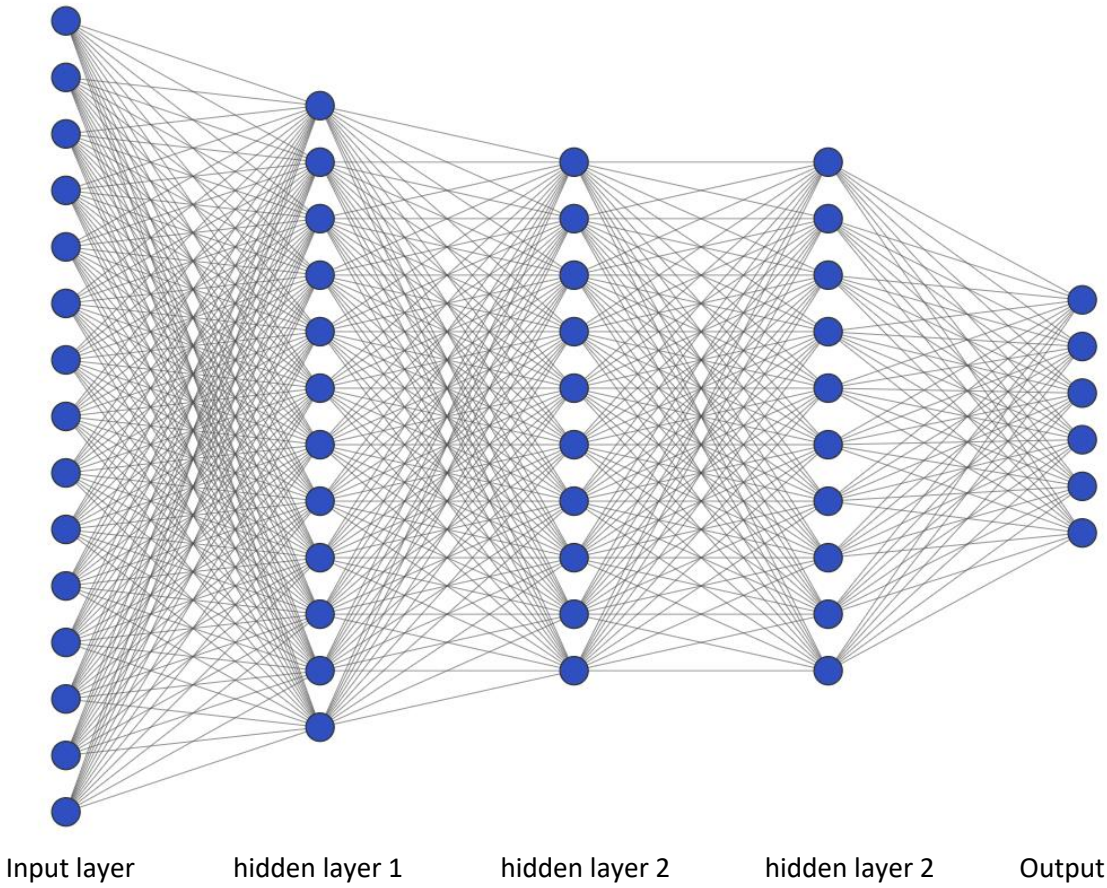


Figure 5.9 NN Architecture diagram - Trajectory Prediction

shifts (Xin-1, Xin, Xin+1, Yin-1, Yin, Yin+1, Xout-1, Xout, Xout+1, Yout-1, Yout, Yout+1, C-1, Curvature, C+1),on the other side of the network is 6 target parameters(X-1, X, X+1, Y-1,Y, Y+1) which are the trajectory points with the sampling rate 1.

The simplified architecture diagram with the number of nodes in the input and output layer, along with the number of the hidden layer, is shown in figure 5.9.

The data samples as a whole gathered for this model learning contains 192990 samples which is split in the range of 80:20 into a training dataset of 154392 and the validation dataset of 38598. The final Keras model involves one input layer, three hidden layers and an output layer with a number of nodes 200, 100, 100, 50, 6, respectively. The summary of this Keras model is as shown below

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 200)               3200
_________________________________________________________________
dense_26 (Dense)             (None, 100)               20100
_________________________________________________________________
dense_27 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_28 (Dense)             (None, 50)                5050
_________________________________________________________________
dense_29 (Dense)             (None, 6)                 306
=================================================================
Total params: 38,756
Trainable params: 38,756
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.10 Model Summary - Trajectory Prediction

The Network is updated during the learning process using the Adam optimization algorithm, with the parameter settings such as learning rate 10 e-04, beta_1 0.9, beta_2 0.999, epsilon 1e-07. The loss functions such as ‘MSE’ and ‘Huber’ are tested for accuracy, and the final grid search score showed Huber loss as the best choice. The Adam optimizer, in combination with the aforementioned setting, resulted in the best possible prediction accuracy. Further, the data samples are split into batches of 15, and the model is trained for the 50 number epochs. As a step of

generalisation and to prevent the overfitting of the model, the call back strategy is adopted. The early stopping with the patience level of 3 to monitor the validation loss during training is configured, which makes the loss history graph with no signs of overfitting. A brief description of the model feature and the hyperparameters are listed in the below table.

Table 7 Configuration and Hyperparameters - Trajectory Prediction Network

Parameters	Value
Learning rate	10 e-04
Optimizer	Adam
Loss Function	Huber
Activation function	Softmax
Number of Epochs	50
Batch Size	15
Top Layers	200, 100, 100, 50, 6

The model, when built, initially showed noise in its prediction of the trajectory. However, after experimentation with a comprehensive combination of the hyperparameters, especially the batch size and the activation function, the Softmax activation function resulted in the smooth trajectory prediction with the noise elimination. Further, the sampling factor also influenced the smoothness in the prediction of the trajectory (the higher the sampling frequency, the more the smoothness in the trajectory prediction). As previously stated, to reduce the dimensionality, sampling of 1 is considered here, with the network structure and parameters providing the best possible prediction result.

Using the described model structure and the methodology, the Neural Network model is trained to predict the centre-line and the optimal trajectory line for any given track separately. The model showed good prediction in both cases with less computational time involved.

Chapter 6

Results and Discussion

6.1 Training Results

All the final neural network models used for the testing showed the validation loss in the range of 10e-04. The training loss history of each model is reported below, which shows the metrics values attained at each epoch. The metrics here are the training loss and the validation loss (these losses denotes the value difference in the expected output with respect to the training and the validation dataset).

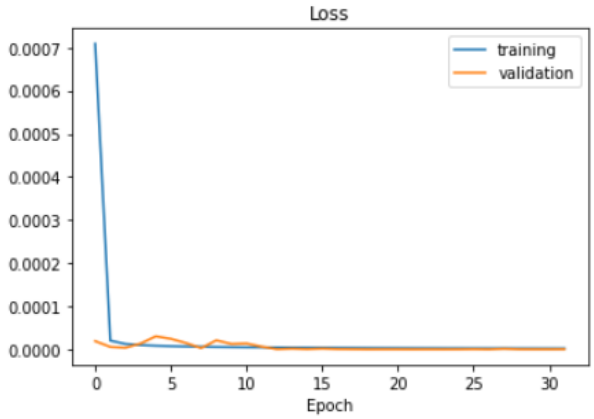


Figure 6.1 Constant Velocity - Lateral Control

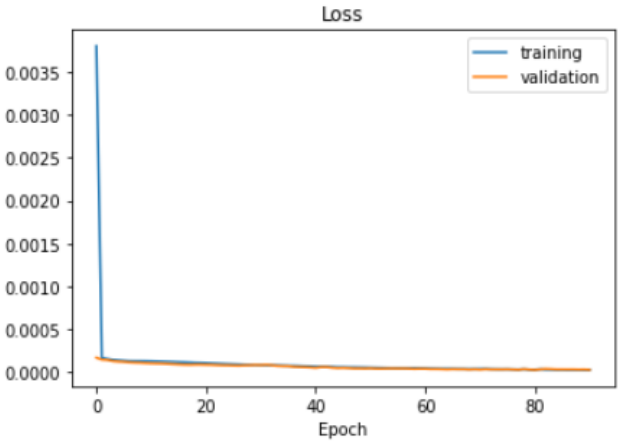


Figure 6.2 MaxPerformance - Lateral Control

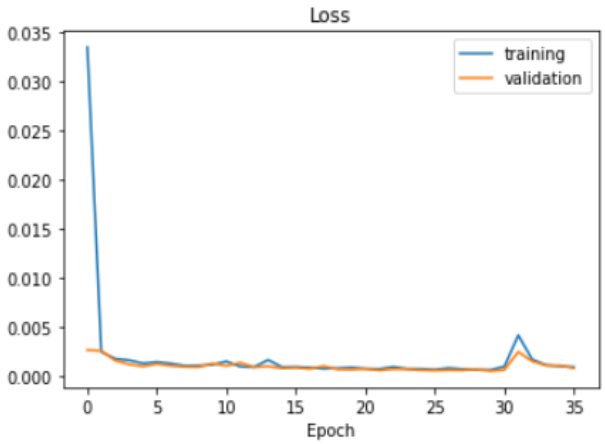


Figure 6.3 MaxPerformance - Longitudinal Control

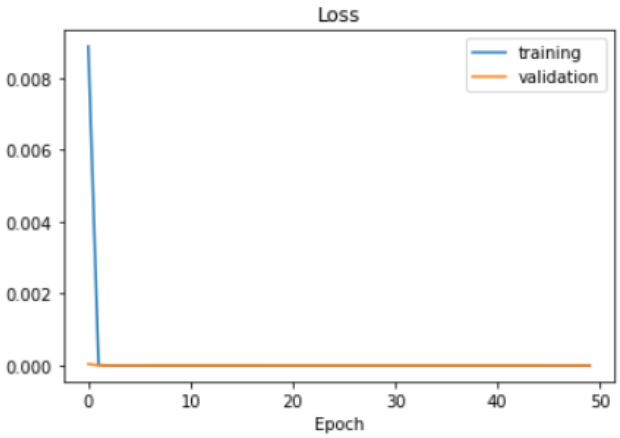


Figure 6.4 Trajectory Prediction

Though all the Neural Network models for both the vehicle control and trajectory prediction are trained with the epoch value 100, early stopping call back on each model stopped the training before 100 epochs. This is done to prevent the overfitting of the model to the training data, which results in poor generalisation of the resulting model. It is evident from all the training graphs reported above that the final models are not overfitted, which will be the case when the validation curve(orange) starts to increase above the training curve(blue) after training for a certain number of epochs. The constant velocity model, though it showed initial fluctuation in terms of the validation loss, manages to achieve a better result at around 32 epochs, with the validation loss in the range of e-04. Comparatively, the Max-Performance lateral control model showed a most consistent result from training and achieved a good result at around 80 epochs. The longitudinal control model had more fluctuation in the validation loss but managed to gain a better outcome

before overfitting. The best checkpoint in terms of validation loss is at epoch 35, with the loss in the range of e-04.

