An Approach To Optimal Trajectory With Neural Networks

Inês Moreira

UMinho | 2023

Universidade do Minho
Escola de Ciências

Inês Mafalda André Ramos Moreira

An Approach To Optimal Trajectory With Neural Networks

Dezembro 2023

Universidade do Minho
Escola de Ciências

Inês Mafalda André Ramos Moreira

An Approach To Optimal Trajectory With Neural Networks

Dissertação de Mestrado
Mestrado em Matemática e Computação
Especialização em Computação

Trabalho efetuado sob a orientação do(a)
Professora Flora José da Rocha Ferreira e da
Professora Sandra Maria Gonçalves de Vilas Boas Jardim

Dezembro 2023

Acknowledgements

There are several people who helped me create this work, to whom I owe my deepest thanks:

Firstly, I would like to thank Project BREUCA for providing me with the opportunity to work on this study. To all my professors, especially Professor Patricio, I want to express my appreciation for all of their assistance throughout the course of this year. I also want to thank my colleagues for the companionship they demonstrated.

Secondly, I would like to express my deepest gratitude to Professor Flora Ferreira for her continuous support, care, and patience with me. None of this work would have been possible without her. Additionally, I would like to thank Professor Sandra Jardim for the meticulous review of this dissertation.

To my friends, especially Filipa and Kseniya, for their unwavering friendship and love. To all the people who, in various ways, made Braga a home for me for six years.

To my family, who always showed so much care. Especially, my heartfelt gratitude to my sister, Catarina, and cousin, Joana, for their continuous support. You make me feel like I am the lucky one.

Lastly, this dissertation is dedicated to my dad, who is an example of perseverance and hard work.

“Long live the walls we crashed through
I had the time of my life with you”

Statement of Integrity

Abstract

The discovery of the optimal trajectory has consistently captivated the interest of the scientific community. With the development of Machine Learning and Deep Learning techniques, solutions have been developed to address this challenge, particularly through the use of Artificial Neural Networks.

This dissertation focuses on creating a Feed Forward Neural Network to predict the optimal path for Formula 1 and Deutsche Tourenwagen Masters (DTM) tracks. Initially, 76 circuits were generated, augmented to a total of 5533 tracks through the application of Data Augmentation techniques, and then fed into the traditional high-accuracy Optimal Control Problem model. Afterwards, data transformation was applied and preprocessed.

Numerous experiments were performed to develop a predictive model that is both simple and effective, with a particular focus on varying hidden layers and neurons. During the model training process, observations were made regarding the values of loss, mean squared error (MSE), mean absolute error (MAE), and root mean squared error (RMSE). This analysis led to the determination that a model with a single hidden layer containing 40 neurons produced the best results, yielding an MSE of 0,048. Two final experiments were performed by increasing the number of epochs to 100 and for two different foresight values: 30 and 100. With these modified models, we generated predictions for three tracks, each varying in complexity. The results indicated that for simpler tracks a lower foresight value suffices, resulting in an MSE of 0,076. However, for more complex tracks, it leads to a higher error. On the other hand, a more complex model will result in a higher error on circuits with fewer bends and a lower error on more complex tracks.

Keywords: Optimal Trajectory, Machine Learning, Deep Learning, Artificial Neural Networks.

Resumo

A descoberta da trajetória ótima tem consistentemente cativado o interesse da comunidade científica. Com a evolução de técnicas de *Machine Learning* e *Deep Learning*, novas soluções foram desenvolvidas para enfrentar este desafio, particularmente através do uso de Redes Neurais Artificiais.

Este estudo tem como objetivo criar uma Rede Neural Feed Forward capaz de prever o caminho ótimo das pistas de Formula 1 e Deutsche Tourenwagen Masters (DTM). Inicialmente, foram gerados 76 circuitos, aumentados para um total de 5533 pistas através de técnicas de *Data Augmentation* e depois inseridos num modelo tradicional, o Problema de Controlo Ótimo (OCP). Depois *data transformation* foi aplicada e os dados foram pre-processados.

Foram realizadas várias experiências com o objetivo de criar uma estrutura simples e altamente preditiva dando especial atenção à variação do número de camadas escondidas e neurónios. Durante o processo de treino, foram feitas observações relativamente aos valores de perda, erro médio quadrático (MSE), erro médio absoluto (MAE) e raiz do erro médio quadrático (RMSE). Essa análise levou à constatação de que um modelo com uma única camada escondida contendo 40 neurónios produziu os melhores resultados, resultando em um MSE de 0,048. Dois testes finais foram realizados aumentando o número de épocas para 100 e para dois valores diferentes de *foresight*: 30 e 100. Com estes modelos modificados, foram feitas previsões para três pistas, cada uma com níveis de complexidade distintos. Os resultados indicaram que para pistas mais simples, um menor *foresight* é suficiente, resultando em um MSE de 0,076. No entanto, para pistas mais complexas, isso resulta em um erro maior. Por outro lado, um modelo mais complexo resultará num erro maior em circuitos com menos curvas e a um erro menor em pistas mais complexas.

Palavras-chave: Trajetória Ótima, Machine Learning, Deep Learning, Redes Neuronais Artificiais.

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence.
ANNs	Artificial Neural Networks.
DL	Deep Learning.
DTM	Deutsche Tourenwagen Masters.
MAE	Mean Absolute Error.
ML	Machine Learning.
MLTT	Minimum Lap Time Trajectory.
MSE	Mean Squared Error.
OCP	Optimal Control Problem.
QSS	Quasi-Steady-State.
RMSE	Root Mean Squared Error.
TORCS	The Open Racing Car Simulator.

Chapter 1

Introduction

1.1 Contextualization and Motivation

This dissertation is part of the BREUCA project, which encompasses data analysis, data processing, model creation, and the development of models to discover the optimal trajectory of vehicles for a chosen track using Machine Learning/Deep Learning techniques.

The **Minimum Lap Time Trajectory** (**MLTT**) is a central problem associated with the generation of a high-precision virtual reality simulator in a game environment. Its goal is to enable simulator users to engage in real-time races against virtual and real pilots on various tracks. For a designated circuit, our aim is to find the optimal parameters and trajectory, providing a solution to the **MLTT**.

One of the primary concerns for the creators of car racing games is to discover the best trajectory for a specific vehicle on a particular track. The majority of business-oriented solutions generally occur through empirical methods. However, Colin McRae Rally (Codemasters) opted for a neural network trained by experts in the field of study. One of the benefits of Artificial Intelligence and Machine Learning techniques is the ability to provide a fast result compared to traditional methods, such as **Optimal Control Problem** (**OCP**).

Through the use of **Artificial Neural Networks** (**ANNs**), finding the optimal trajectory can be 9000 times faster compared to the performance obtained by the use of algorithms of optimal control.

Trying to learn an optimal path has always been an interest of the car racing community, and it dates back to the 1950s. Initially, technology had not yet reached an advanced stage of development, so traditional methods were the only option. The most common approaches are **OCP** and **Quasi-Steady-State** (**QSS**) [2]. Although the OCP is less time-consuming than the QSS method, it's still not enough to use in real-time [18].

In response to these challenges and driven by advancements in Artificial Intelligence, crucial Machine Learning/Deep Learning techniques have emerged. One of their primary advantages lies in their ability to make highly accurate predictions. Therefore, in addressing the issue of finding the optimal racing path on a given track, Machine Learning/Deep Learning methods played a vital role. It instantly became an alternative to traditional methods. Not only can it provide the prediction of the optimal trajectory, but even velocity, tuning of the car chassis and tracking of vehicles underwater [3, 21, 36].

1.2 Objectives

The primary objective of this dissertation is to leverage and investigate the application of **ANNs** to predict the optimal racing line. The core concept revolves around developing an ANN architecture that is not only straightforward but also capable of swiftly and accurately predicting the racing line.

To achieve this goal, the focus is on simplifying the ANN architecture while ensuring its predictive accuracy remains at a high level. The key challenge lies in identifying the most efficient structure that can rapidly generate precise racing line predictions.