6.2 Testing Results

To evaluate the performance of the final resulting model with the unseen data(which has not been used in training), two methodologies are adopted, which are offline and online validation. The offline validation follows the same methodology as carried out during the training of the neural network. The VI-CarRealTime simulation data are gathered for the road tracks which have not been used in training, and the input features corresponding to the model are selected and provided as the input to the model to predict the target features. The output resulting from the model is compared to the relevant simulation VI-driver output. The term online validation refers to the testing of the model during the real-time simulation. The resulting NN models are used to provide the control input to the vehicle model during the co-simulation in the Simulink interface during each integration step.

6.2.1 Vehicle Control: Offline Prediction

Constant Velocity-Lateral Control Model

Before validating the performance of the model in the unseen tracks, the model is checked for its prediction result concerning some of the road tracks used during training and compared the result with corresponding VI-Driver output. To evaluate the prediction in the unseen track, firstly, the prediction results from simple manoeuvres are tested and then the full lap data are feed to the model to check the lateral control prediction for the same. Listed below are the unseen road tracks on which the model is validated

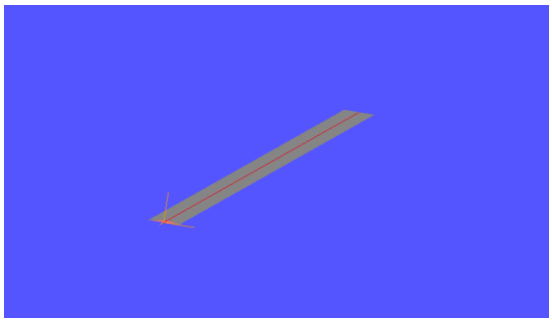


Figure 6.5.1 Unseen Track : Straight

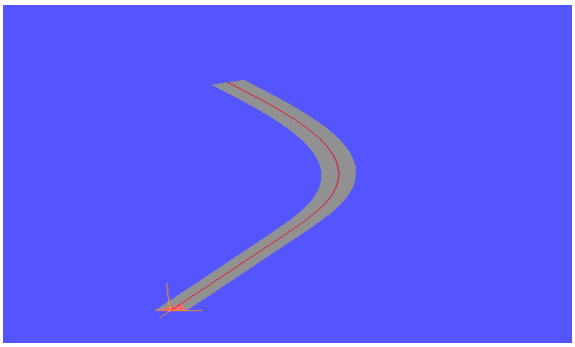


Figure 6.5.2 Unseen Track : Single Turn

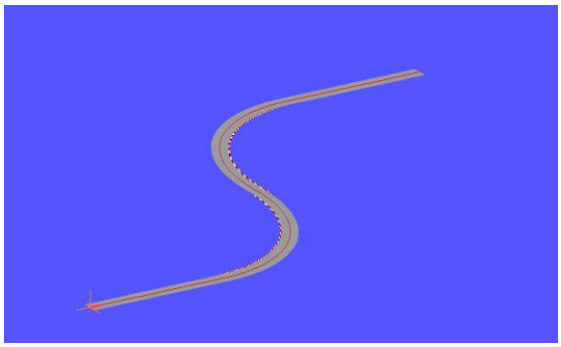


Figure 6.5.3 Unseen Track : Chicane

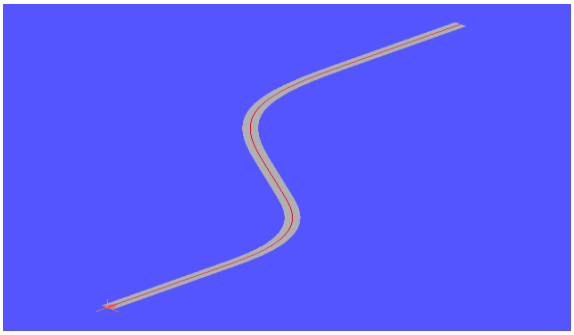


Figure 6.5.4 Unseen Track : Unknown Track 1

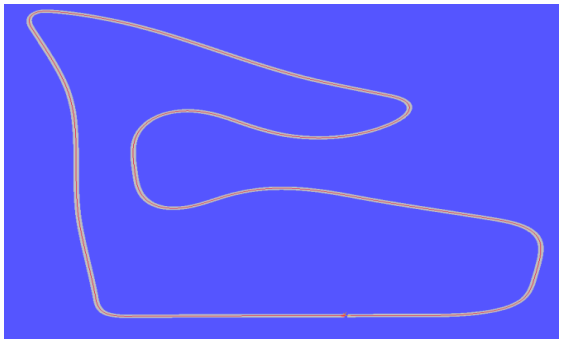


Figure 6.5.5 Unseen Track : Spielberg

Training Tracks Predictions

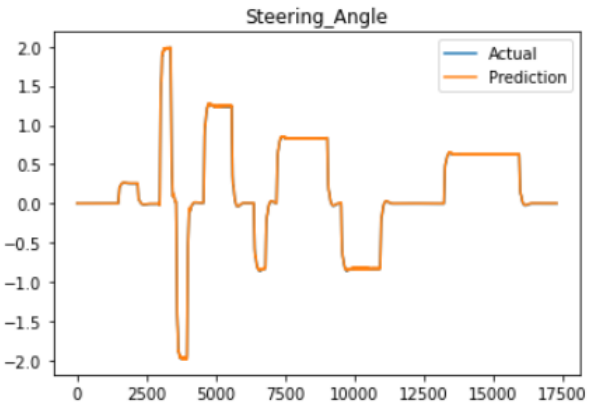


Figure 6.6.1 Prediction : VI-Track

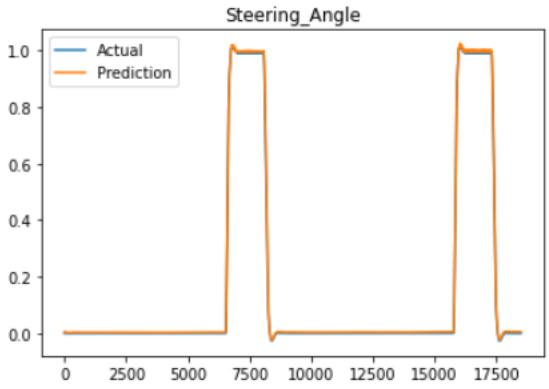


Figure 6.6.2 Prediction : VI-Road Example

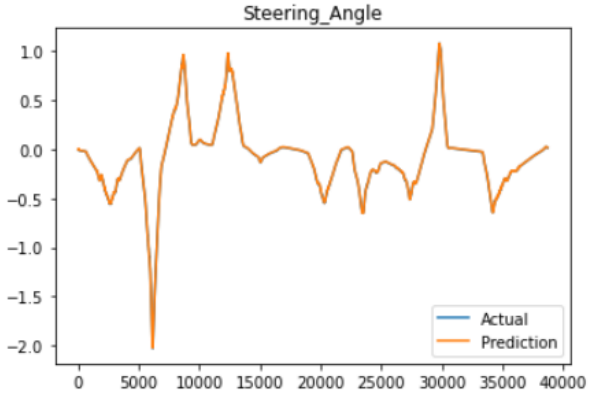


Figure 6.6.3 Prediction : Brand hatch

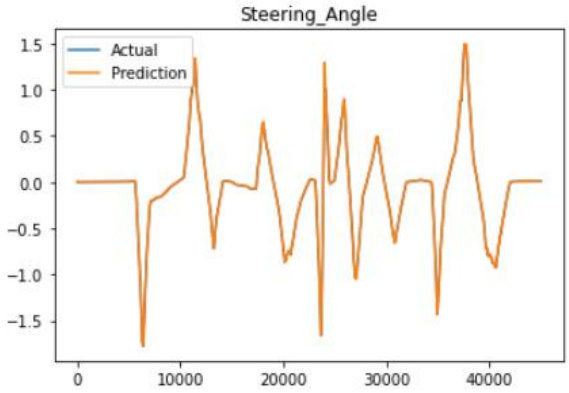


Figure 6.6.4 Prediction : Budapest

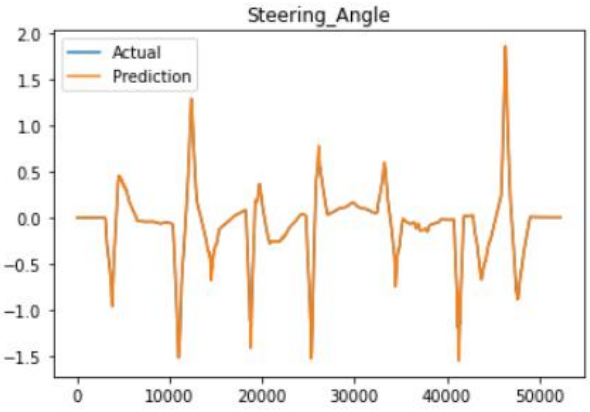


Figure 6.6.5 Prediction : Melbourne

The above figures show the NN models predictions in terms of lateral control for the five road tracks used for training. The actual values denote the VI-Driver steering angle, and the prediction represents the predicted steering angle from the NN model. It is quite evident from the plots that the prediction accurately coincides with the actual steering angle of the VI-Controller.

Unknown Tracks Predictions

To evaluate the prediction performance in the unseen track, the prediction results from simple manoeuvres such as straight, single turn and double turns are tested. Then, the complete lap data are fed to the model to check the lateral control prediction.

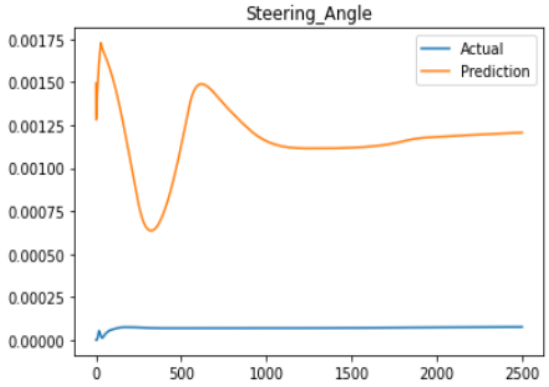


Figure 6.7.1 Unseen track prediction : Straight

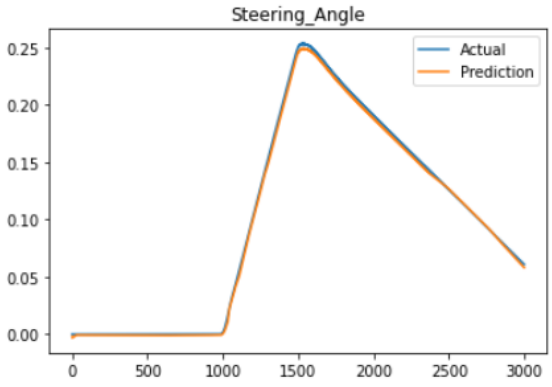


Figure 6.7.2 Unseen track prediction : Single Turn

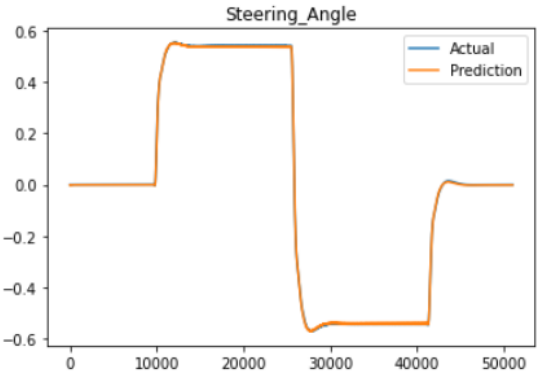


Figure 6.7.3 Unseen track prediction : Chicane

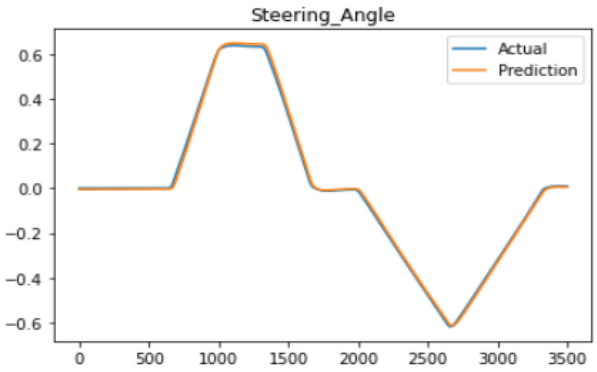


Figure 6.7.4 Unseen track prediction : Unknown 1

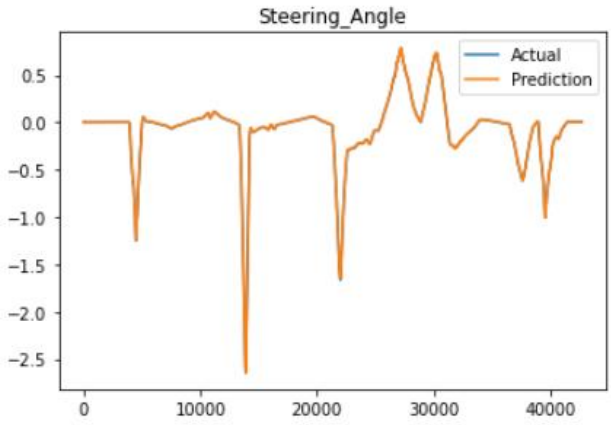


Figure 6.7.5 Unseen track prediction : Spielberg

Except for the straight manoeuvre, the steering angle prediction coming from the Neural Network model is almost the same as that of the lateral control provided by the VI-Driver Controller for the same unseen tracks. For the straight manoeuvre, the model predictions are still in the range of e-03, which is not far from the expected desired value to keep the vehicle straight.

MaxPerformance - Lateral Control Model

Similar to the constant velocity model, the built-in Neural Network model for the dynamic limit conditions are tested at first for its prediction with respect to its training tracks. Then the model generalization is evaluated by checking its predictions result in the unseen road tracks shown below.