In pursuit of this objective, the research delves into two crucial aspects:

1. Preparation of the Dataset: A critical and integral part of this study is the meticulous preparation of the dataset. This process revolves around the cultivation and refinement of the database that will subsequently serve as input for the Machine Learning model. The employed methodologies are meticulously tailored to process the dataset, thereby ensuring its optimal suitability for utilization as input for the model

2. Enhancing the Performance of the **ANNs** through Hyperparameter Fine-Tuning: To optimize the performance of the ANN, a spectrum of hyperparameters is investigated. These include determining the optimal number of hidden layers, configuring neurons within these layers, selecting appropriate activation functions, defining batch size, tuning the learning rate, and employing regularization

techniques. The aim is to delicately balance model complexity with predictive accuracy, ultimately crafting an ANN that excels in delivering precise predictions while maintaining a manageable and interpretable architecture.

By addressing these considerations, this dissertation aspires to develop an ANN-based solution that excels in rapidly and accurately predicting the racing line, contributing to advancements in the field of racing line prediction using machine learning techniques.

1.3 Document Organization

The content of this dissertation is organized as follows:

Chapter 2 addresses the state of the art of real-time racing trajectory techniques and it's divided as follows:

- The first section gives a broad scope of traditional methods used to predict the optimal trajectory, as well as a brief history of the topic.
- The second offers a historical overview of machine learning and neural network development, covering significant milestones and challenges. It also introduces key concepts related to neural networks, learning and various learning types.
- The third provides the reader with the knowledge of how the prediction of the optimal racing has evolved, emphasizing on Machine Learning approaches with ANNs.

Chapter 3 focus on a detailed overview over the process and processing of the data that, later on, will serve as input for the ANN. This chapter is structured as follows:

- In the opening section, we delve into the characteristics of the utilized data, offering a detailed explanation of the data extraction methodology that lays the foundation for its subsequent handling and incorporation within the neural network.
- The subsequent section provides a thorough examination of the post-extraction treatment and refinement of the data, a pivotal step to ensure the quality of the data is high enough to obtain a model

with good precision. This encompasses procedures that have been meticulously employed to enhance the data's suitability and compatibility for effective integration within the neural network's architecture.

Chapter 4 offers a comprehensive overview of the methodologies used in this dissertation. It includes a detailed exposition of ANNs, with an emphasis on Feed Forward Neural Networks.

Chapter 5 presents the outcomes of the testing performed in the aforementioned model. A discussion is also made of the results accomplished.

Chapter 6 gives an overview of the work developed, presents the main conclusions, and provides suggestions for future work.

Chapter 2

State of the Art

This dissertation addresses approaches to the optimal trajectory with the use of Neural Networks. Therefore, it is crucial to understand the advances made in this field of study until now. In a first moment, we will analyse the traditional methods, giving clarity on their contributions. Furthermore, it will be given a overview on Machine Learning history, detailing key concepts and, later on, explaining the evolution of methods within this field.

2.1 Evolution and Challenges in Real-Time Racing Trajectory Techniques

Over the past several decades, **MLTT** or Optimal Trajectory in Minimum Time for a given vehicle and track has drawn significant attention from researchers in control optimization and vehicle dynamics. The car racing community has taken a keen interest in this area, spurred by advancements in computation and numerical optimization [17]. As a result, the applications for the optimal lap have been immensely used for the improvement of car racing performance.

The earliest attempts to address this challenge dates back to the 1950s. Solutions to this problem can broadly be classified in two main categories: **QSS** and **OCP**. The **QSS** begins with the requirement of prior track information as input. Subsequently, this track information is subdivided into smaller segments. Within these segments, the vehicle is assumed to remain in a stationary state, with the exception of a few state variables, including speed, as it follows the racing line. At each corner apex, the system calculates the vehicle's maximum velocity. The zones for braking and acceleration are reconstructed through forward and backward integration. In the 1980s, a part of a Formula One track was simulated using the **QSS** method [2]. On the other hand, dynamic models that allow for unconstrained trajectories typically need the resolution of a nonlinear optimal control problem (**OCP**), which can be tackled through direct or indirect methodologies [34]. The **OCP** can be hard to solve depending on the complexity of the optimization problem. Therefore, most of the time, the car model is simplified [1].

Furthermore, even though these two methods have a significant amount of resolution power, they are heavy computationally due to the requirement of solving differential equations through direct/indirect methods (**OCP**) and backward/forward integration at each apex (**QSS**). In the experiment conducted by Lenzo and Rossi, the **QSS** method took 16 minutes to simulate the Barcelona circuit. Although it's faster than the **OCP**, which took approximately 39.8 hours, it's not fast enough to be used in real time [18].

Optimal trajectory techniques have undoubtedly evolved, with some approaches even combining both **OCP** and **QSS** simultaneously [34]. However, despite the rapid computation capabilities of the Quasi-Steady-State method, even when applied to complex models [2], it has yet to achieve the real-time speeds necessary for practical applications. For more details on these methods see references [2] and [34].

2.2 Introduction to Machine Learning and Artificial Neural Networks

Machine Learning (**ML**) can be defined as "the field of study that gives computers the ability to learn without being explicitly programmed" [13]. ML is a subfield of **Artificial Intelligence** (**AI**), which is the "study of any device that perceives its environment and takes actions that maximize its chance of success at some goal" [25]. Furthermore, within the field of ML, we have **Deep Learning** (**DL**), which is characterized by "the study of Artificial Neural Networks and related Machine Learning algorithms that contain more than one hidden layer" [25].

ML began in 1943 when Warren McCulloch and Walter Pitts presented the first mathematical model of **Neural Networks**. In 1949, Donald Hebb published a book called *"The organization of behaviour,"* where he stated a physiological learning rule for synaptic modification. As a result, several developments were made, including the first attempt to use a computer simulation to test a well-formulated neural theory. In 1958, a new approach to pattern recognition was presented by Rosenblatt - **the Perceptron** [14, 30].

For years, neural networks were considered limitless. However, in 1969, Marvin Minsky and Seymour Papert presented the limitations of the single-layer perceptron mathematically. They also stated that those limitations could not be surpassed by a MultiLayer Perceptron. This created what is known today as a period of dormancy in this field of study, lasting almost a decade [14].

Those limitations were eventually overcome, leading to major developments that have brought us the technology available today.

Neural Networks can also be defined as *"a machine designed to model the environment in which the brain performs a particular task or function."* They involve an extensive interconnection of processing units, namely neurons, which are a key component of any neural network [14].

Learning, referred to as *"a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded"* is arguably one of the most critical attributes of a neural network [14]. Without it, improving performance or achieving the required task becomes challenging. The ML model learns through an interactive process and hyperparameter Fine-Tunning (e.g., the number of neurons per layer, weights, bias, etc.). There are several types of learning:

Supervised Learning: The input data has labels and has a straightforward association with an output. The accuracy of the model can be calculated through the difference between the true and predicted values [14].

Unsupervised Learning: It is characterized by the input data having no labels. Once the network becomes tuned to the statistical regularities of the input data, it gains the capacity to form representations for encoding features of the input and thereby create new classes automatically. Some examples of unsupervised learning are: K-Means, One-class Support Vector Machine (SVM) and Principal Component Analysis (PCA) [13, 14].

Semi-Supervised Learning: It is a mixture of supervised and unsupervised learning, i.e., the dataset is composed of labeled and unlabeled data [14]. This type of learning can find applications in classification, regression, and predictive tasks [25].

Reinforcement Learning: It is an input-output mapping performed through a loop. It teaches itself the best strategy with the desire to minimize the cost function [13]. Additionally, it is characterized by three key elements: the agent, the environment, and actions [25].

These different types of learning form the foundation of machine learning, and each has its own set of techniques, algorithms, and applications. The choice of learning approach depends on the specific problem and the nature of the available data.

2.3 Machine Learning in Racing Trajectory Solutions

The usage of ML/DL methods has skyrocketed due to the ability of making predictions. Therefore, as an alternative to traditional methods, ML techniques appeared with the aim to solve the optimal trajectory problem. One of the most well known approaches for car racing is in game environment. In 2010, for the Simulated Car Racing Championship, a model was built with the help of the **TORCS** simulator who pre-processes tracks and, afterwards, with the help of neural networks, makes a prediction for the optimal trajectory and velocity. The networks were fed with data retrieved from sensors. Although the results were sufficient and the prediction obtained was similar to what a pilot does in real time, it's still 20% slower [21]. In 2012, a heuristic control using **ANNs** was developed, with the **ANNs** determining optimal parameters based on sensor input data stored in the **TORCS** simulator. Other techniques have appeared using this simulator [5, 28]. Besides **TORCS**, other simulators have been helpful for optimization of car racing, for example: CarSim and Gran Turismo Sport [6, 9].