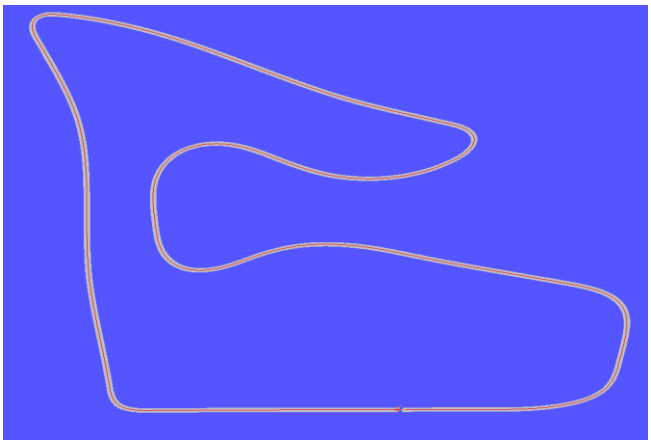


Figure 6.8.1 Unseen Track 1 : Spielberg

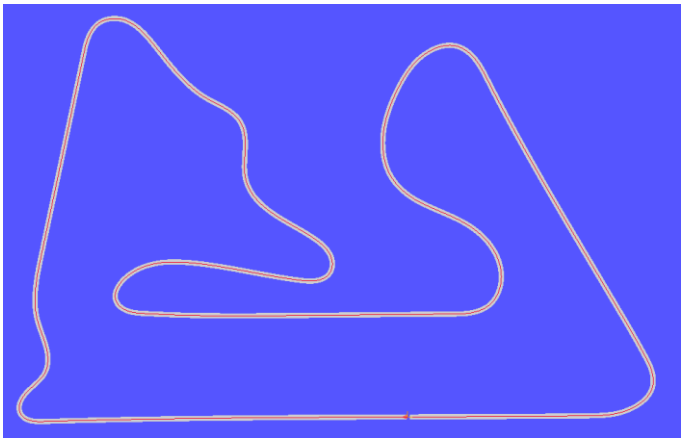


Figure 6.8.2 Unseen Track 2 : Sakhir

Training Tracks Predictions

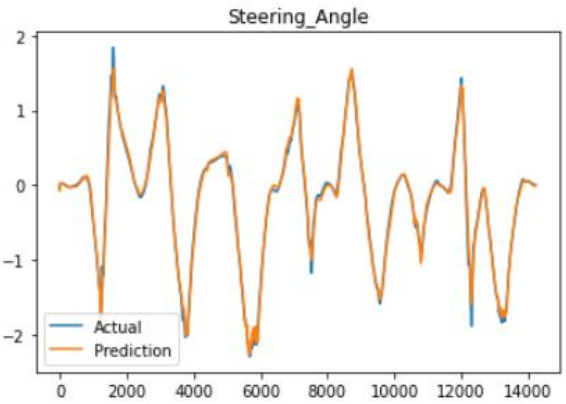


Figure 6.9.1 Prediction : VI RaceTrack

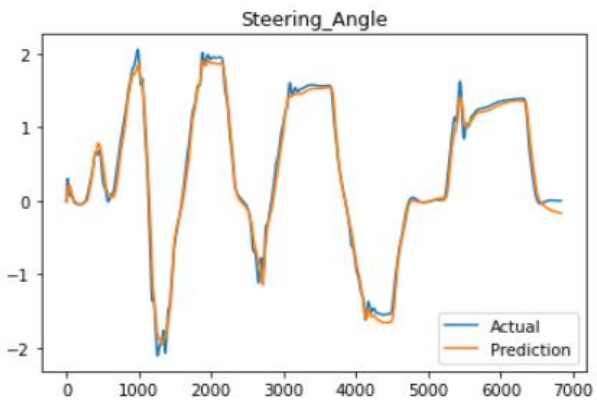


Figure 6.9.2 Prediction : VI-Track

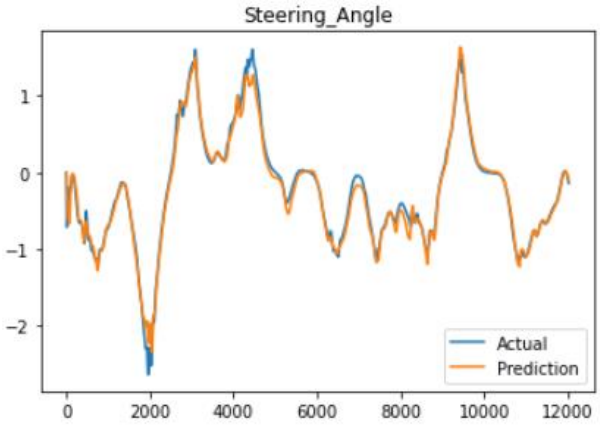


Figure 6.9.3 Prediction : Brand Hatch

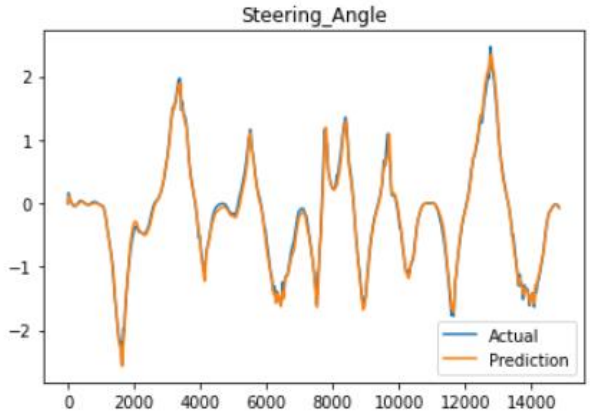


Figure 6.9.4 Prediction : Budapest

Unknown Tracks Predictions

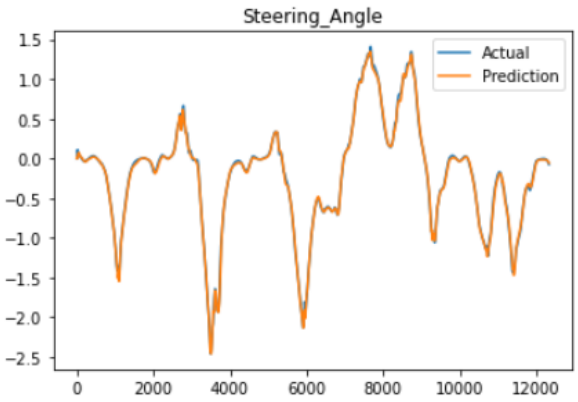


Figure 6.10.1 Unseen Track Prediction : Spielberg

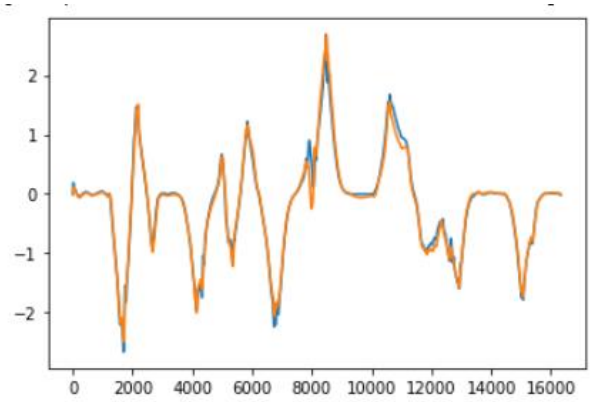
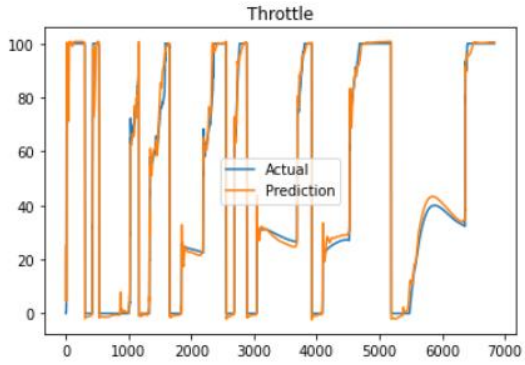


Figure 6.10.2 Unseen Track Prediction : Sakhir

MaxPerformance - Longitudinal Control Model

Training Tracks Predictions



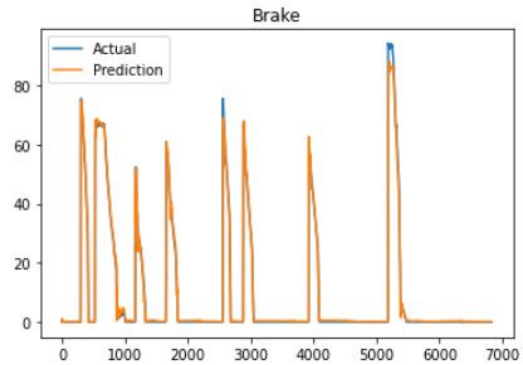
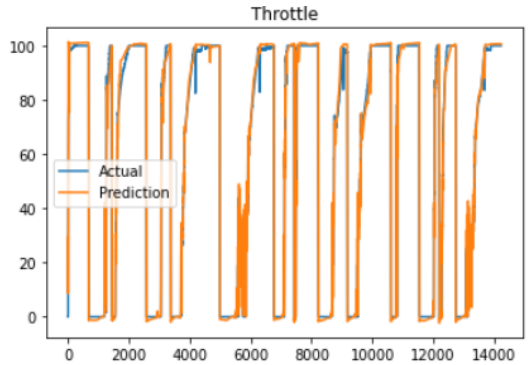