ML/DL techniques have been developed for applications beyond optimal trajectory, including drift situations, tuning of a car chassis, tracking of vehicles trajectories underwater or even calculation of the Side-Slip angle [3, 4, 6, 29, 36]. In 2019, a neural network underwent training using 45 million video frames obtained from a Tesla vehicle. Nevertheless, the primary objective focused on ensuring safe navigation rather than discovering the optimal path [8].

With that in mind, Garlick and Bradley [10] presented an approach where the input data for the Neural Network consists of the inner and outer boundaries of race tracks and the respective reference line. From 82 real circuits (60 manually generated and 22 taken directly from TUMFTM [33]), posteriorly augmented by geometric operations (flipping, reverse of direction and scaling) they were able to reach 6058 tracks. The post-augmented track are applied in the **OCP** where the output is the optimal track. The **OCP** can not be used in real time due to the computational expense [2]. However, a neural network can be trained with the data from the **OCP**. As **ANNs** are computationally faster, a Feed Forward Neural Network was employed. The sliding window approach was used to divide each circuit into segments, resulting in a dataset comprising 2.7 million track segments. The ANN was trained with this dataset, and it's predictions were compared to the actual racing lines. The authors highlight the flexibility of the **ANNs**, which can accept training data from any existing method of generating racing line [10]. This means that the **ANNs** can be trained to 'drive like a particular driver (or simulator) in a particular car' and offers a robust solution for a wide range of applications. The results showcase an average mean absolute errors of ±0.27m for the predicted racing lines, with even higher accuracy at the corner apex (±0.11m). The accuracy achieved by the machine learning model is comparable to professional racing drivers and other traditional methods of obtaining racing lines.

Moreover, the ANN calculates predictions for full-size racing circuits in approximately 0.033 seconds, making it over 9,000 times faster than rapid OCP methods and 800 times faster than the most expedient non-trivial approach found in the literature. However, it is worth noting that this approach requires extensive training on existing data, and the current method does not generate a speed trace [10].

Chapter 3

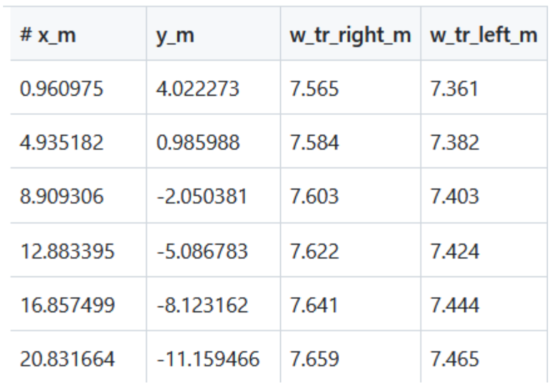
Database and Processing

The objective of this chapter is to outline the creation of the database used for training the Feed Forward Neural Network. Furthermore, it provides insights into data augmentation techniques employed to increase data quantity and explains the preprocessing steps pursued before serving as input for the neural network.

3.1 Dataset

3.1.1 Dataset Description

The original dataset contains the inner and outter boundaries, as well as the centreline of 76 real tracks from various locations around the world, spanning events from Formula 1 to **Deutsche Tourenwagen Masters** (**DTM**). Of these, 27 were obtained from TUMFTM's GitHub [33], while the remaining 49 were generated manually through *Open Street Map*. Each CSV track file contains centreline coordinates (x_m and y_m) and track widths to the right (w_tr_right_m) and left (w_tr_left_m) (refer to Figure 1 for an illustrative example).



# x_m	y_m	w_tr_right_m	w_tr_left_m
0.960975	4.022273	7.565	7.361
4.935182	0.985988	7.584	7.382
8.909306	-2.050381	7.603	7.403
12.883395	-5.086783	7.622	7.424
16.857499	-8.123162	7.641	7.444
20.831664	-11.159466	7.659	7.465

Figure 1: Austin track CSV file. Source: [33]

3.1.2 Data Generation

To obtain accurate predictions with a machine learning model, the quality of the data used for training the model holds paramount significance, as the quantity of data has a direct correlation with the model's capacity to generalize. Therefore, 76 original tracks are not enough to train a neural network. To address this limitation, Data Augmentation techniques come into play. This approach consists of applying specific procedures to generate synthetic data, creating a larger dataset. For example, when working with images, techniques such as flipping, changes in contrast, several types of brightness or even saturation can be employed. The main benefits of Data Augmentation are: improved generalization, reduced overffiting and increased model accuracy [13].

Given the nature of our data, we've employed geometric transformations to further augment the dataset. These transformations include:

1. **Scaling**: This involves resizing objects, allowing for both uniform and width-based adjustments. Scaling factors along the x and y axes, denoted as sx and sy, respectively, determine the extent of resizing (see Figure 3 for an example of scaling applied to track illustrated in Figure 2). The equations are:

   (a) For a point (x, y):

      i. After scaling: $(x', y') = (sx \times x, sy \times y)$

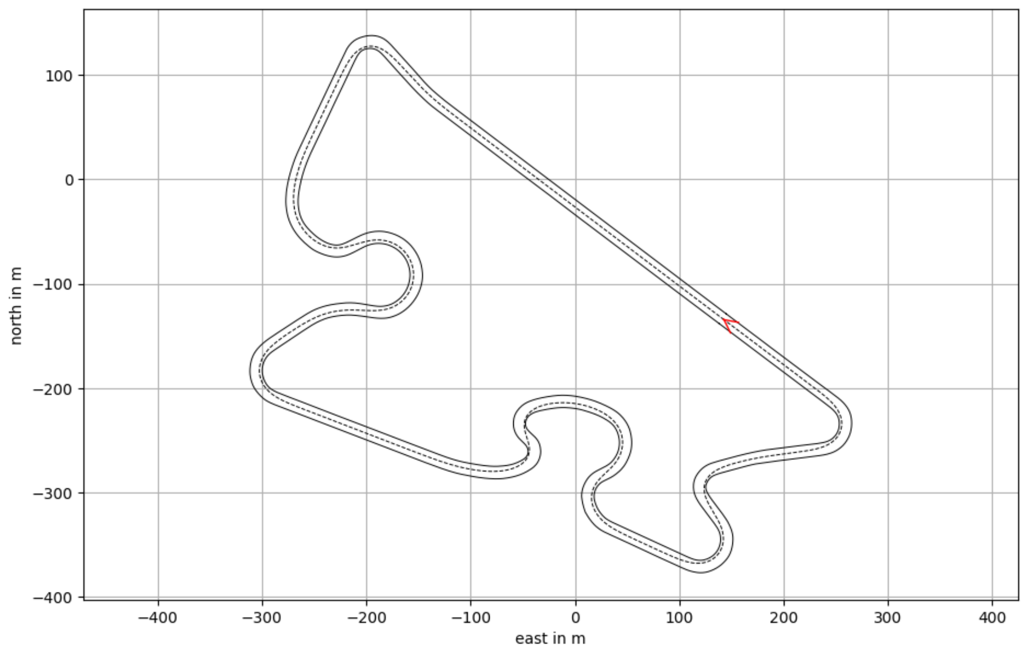


Figure 2: Original Modena track with no augmentation technique applied.