Figure 6.11.1 Prediction : VI-Track



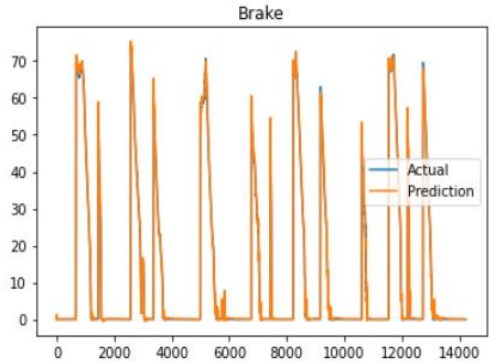


Figure 6.11.2 Prediction : VI-RaceTrack

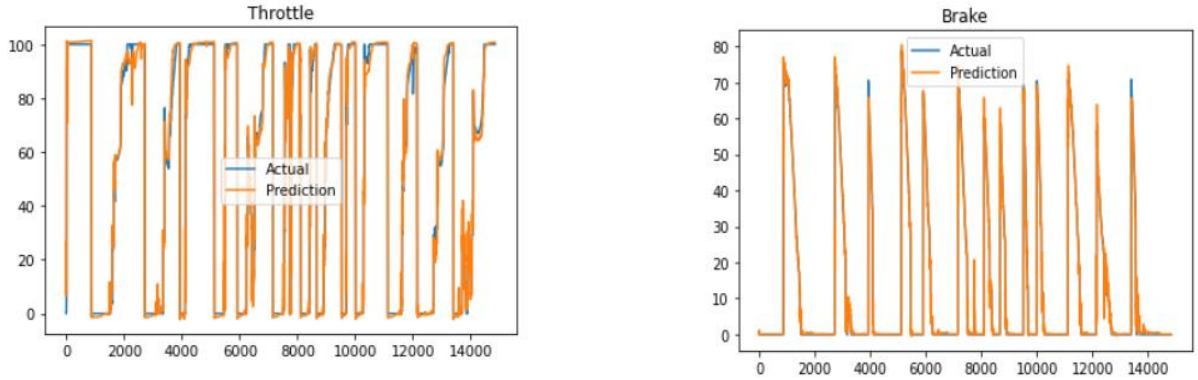


Figure 6.11.3 Prediction Brand hatch

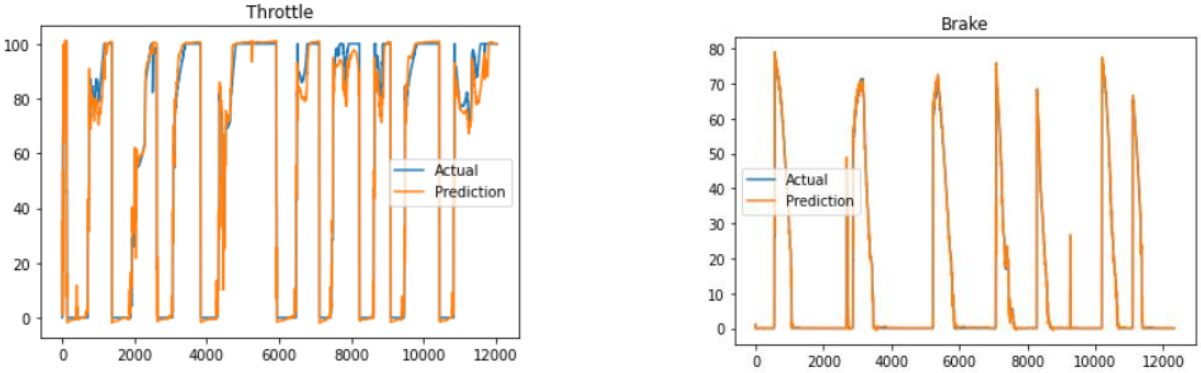


Figure 6.11.4 Prediction : Budapest

Unknown Tracks Predictions

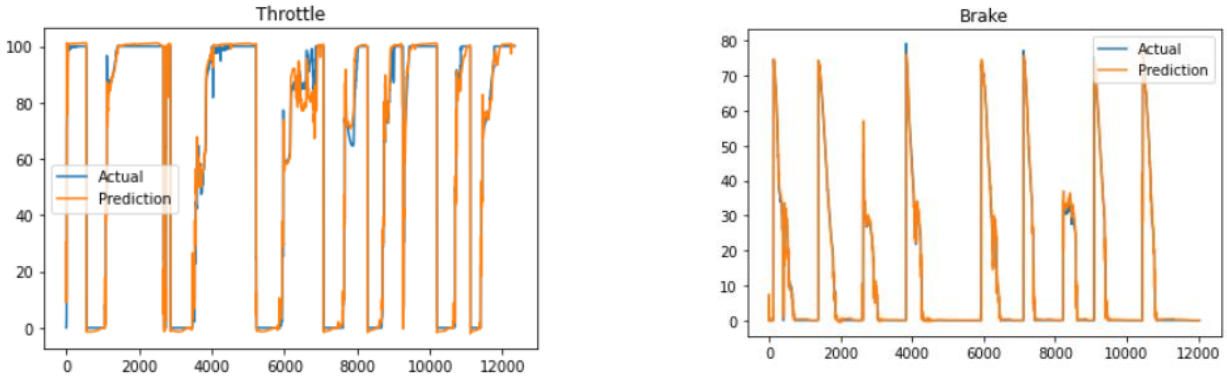


Figure 6.12.1 Unseen track Prediction : Spielberg

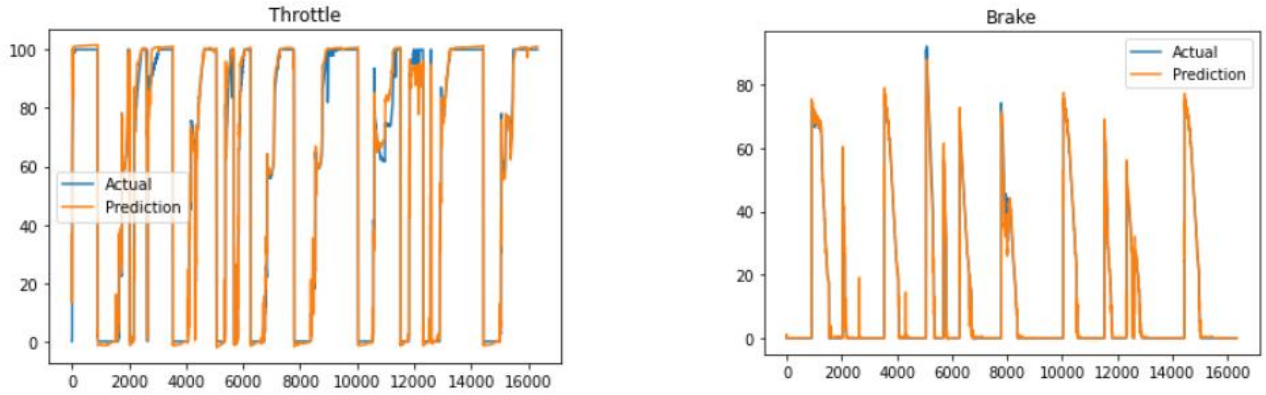


Figure 6.12.2 Unseen Track Prediction : Sakhir

The lateral and longitudinal control prediction for the MaxPerformance condition is depicted in the above plots. The prediction concerning the training tracks is quite the same as that of the VI-Driver output with little variation, only by a few margins. Similar to that of the training, the full lap prediction for the two unseen tracks, which are reported, shows that the model is generalized enough to make the prediction as same as that of the VI-Controller in case of dynamic limit events for any given road track.

6.2.2 Vehicle Control: Online Prediction

The online validation of the developed Neural Network model for vehicle control is performed in the Simulink environment using the co-simulation events discussed in section 4.4. The model built for the constant velocity condition for lateral control prediction is tested for its performance during the real-time simulation. To synchronise the Simulink co-simulation with the prediction output from the NN model created, the same solver settings (Integration steps 0.001 and the solver Runge-Kutta) are made in the Simulink and the VI-environment. But with this setup and the synchronization, when performed the co-simulation with the NN model to predict a steering input, there existed an offset in the prediction value getting into the vehicle model during the simulation. To be precise, with the state feedback value recorded at a time ‘t’ (which is provided as an input feature to the Neural Network), the NN model makes a steering angle prediction which is given to the vehicle model at a time ‘t+1’. To overcome this issue, the NN model structure is slightly

modified to predict the next step steering angle by taking the state feedbacks and the current step steering angle as an input. The modified NN model block is as below in figure 6.13

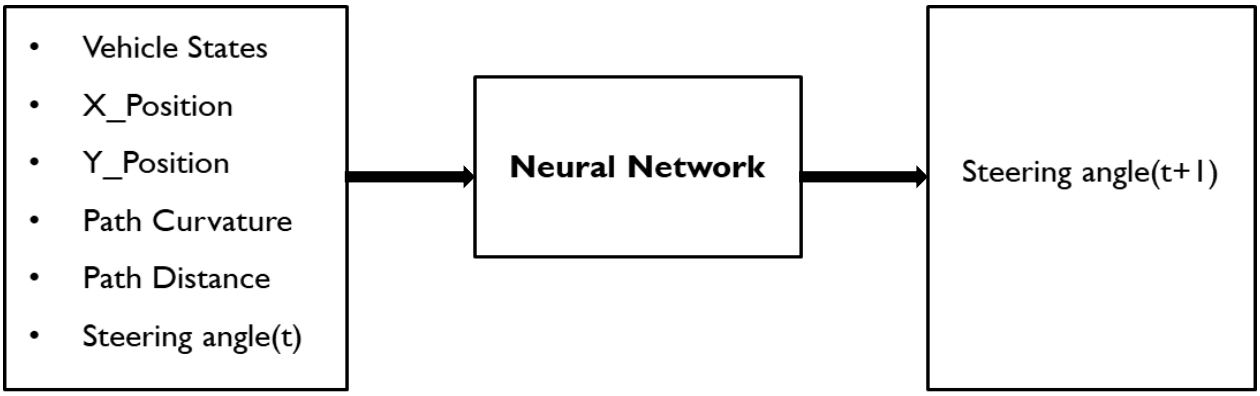


Figure 6.13 Modified NN layout

The Simulink space where the co-simulation is performed with the passive vehicle model on integration with the above modified Neural Network model is shown below in figure 6.14

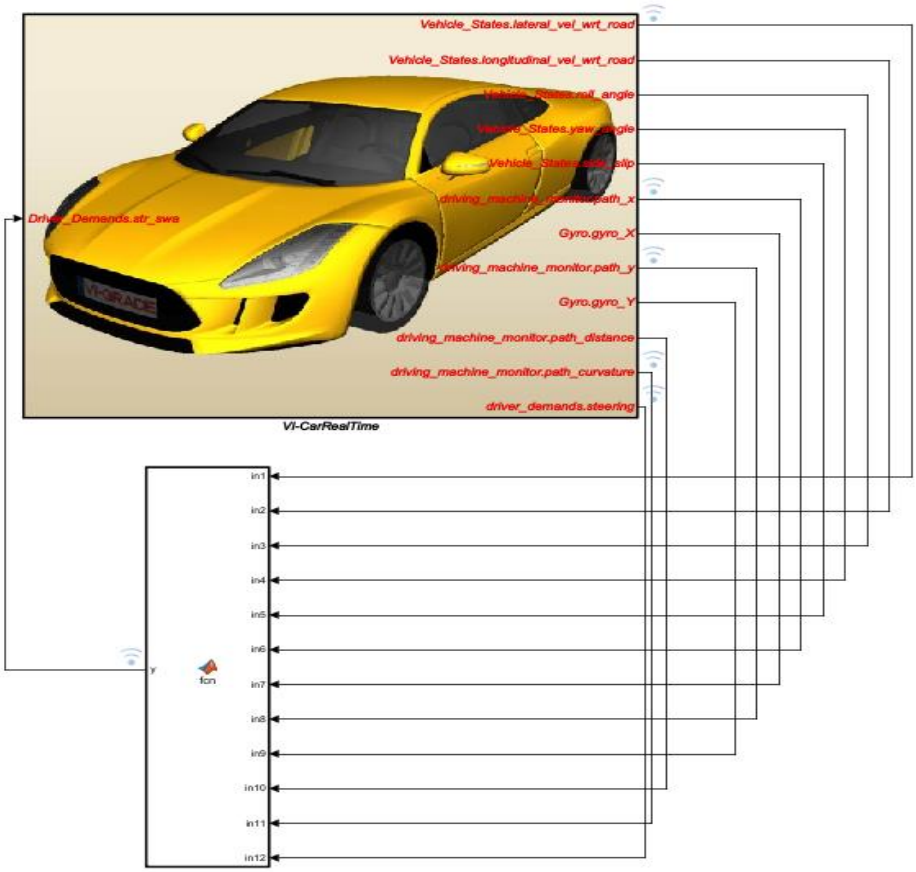


Figure 6.14 Simulink Environment : Online validation setup

With the above setup and the final NN model as per the architecture detailed above, the prediction coming from the NN model was able to get into the vehicle model successfully during the co-simulation in the Simulink. But its performance during this online validation is not the same as that of the offline prediction results. Upon investigating the issue, it is addressed that the scaling (normalization: pre-processing), which is performed before training the NN model, causing the abrupt change in the prediction coming from the Neural Network model. This is because the normalization procedure during the pre-processing scales the data value based on the maximum and minimum value limit available in the training dataset, which is used while building the model. During the real-time simulation, when the raw input data, which is the state feedback in our case, goes beyond the maximum or below the minimum limit range according to the training dataset, the NN model gets perplexed, and the prediction gets out of bound (i.e.) not as expected. As a general rule, the scaling of the input feature is necessary to achieve a good prediction result with the regression neural network. Still, it is not mandatory for all applications. Hence the model is again trained without the scaling procedure and found to be performing almost the same as that of the scaled model in the case of the offline prediction. The unscaled model, when performed the online validation shown slight improvement in the result, but in this case also it is not quite the same as that of the offline prediction results. The improvement is seen with this model in the case of the straight segments, where the model was able to make the prediction well enough to keep the vehicle in the straight path. When approaching the curves, the model could not predict the desired steering angle, which makes the vehicle not follow the desired trajectory. However, this is the case with the model without having the input feature path-distance as input to the model. This is also due to the same fact as discussed above in the case of the scaling issue. In the training dataset, the path distance value (denotes the lateral offset with respect to the reference trajectory) will have its maximum in the range of 10 e-02 since the VI-controller here provides the trajectory control to track the reference path as close as possible. Because of this, during the real-time simulation with the NN model, when the path distance input goes beyond its maximum value in the training dataset, the Neural Network was not able to provide the steering angel prediction to bring the vehicle to its reference path.