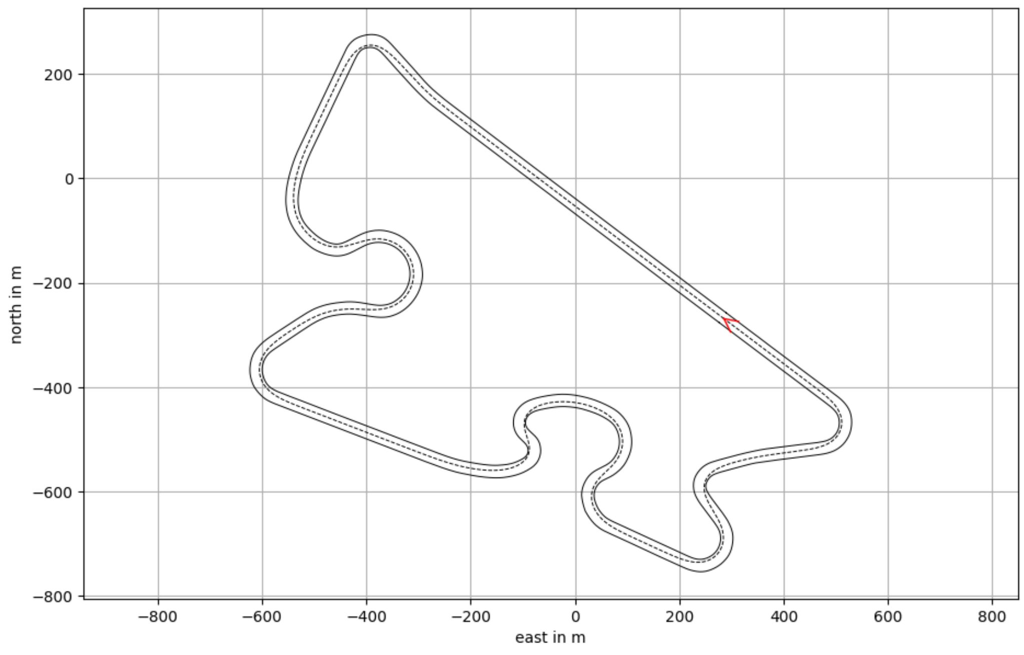


Figure 3: Modena track with scaling applied to create synthetic data.

2. **Reversing (Reflection)**: This transformation mirrors objects along the specified axes. Reflection along the x-axis negates the y-coordinate, while reflection along the y-axis negates the x-coordinate (see Figure 4 for an example of reflection applied to the track illustrated in 2). The equations are:

(a) For a point (x, y):

i. Reflection along the x-axis: (x', y') = (x, -y)

ii. Reflection along the y-axis: (x', y') = (-x, y)

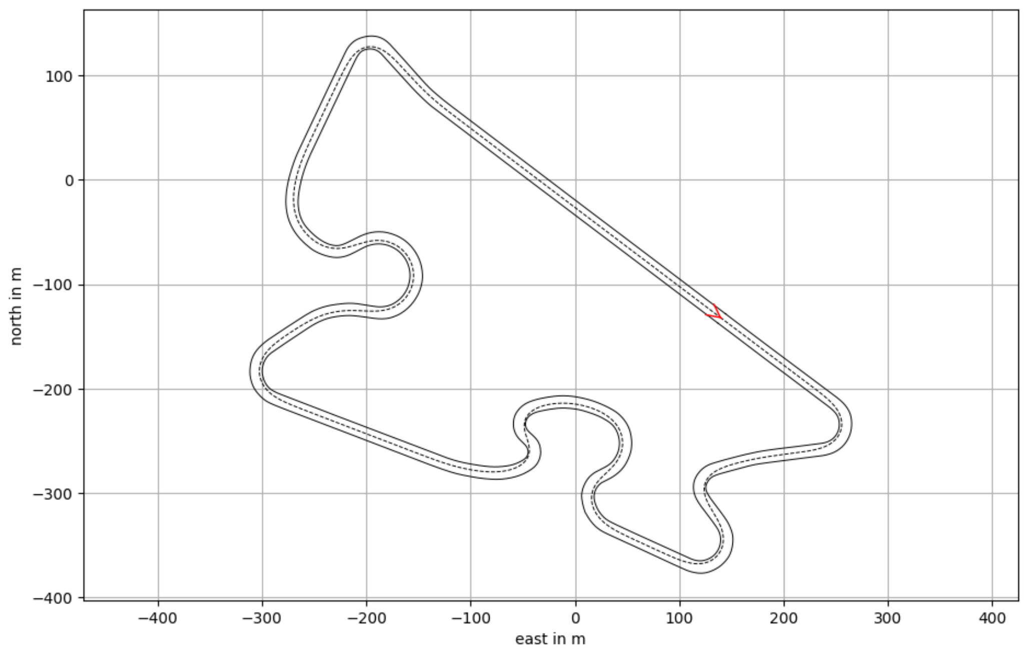


Figure 4: Modena track with augmentation technique reverse applied. It is seen that the direction of the track flipped.

3. **Shearing**: Shearing skews objects along one axis while keeping the other axis fixed. Shearing factors shx and shy determine the skewing amount (see Figure 5 and Figure 6 for an example of sheering along the x-axis and y-axis, respectively, applied to the track illustrated in Figure 2). The equations are:

   (a) For a point (x, y):

      i. After shearing along x-axis: (x', y') = (x + shx $\times$ y, y)

      ii. After shearing along y-axis: (x', y') = (x, y + shy $\times$ x)

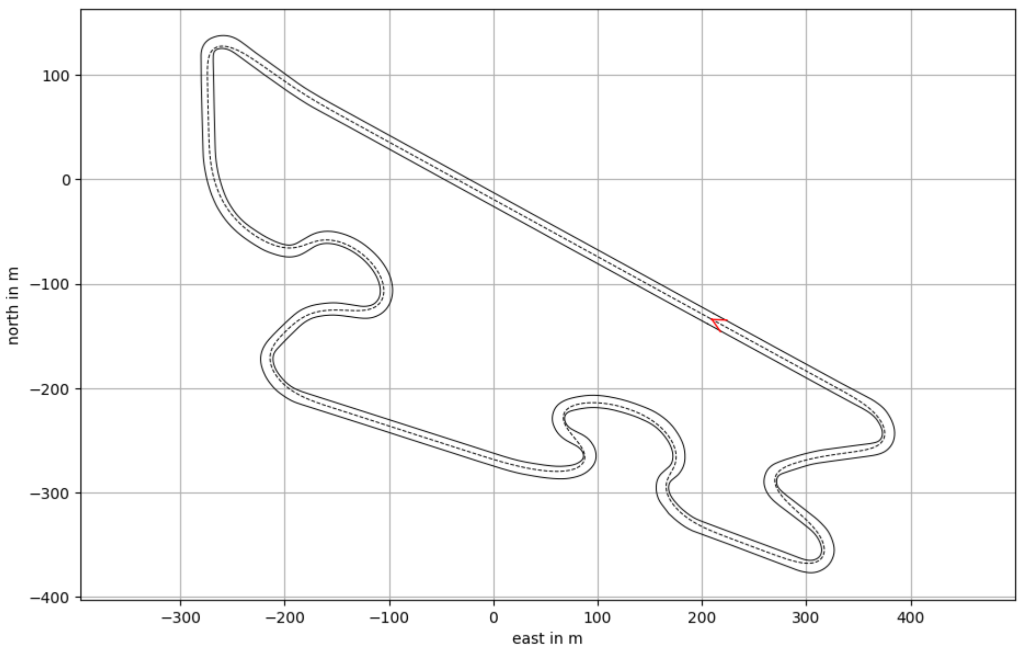


Figure 5: Augmentation with shearing through the axis x.

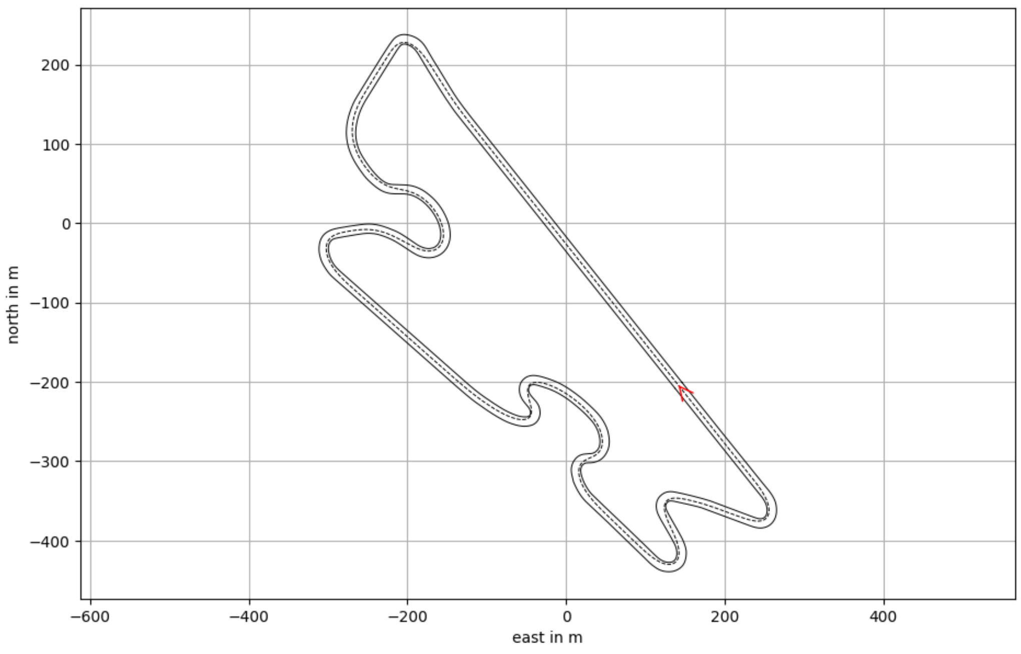


Figure 6: Augmentation with shear through the axis y.

Combinations of the former operations were also applied - consider, for instance, a track undergoing both reverse and width scaling simultaneously (see Figure 7).

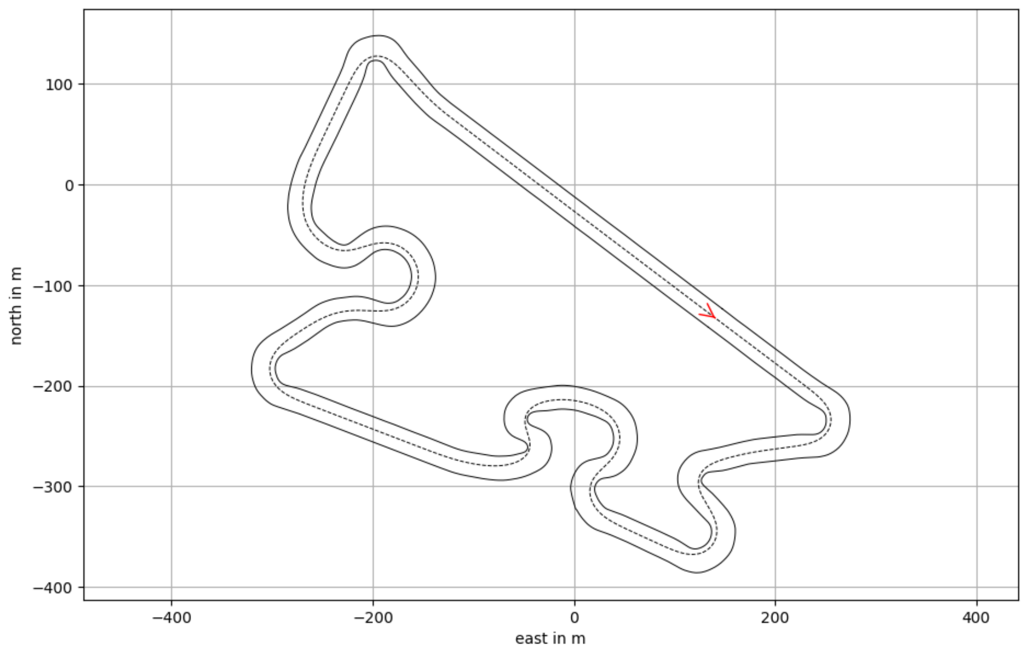


Figure 7: Post-augmentation Modena track where a combination of reverse and scale width was applied.

The database was expanded from 76 to 5533 tracks.

3.1.3 Data Transformation

A traditional method was required to obtain the optimal racing line before inserting it into the ANN. The **OCP** offers an optimal racing line with high accuracy, albeit at the cost of being time-intensive [10]. The **OCP** model was executed with the default parameters of a GP2 car.

Each circuit is subdivided into a series of perpendicular lines to the centreline, those lines are called **Normal Lines**. If two or more Normal Lines intersect, they undergo adjustments until no intersection remains. This process entails altering the angle between the Normal Line and the track's centreline, moving it away from the original 90 degrees. Lines subjected to these adjustments are referred to as **Pseudo-Normals** (see Figure 8). Also, a fixed interval of 5 meters between each Normal Line was defined with the intention to maintain the input dimension minimal [10].

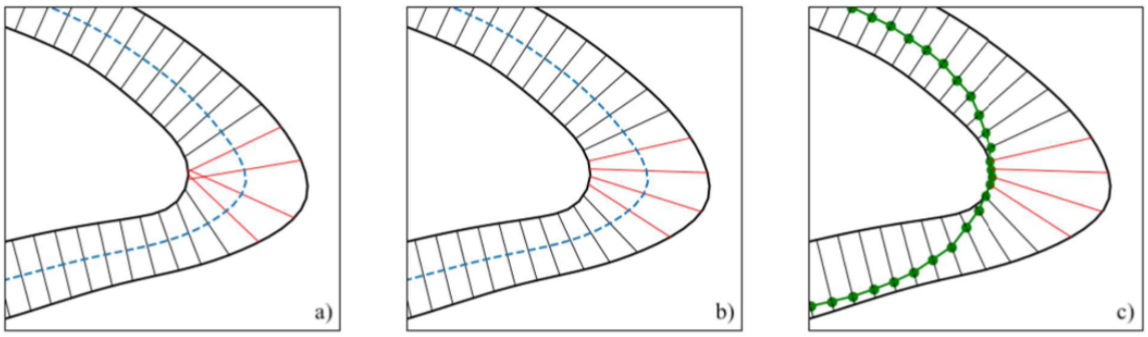


Figure 8: a) Lines normal to the track centreline divide the circuit, b) Normals adjust to become *'pseudo-Normals'* at tight harpins, c) Waypoint location identifies the intersection of the racing line. Source: [10]

Additionally, based on [10] a **Feature Extraction** was made before applying the data into the model. Thus, the following features were extracted:

1. Circuit width or length of each normal line.
2. The location of the intersection of the racing line obtained from the traditional method with the each normal line, i.e., the location of each **waypoint** (w).
3. Information describing the circuit bends in each normal line.

The third feature can be described through the variations between the normal lines, which characterizes the structure of the track.

The knowledge of the track geometry was required due to the necessity of encoding the following features [10] :

1. Length of each normal (l).
2. Angular change between normals (α).
3. Offset angle (θ) between the Normal and the true normal to the track centreline (i.e., 90 degrees, except in the case of an adjusted 'pseudo-normal').

In summary, each normal line has three attributes associated, thus the network has three dimensions for input (see Figure 9). The racing line is described by the location of the waypoint (w) on the normal line, which ranges between values of 0 (left) and 1 (right). To effectively predict the optimal racing line, the values of w are also fed into the network [10].