The below plot shows the offline prediction result of the NN model for the straight path and the result of the same model during the online validation (VI-CRT simulation output). In this case,

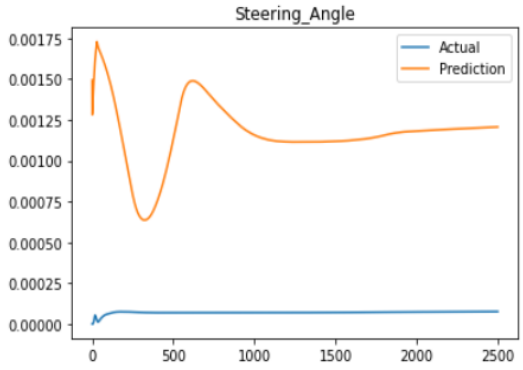


Figure 6.15 Offline Prediction : Straight Path

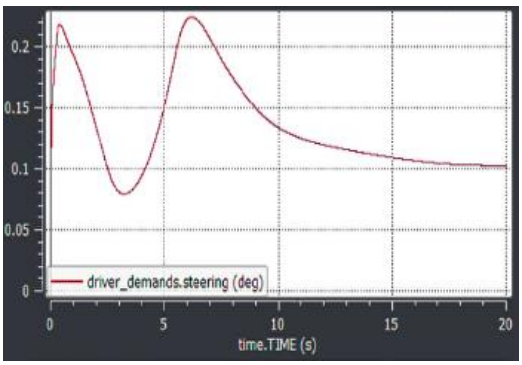


Figure 6.16 Online Prediction : Straight Path

offline prediction is the same as that of the steering angle prediction coming during the real-time co-simulation.

6.2.1 Vehicle Trajectory Prediction

The final Neural Network model in the case of the optimal trajectory prediction is evaluated by validating its prediction performance in the case of the unseen road tracks by providing the input of the road boundaries and the path curvature. Before feeding the input to the final model, the boundary coordinate's and the curvature is subjected to the data shift (sliding window) same as performed during the training procedure. The initial prediction by this procedure showed a noise in the trajectory line, which is reduced with the help of the change in the Neural Network Activation function.

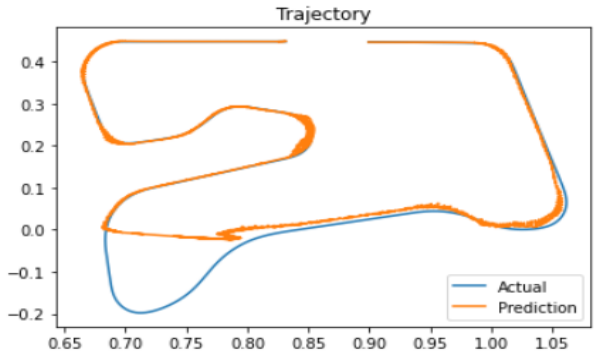


Figure 6.17 Noise level before change in Activation function

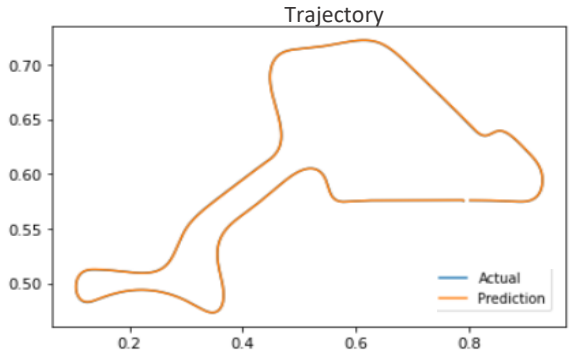


Figure 6.18 Noise level after change in Activation function

The final optimal trajectory prediction for the two unseen tracks, which is compared to the optimal trajectory line of those tracks as generated from the path builder script based on the corner-cutting value, is reported below.

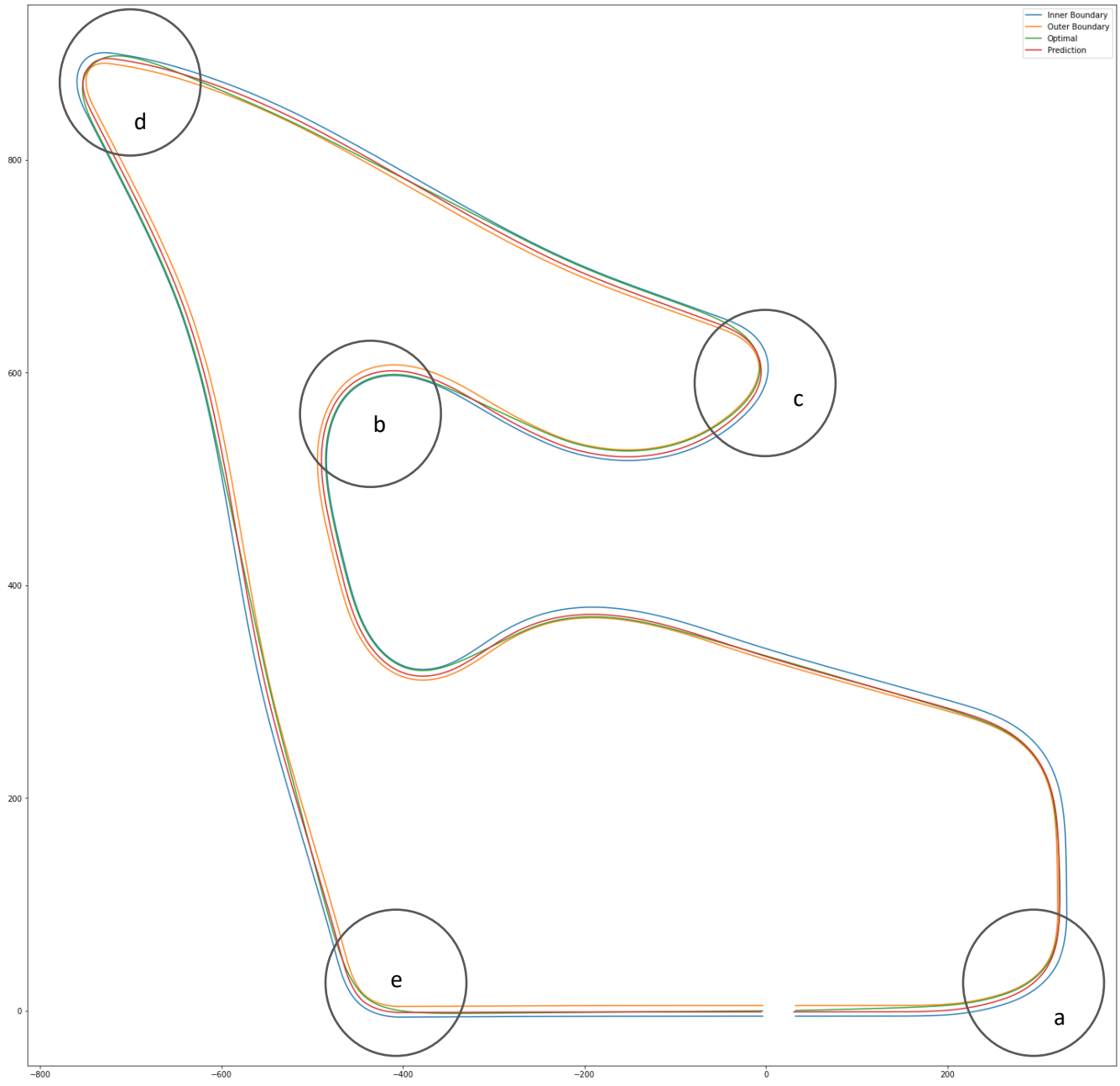
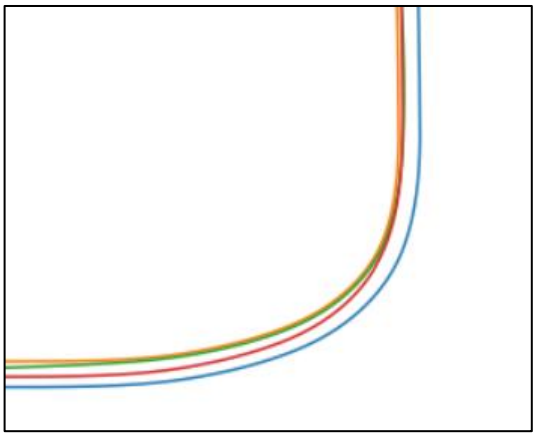
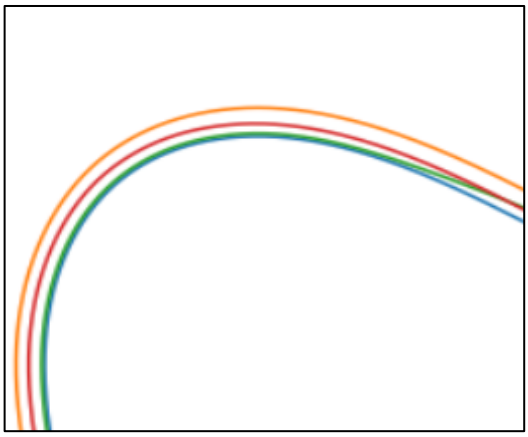


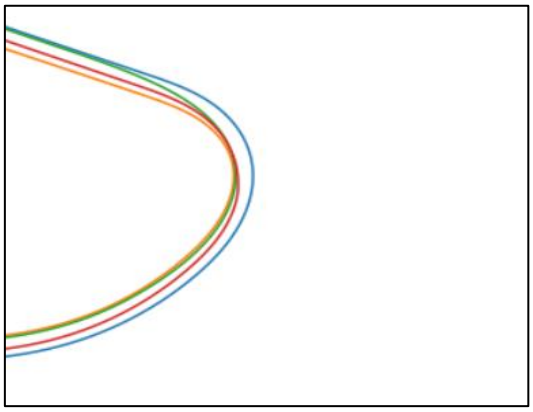
Figure 6.19 Unseen Trajectory Prediction : Spielberg