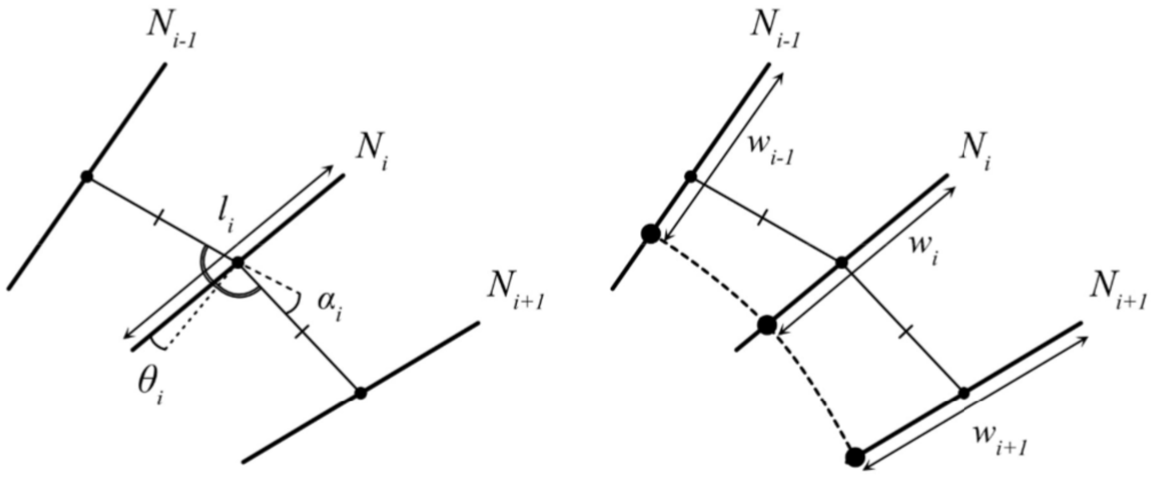
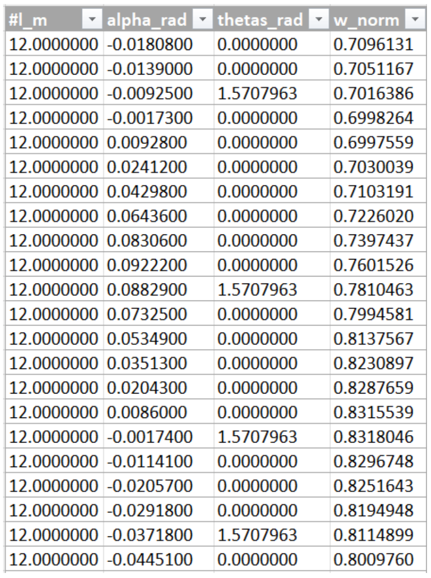


Figure 9: Track Geometry (left) where we can see 3 normal lines and features that will be encoded. Waypoints describing the racing line (right). Each normal line contains a single waypoints. Source: [10]

After feature extraction we obtain the data that will be inserted in the neural network (refer to Figure 10).



#l_m	alpha_rad	thetas_rad	w_norm
12.0000000	-0.0180800	0.0000000	0.7096131
12.0000000	-0.0139000	0.0000000	0.7051167
12.0000000	-0.0092500	1.5707963	0.7016386
12.0000000	-0.0017300	0.0000000	0.6998264
12.0000000	0.0092800	0.0000000	0.6997559
12.0000000	0.0241200	0.0000000	0.7030039
12.0000000	0.0429800	0.0000000	0.7103191
12.0000000	0.0643600	0.0000000	0.7226020
12.0000000	0.0830600	0.0000000	0.7397437
12.0000000	0.0922200	0.0000000	0.7601526
12.0000000	0.0882900	1.5707963	0.7810463
12.0000000	0.0732500	0.0000000	0.7994581
12.0000000	0.0534900	0.0000000	0.8137567
12.0000000	0.0351300	0.0000000	0.8230897
12.0000000	0.0204300	0.0000000	0.8287659
12.0000000	0.0086000	0.0000000	0.8315539
12.0000000	-0.0017400	1.5707963	0.8318046
12.0000000	-0.0114100	0.0000000	0.8296748
12.0000000	-0.0205700	0.0000000	0.8251643
12.0000000	-0.0291800	0.0000000	0.8194948
12.0000000	-0.0371800	1.5707963	0.8114899
12.0000000	-0.0445100	0.0000000	0.8009760

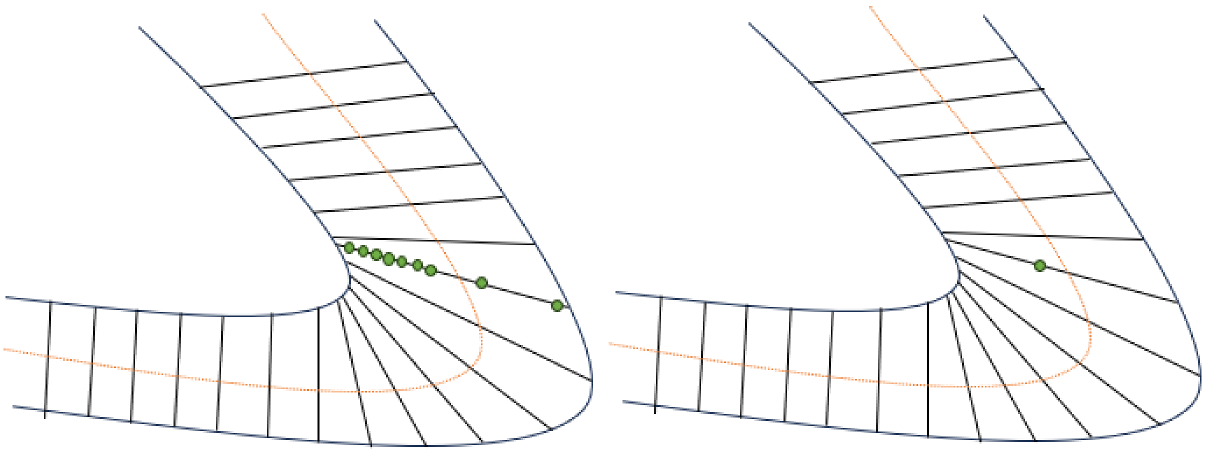
Figure 10: After feature extraction, each track's information is translated into four columns: the length of the normal line, angle alpha, angle theta, and waypoint location.

To obtain a larger dataset and to reduce the amount of real circuits required, the **Sliding Window** method was applied, which consists on dividing each track into same size segments. The benefit of having same size segments, i.e., same number of normal lines per window, is the ability to maintain the dimension of the input as it is. Therefore, not only we have more data to train the machine learning model but it also empowers the capability to make predictions for any track [10].

The parameter called **Foresight** (f) determines the number of normal lines present in each window. It is a bidirectional parameter and encompasses both the previous and subsequent positions. It enables the model to understand the car's trajectory leading up to the current position as well as the trajectory for the next position. To optimize the input size of the network, it is ideal to keep the foresight as minimal as possible. However, a larger track coverage enhances the accuracy of predicting the racing line. It is important to guarantee that the foresight is large enough to cover the longest straight in the dataset. Otherwise, in a straight section of the circuit where there is no visible curvature, the network lacks the knowledge of whether the upcoming bends are left or right-handed. Consequently, it would be incapable of determining the correct racing line for that section [10].

The input size can be represented as $3 \times (2 \times f + 1)$, where the number '3' accounts for the encoded parameters, '2f' is attributed to the bidirectionality of foresight, and '+1' represents the current position.

Similarly to the network's input, minimizing the output is also highly beneficial. However, predicting only the central waypoint of each window would result in a rough line. Therefore, multiple waypoints are predicted per normal line and averaged. Additionally, each point obtained after averaging is connected to the point calculated in the previous and next normal line. The same is applied in the neighboring windows and connected throught the waypoints. This is called the 'moving average' process and, consequently, generates a smooth racing line. The parameter referred to as the **sampling level** (s) determines the number of waypoint pairs surrounding each waypoint predicted by the network. For each normal line, there are $2s+1$ predicted waypoint locations. In the study by Garlick and Bradley [10], it was determined that the optimal value for s is 4, resulting in a total of 9 outputs (see Figure 11). For the purpose of this thesis, we will assume this value as fixed, meaning that all testing will be performed with $s = 4$.



(a) Prediction of all waypoints. (b) Mean of waypoints.

Figure 11: The neural network predicts nine possible waypoints for each normal line on the window. Subsequently, a mean is calculated to create a single waypoint for each line. Later, all waypoints from each line are connected forming a moving average.

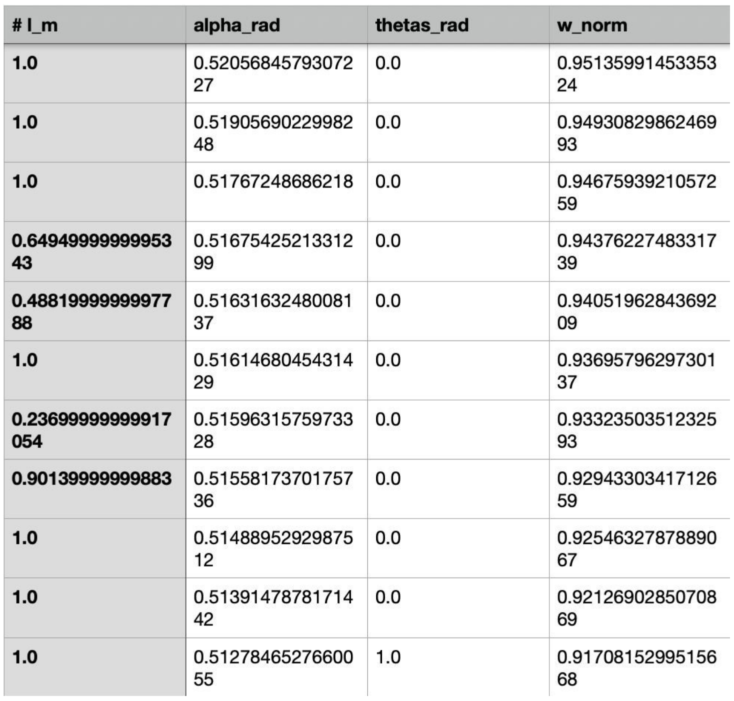
3.2 Processing the Data

Unprocessed or raw data is typically seen as inconsistent, noisy, and may even contain missing values. These characteristics can significantly impact the model's performance. To mitigate these potential issues, preprocessing the dataset has become a necessary step to achieve more favorable results. Data preprocessing encompasses various techniques, including data generalization, smoothing, normalization, and scaling [24]. In this dissertation we will focus on normalization and scaling.

The key distinction between scaling and normalization lies in how these two techniques transform data, with the goal of facilitating comparisons between variables or enhancing the performance of an Artificial Neural Network. Scaling adjusts the data range, ensuring that different variables can be compared within a common range. In contrast, normalization modifies the data range to compress all values into the 0 to 1 range, without compromising the inherent distinctions between value ranges or sacrificing data integrity. Generally, scaling promotes equitable variable comparisons, while normalization is employed to enhance network model performance by ensuring numerical stability and improving the quality of the training process [7].

As previously mentioned, data preprocessing plays a crucial role in achieving optimal results with a machine learning model. Therefore, when considering the data obtained through the OCP method, it becomes evident that each column displays differing ranges that require transformation.

In our initial approach, we applied normalization using Scikit-Learn's **Normalize()** function. However, this approach generated a non-changing result between epochs. Subsequently, we transitioned to an alternative technique: scaling. Scaling was implemented using the **MinMaxScaler()** function, also from Scikit-Learn's library, and was applied to all four columns of the CSV files. This process resulted in a more cohesive dataset (see Figure 12).



# l_m	alpha_rad	thetas_rad	w_norm
1.0	0.5205684579307227	0.0	0.9513599145335324
1.0	0.5190569022998248	0.0	0.9493082986246993
1.0	0.51767248686218	0.0	0.9467593921057259
0.6494999999995343	0.5167542521331299	0.0	0.9437622748331739
0.4881999999997788	0.5163163248008137	0.0	0.9405196284369209
1.0	0.5161468045431429	0.0	0.9369579629730137
0.23699999999917054	0.5159631575973328	0.0	0.9332350351232593
0.90139999999883	0.5155817370175736	0.0	0.9294330341712659
1.0	0.5148895292987512	0.0	0.9254632787889067
1.0	0.5139147878171442	0.0	0.9212690285070869
1.0	0.5127846527660055	1.0	0.9170815299515668

Figure 12: After applying the MinMaxScaler function, the data presents more stability and a cohesive range.

Furthermore, it is essential to highlight that when dealing with a substantial volume of data, specific precautions must be implemented to ensure the feasibility of processing such a large dataset. Initially, the dataset consisted of 5533 tracks. However, due to constraints related to computational memory (RAM) capacity, the dataset had to be downsized to 3000 tracks. To mitigate memory usage, streaming techniques can be employed, which allow for the efficient utilization of available memory resources. Additionally, conversion of data from float64 to float32 can be applied, resulting in a reduction of memory consumption by half compared to the initial requirements.

Chapter 4

Methodology

The understanding that the human brain computes differently than a digital computer inspired the development of ANNs. The primary objective of this chapter is to provide a detailed explanation of how neural networks function while also exploring a wide range of hyperparameters.

4.1 Neural Network Architecture and Hyperparameters

The simplest Artificial Neural Network is called **SingleLayer Perceptron** or **SingleLayer Feed Forward**, due to the fact that the network is the acyclic type.

The SingleLayer Perceptron has an input layer where takes the data $x_1, ..., x_n$, and each input has an adjustable weight associated $w_1, ..., w_n$. We also have the parameter **Bias** (b) - a constant insert to offset the result. Afterwards, it's made the **sum** of the inputs with their respective weights and bias.

This can be seen mathematically as:

$$s = \sum_{i=1}^{N} w_i x_i + b, \tag{4.1}$$

where N is the number of samples.

Then, the **Activation Function** $\phi(s)$ takes all the data from the previous layer and generates a non-linear transformation, limiting the possible amplitude range of the output signal to some finite value. Lastly, the information is passed to the output layer, (y) (see Figure 13):

$$y = \phi(s). \tag{4.2}$$

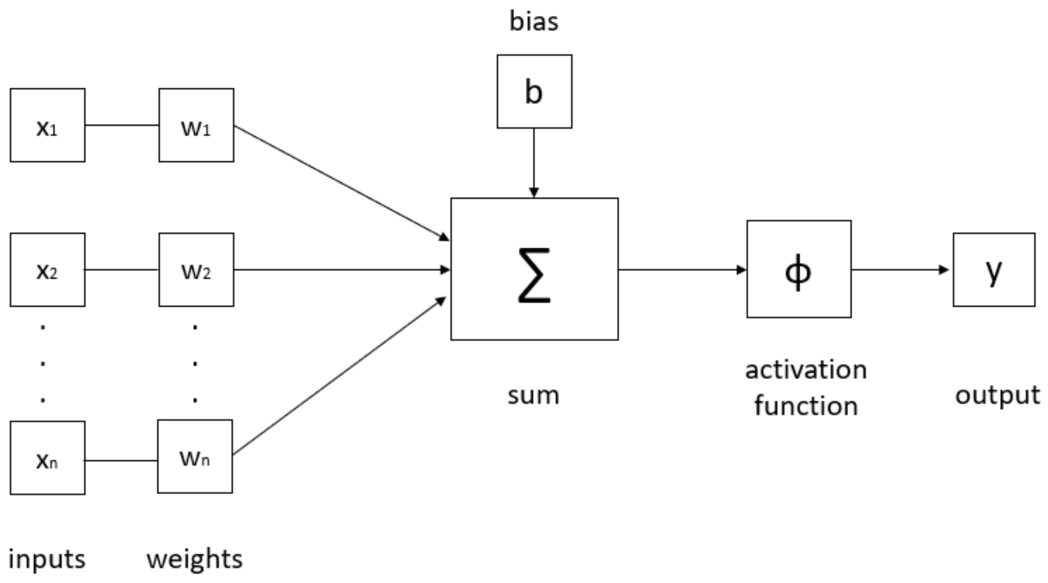


Figure 13: Structure of the SingleLayer Perceptron where the input $x_1, ..., x_n$ has a weight $w_1, ..., w_n$ associated. Bias (b) to offset the result. Subsequently, it's made the sum of input with their respective weight and bias. Lastly, it's made a non-linear transformation using an activation function $\phi(\cdot)$ and passed to the output layer (y).

Hyperparameters are essential components in shaping the behavior of neural networks. They can be broadly categorized into two groups: those determining the network's structure, such as the number of hidden units or layers, and those governing the network's training process, including parameters like the learning rate.

It is important to note that hyperparameters are configured prior to the training phase, preceding the optimization of weights and biases. Properly setting these hyperparameters is a critical step in designing an effective neural network.

4.1.1 Hyperparameters Related to Network's Structure

4.1.1.1 Number of Hidden Layers and Neurons

As aforementioned the simplest neural network is the **SingleLayer Perceptron**. However, the complexity of such model can be increased by adding hidden layers. To this new model with hidden layers we call **MultiLayer Perceptron**. If the model has more than one hidden layer it's called a **MultiLayer Feed Forward Neural Network** (see Figure 14). Hidden layers have the role of assuring the flow between the input and output layers. They perform a non-linear transformation on the input data, allowing the network to detect the features describing the training data. The layers of a Neural Network can be **fully connected**, meaning that a neuron in any layer of the model is connected to all the neurons in the previous layer and so forth. The input propagates through the network in a forward direction and layer-by-layer basis [14].