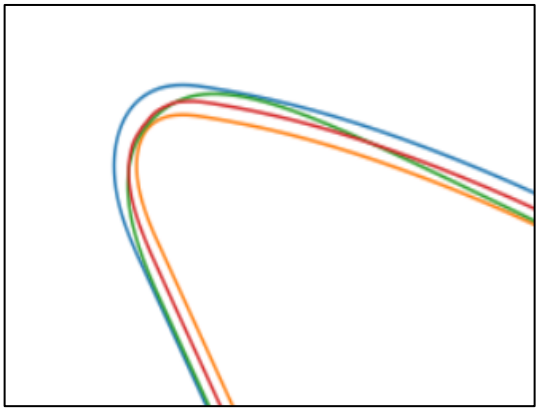
Curve a



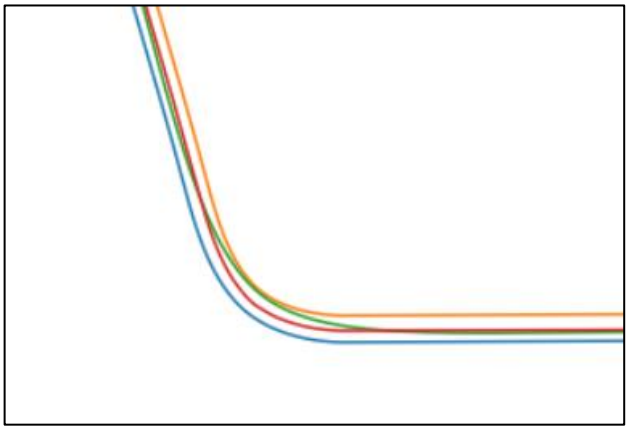
Curve b



Curve c



Curve d



Curve e

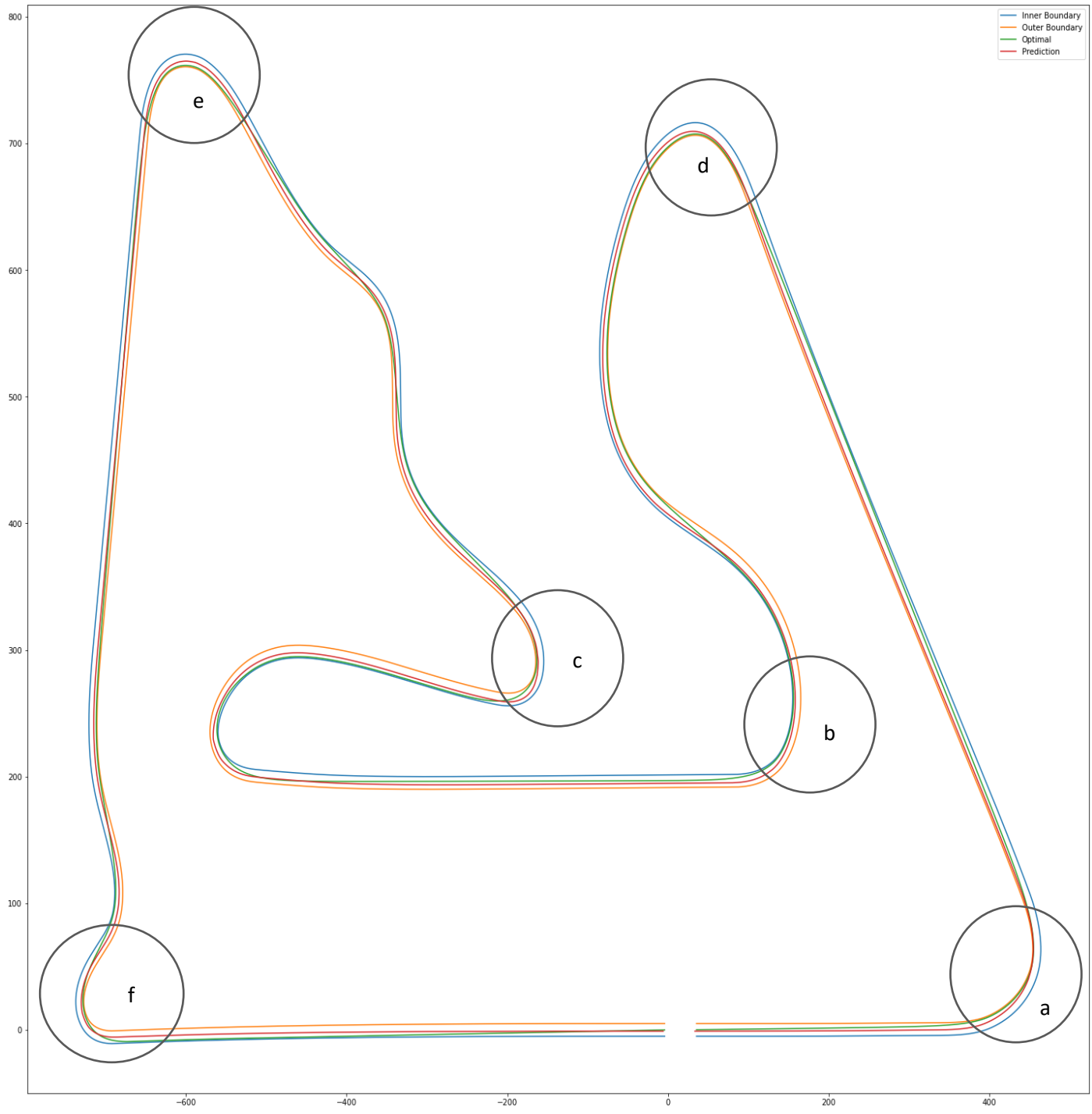
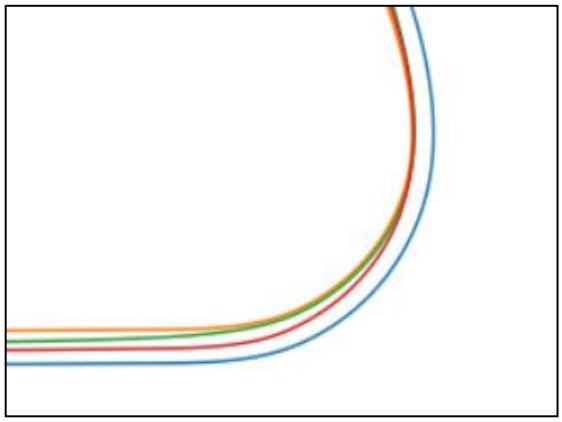
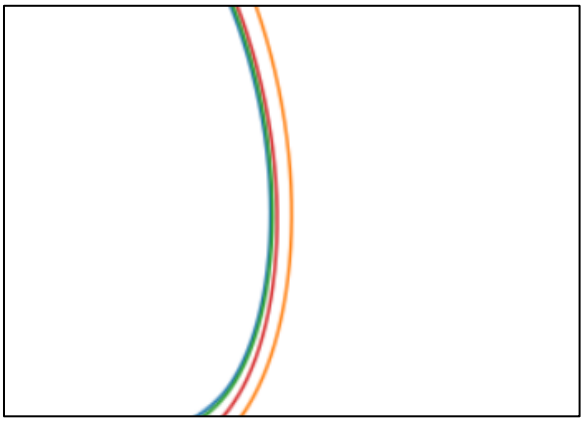


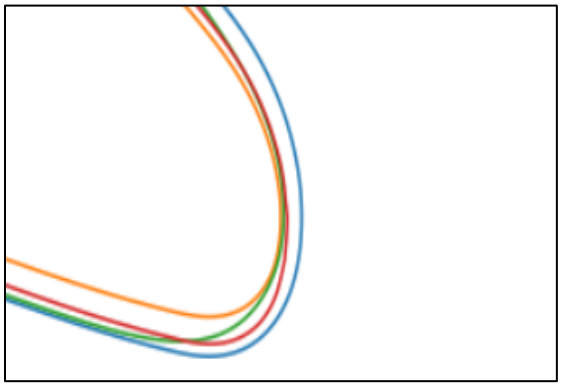
Figure 6.20 Unseen Trajectory Prediction : Sakhir



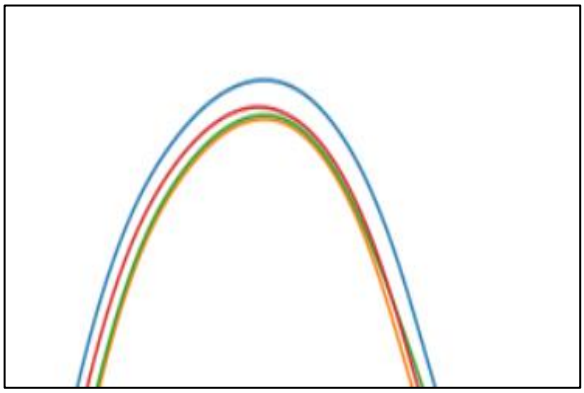
Curve a



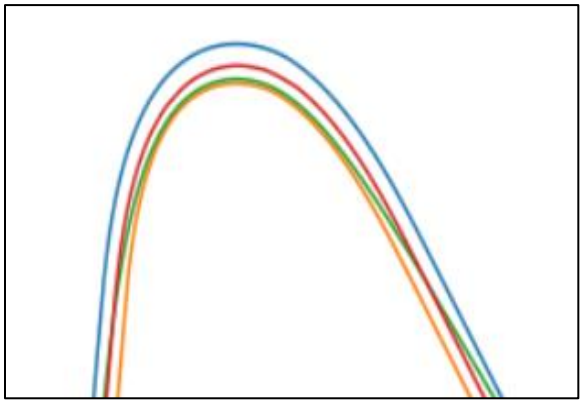
Curve b



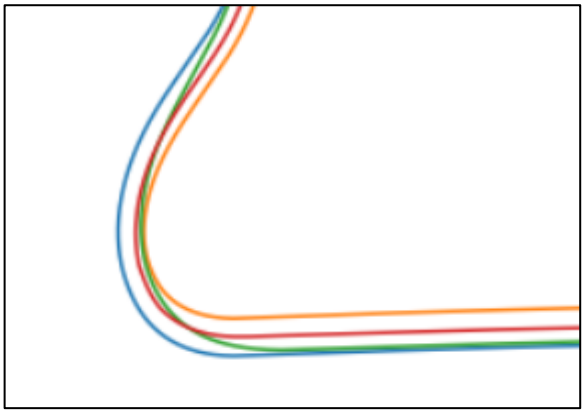
Curve c



Curve d



Curve e



Curve f

The above two plots show the prediction of the Neural Network model for the optimal trajectory for the two unseen tracks. Figure 6.19 and 6.20 shows the global optimal trajectory line predicted by the Neural Network when fed with the boundary and curvature details of the respective road tracks. For a better visualisation purpose, the inner and outer boundaries are plotted together, which is shown by the orange and blue lines. The predicted trajectory line is denoted as a red line, and the optimal trajectory (fastest trajectory for the track) corresponds to that track obtained by the path-builder script is represented by the green line.

The final NN model was able to make the trajectory prediction for any given track within the road boundaries, and in the corners, its prediction behaviour (which is depicted in the following figures of each track trajectory prediction) is almost the same as that of the optimal trajectory line of that road track. This predicted trajectory line is exported from the google colab, and it is used as a reference path in the VI-CarRealTime simulation, and the vehicle was able to follow the trajectory predicted from the neural network. But with this predicted path, there observed little amount of noise in the tight corners. In comparison to the optimal line generated by the path builder script, the prediction made by the Neural Network showed a slight increase in the lap-time, which is mainly due to the small amount of noise as described above in deep curves.

Chapter 7

Conclusion and Future Work

Autonomous vehicle development is a highly discussed subject in the present research world as well as within the industry. To realise it as a complete system in an on-road vehicle, several complexities in the system need to be worked together. In this thesis work, the current advancement in Machine learning and Neural Networks is evaluated as a realistic option to replace the existing traditional control method for the task of trajectory planning and control.

In many cases, the existing path planning and tracking methods based on the control strategies have shown unsatisfactory results. This made ANN a good option for such tasks as it is proven to imitate any mathematical function accurately. In this experimental thesis work, the performance of the Multi-Layer Perceptron Neural Network Model for the task of trajectory planning and control is conceptualized as a possible choice for the development of the autonomous vehicle system. In the first part of the work, the vehicle control model is validated with unseen data from the simple to the complex driving tracks. The offline prediction, which showed better performance results, indicates the ability of the Neural Network to learn the task of driving in the closed tracks without the information of visual information (i.e. no camera images are used as input data in the thesis work). It also shows that it can regularize well for the new unseen driving tracks, which confirm good generalization. The online validation setup points out the successful integration of the Neural Network model with the simulation environment, as the model was able to provide the steering command as input during the simulation. But the results are not robust enough in this case as it can make predictions only in the straight segment. The main reason for this could be the simplicity of the MLP model in general and also the training strategy utilized here, which is offline in our case. In addition, since the training data here are gathered from the VI-Driver controller, the data will already be in the regulated condition (i.e.) the data denoting the unsatisfactory behaviour is not captured in this case during the simulations, and because of this missing information, Neural Network model fails to perform the corrective measure when the driving condition goes far beyond the normal. With the hybrid/complex model schemes and by increasing the wide range of driving uncertainties in the training dataset, the Neural Network will be able to show better performance

in the case of the real-time simulation. In the second part of the thesis work, the trajectory prediction, the model is validated to predict the optimal global trajectory for the two complex road tracks. The validation results are promising, as the model developed can consider the road boundaries and provide the optimal trajectory line in between. Also, this indicates that this method could be a time-efficient one compared to other path planning strategies. Compared to the optimal trajectory line generated based on the path builder script, there is a slight degree of mismatch in the model predicted trajectory line. However, the prediction accuracy could be improved much with the same model by training with a large number of road tracks. To make the trajectory smooth as possible, the sampling rate could be increased (i.e.) the number of lines in each sliding window could be increased; this, in turn, increases the complexing of the NN model as the number of input and output features will be more.

This research work can be carried out to integrate the two separated NN models as a single complete system to be a good choice for the trajectory planning and control part.

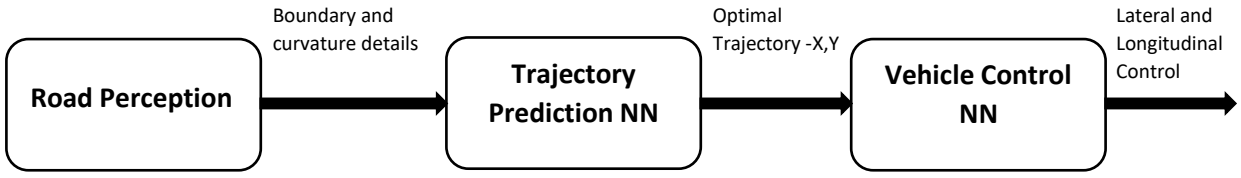


Figure 7.1 Complete System Layout

The above block shows the layout of the complete system where two Neural Network models could be combined and automated to provide its prediction in real-time. In addition, considering this as a promising starting point, certain improvements can be carried out in the future to make the model more robust enough to be implemented as an external driver model for the VI-CarRealTime as an additional option to the already existing driver model.

Bibliography

[1] G. T. a. J. K. S. Sharma, "Behavioral Cloning for Lateral Motion Control of Autonomous Vehicles Using Deep Learning," in *IEEE International Conference on Electro/Information Technology (EIT)*, 2018.

[2] G. T. a. J. K. S. Sharma, "Lateral and Longitudinal Motion Control of Autonomous Vehicles using Deep Learning," in *IEEE International Conference on Electro Information Technology (EIT)*, 2019.

[3] C.-H. W. a. Y.-R. C. Tsung-Ming Hsu, "End-to-End Deep Learning for Autonomous Longitudinal and Lateral Control based on Vehicle Dynamics," in *International Conference on Artificial Intelligence and Virtual Reality*, New York, NY, USA, 2018.

[4] L. J. Viktor Insgård, "Heavy vehicle path control with neural," Department of Mechanics and Maritime Sciences, CHALMERS UNIVERSITY OF TECHNOLOGY., Gothenburg, Sweden, 2018.

[5] A. B. D. M. P. O. J. H. P. &. B. P. Wasala, "Trajectory based lateral control: A Reinforcement Learning case study," *Engineering Applications of Artificial Intelligence,* 2020.

[6] G. P. P. A. F. &. M. F. Devineau, "Coupled Longitudinal and Lateral Control of a Vehicle using Deep Learning," in *21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[7] U. A. J. Deepak S. Gaikwad, "Classifying Trajectories on Road Network using Neural Network," *International Journal of Computer Applications© 2013 by IJCA Journal,* pp. Volume 81 - Number 13, 2013.

[8] S. &. M. N. &. H. K. &. R. C. Dabiri, "A deep convolutional neural network based approach for vehicle classification using large-scale GPS trajectory data," *Transportation Research Part C: Emerging Technologies.,* 2020.

[9] S. W. a. A. W. S. Mukherjee, "Interacting Vehicle Trajectory Prediction with Convolutional Recurrent Neural Networks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[10] S. &. B. A. Garlick, "Real-Time Optimal Trajectory Planning for Autonomous Vehicles and Lap Time Simulation Using Machine Learning".

[11] "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[12] J. Brownlee, "How to Configure the Learning Rate When Training Deep Learning Neural Networks," Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/.

[13] J. Brownlee, "Overfitting and Underfitting With Machine Learning Algorithms," Machine Learning Mastery. [Online].

[14] VI-Grade, VI-CarRealTime 20.2 Documentation.

[15] J. Brownlee, Deep Learning with Python, 2016.

[16] TUMFTM, *Racetrack-Database. https://github.com/TUMFTM/racetrack-database,* 2020..

[17] J. Brownlee, "How to use Data Scaling Improve Deep Learning Model Stability and Performance," [Online].

[18] MathWorks, *Curve intersections Algorithm.*

[19] I. G. a. Y. B. a. A. Courville, "Deep Learning," MIT Press.

[20] "MathWorks," [Online]. Available: https://it.mathworks.com/matlabcentral/fileexchange/22441-curve-intersections.

[21] J. Brownlee, Deep Learning for Time Series Forecasting, 2018.

[22] A. S. A. K. a. J. X. C. Chen, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," *IEEE International Conference on Computer Vision (ICCV),* pp. 2722-2730, 2015.

[23] A. N. Jaob Genander, "Control of Self-Driving Vehicles," Master's thesis in Computer Science - CHALMERS UNIVERSITY OF TECHNOLOGY, Gothenburg, Sweden, 2018.

[24] J. Dammen, "End-to-end deep learning for," Master's thesis in Computer science - Norwegian University of Science and Technology, 2019.

[25] S.-Y. O. a. Y. Yim, "Modeling of vehicle dynamics from real vehicle measurements using a neural network with two-stage hybrid learning for accurate long-term prediction," *IEEE International Symposium on Computational Intelligence in Robotics and Automation,* pp. 83-88, 1999 .

[26] X. H. C. L. Y. L. J. W. Xuewu Ji, "Adaptive-neural-network-based robust lateral motion control for autonomous vehicle at driving limits," *Control Engineering Practice,* vol. Volume 76, pp. 41-53, 2018.

[27] D. L. P. L. L. a. A. P. B. L. Cardamone, "Searching for the optimal racing line using genetic algorithms," *IEEE Conference on Computational Intelligence and Games,,* pp. 388-394, 2010.

[28] R. J. Alexander Bukk, "Vehicle trajectory prediction using," Master's thesis in Complex Adaptive Systems - CHALMERS UNIVERSITY OF TECHNOLOGY, 2020.

[29] J. W. J. M. D. H. Z. a. H. Z. Wenda Xu, "A real-time motion planner with trajectory optimization for autonomous vehicles," *IEEE International Conference on Robotics and Automation,* pp. 2061-2067, 2012.