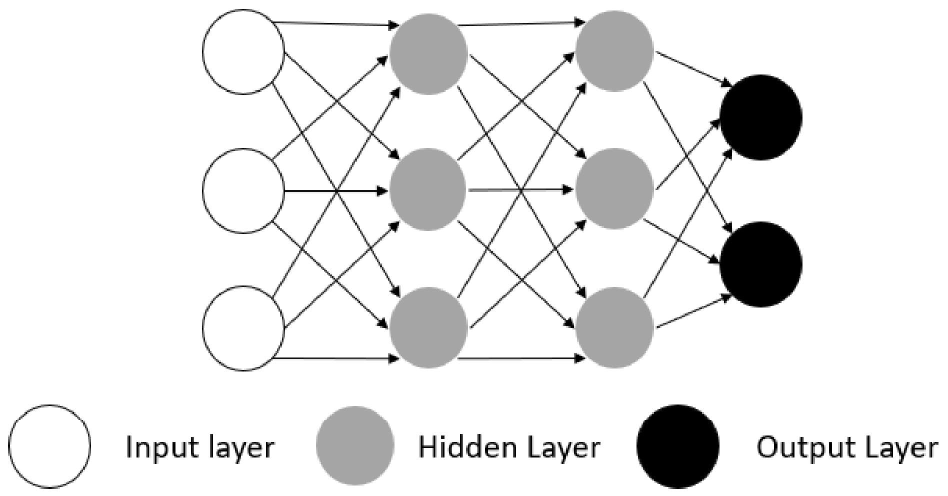


Figure 14: Structure of MultiLayer Feed Forward Neural Network where all four layers are fully connected and encompasses 2 hidden layers.

There are other neural networks besides those previously mentioned, such as: Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs) or, even, Long Short-Term Memory (LSTM). However, those are not in the scope of this thesis.

One of the most challenging aspects of building a Deep Neural Network model that provides high accuracy or has the ability to perform well on the given data is determining the correct values for the hyperparameters. This ability is called **generalization**. In this dissertation, our primary focus will be on the number of layers of the model and the number of neurons per layer. The number of neurons in the input and output layers is determined by the problem at hand. Therefore, a question has arisen regarding the hidden layers, resulting in the following questions:

1. How many layers should exist?
2. How many neurons per layer should be applied?

Since the 1990s, this has been a discussion in the scientific community [16]. Using too few neurons can result in underfitting, where the model cannot ascertain changes in a complicated dataset and results in a very small training error. On the other hand, using too many neurons can result in overfitting, where the model contains too much information from the training set to effectively train all the neurons in the layer. Additionally, using a large number of neurons per layer increases computational time, possibly making it impractical for training [11].

Several techniques have been developed over the years in an attempt to address these issues. However, for the purpose of this dissertation we will work with the following three:

- **Rule-of-thumb:** [15]

  1. The number of hidden neurons should fall within the range defined by the sizes of the input layer and the output layer.

     $$N_i \leq N_h \leq N_o \tag{4.3}$$

     Where:

     N_i : is the number of neurons in the input layer.

     N_h : Number of neurons in the hidden layer.

     N_o : Number of neurons in the output layer.

  2. The number of hidden neurons should be two thirds (2/3) the size of the input layer, plus the size of the output layer.

     $$N_h = \frac{2}{3} \times N_i + N_o \tag{4.4}$$

  3. The number of hidden neurons should be smaller than double the size of the input layer.

     $$N_h < 2 \times N_i \tag{4.5}$$

Although these rules exist, the selection of the architecture comes down to 'trial and error' and it can be very time consuming [11].

- **Geometric Pyramid Rule:** The number of neurons within the neural network should follow a pyramid-like structure, where the layer with the greatest number of neurons is the input layer and the number of neurons diminishes progressively in each subsequent layer [26]:

Table 1 provides a summary of the Geometric Pyramid Rule, where key variables are defined as follows:

- r - Represents the ratio.
- h_i - Signifies the number of hidden neurons in layer i.

Table 1: Geometric Pyramid Rule.

Number of Layers	r	h_i
1	$r = \sqrt{N_i \times N_o}$	$h_1 = r$
2	$r = \sqrt[3]{\frac{N_i}{N_o}}$	$h_1 = m \times r^2$, $h_2 = m \times r$
3	$r = \sqrt[4]{\frac{N_i}{N_o}}$	$h_1 = m \times r^3$, $h_2 = m \times r^2$, $h_3 = m \times r$
$\vdots$	$\vdots$	$\vdots$
i	$r = \sqrt[\frac{1}{i+1}]{\frac{N_i}{N_o}}$	$h_1 = m \times r^i$, $h_2 = m \times r^{i-1}$, ..., $h_i = m \times r$

- **Sheela and Deepa:** Sheela and Deepa [31] introduced a novel method for identifying hidden neurons, demonstrating superior accuracy and minimal error when compared to 101 other criteria. Moreover, in a separate study [35], various methods for determining the number of hidden layers were explored and compared. In this comparison, Sheela and Deepa's method resulted in a neural network that required a smaller number of iterations for training and achieved the the lowest test error.

$$N_h = \frac{4N_i^2 + 3}{N_i^2 - 8} \tag{4.6}$$

High accuracy in the Neural Network is often associated with the incorporation of additional hidden layers. However, it is noteworthy that experts in the field of Machine Learning and Deep Learning have stated that a single hidden layer is typically sufficient to address the majority of problems in this domain. The consideration of adding a second hidden layer should only arise when the single hidden layer fails to provide the desired level of accuracy [11, 23].

4.1.1.2 Network Weight Initialization

Weight initialization or setting the initial weights values is a fundamental task in neural networks. It dictates how well the model is going to perform and it can be divided into two main categories: with or without pre-training. When considering methods without pre-training it can be divided into new categories [22]. However, we will take particular focus on Random Initialization. This category includes:

- **Interval based initialization**: This technique is characterized by choosing the weights from a thought out range or have a upper bound defined. This range is defined with the goal of ensuring that the hidden node activation remains within their activation regions [22].
- **Variance scaling based initialization**: The initial weights are chosen from random distributions and later on scaled to ensure that the variance between input and output layers remain stable or that the output layer remains with the desired level of variance during training and, therefore, preserving the variance of gradients [22].
- **Data-driven initialization**: Weight Initialization is based on information retrieved from the training data [22].

When considering initialization with pre-training, the starting point is determined through an unsupervised approach applied before using a supervised training algorithm [22].

In this dissertation, we have employed a specific weight initialization technique that falls under the category of weight initialization without pre-training. To be more precise, we used the Xavier/Glorot Initialization method, which is characterized as a scaling-based initialization method within the Random Initialization category.

4.1.1.3 Activation Function

Beyond weight initialization there are other parameters that have high importance when tuning a neural network. With that being said, the **Activation Function** plays a crucial role determining the output and enables the network to make the predictions desired.

One common activation function frequently employed in neural networks is the **Sigmoid Function** [14]. It returns values between 0 and 1 and is described mathematically as [13]:

$$\phi(x) = \frac{1}{1+e^{-x}}. \tag{4.7}$$

Other frequently used activation functions include

- **Rectified Linear Unit Function (ReLu)**: Known for its computational efficiency and continuity, it can be expressed as:

$$ReLu(x) = max(0, x). \tag{4.8}$$

- **Hyperbolic Tangent Function (Tanh)**: This function closely resembles the Sigmoid function in terms of its S-shaped, continuous, and differentiable characteristics. However, Tanh yields within the range of -1 to 1 and can be mathematically represented as [13]:

$$Tanh(z) = 2\phi(2z) - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4.9}$$

These activation functions are fundamental in shaping the behavior of neural networks and their choice can have a significant impact on model performance.

4.1.2 Hyperparameters Related to Training Algorithm

4.1.2.1 Learning Rate

The **learning rate** dictates the step size at which a model updates its parameters during the training process. The smaller the learning rate, the smaller the changes in the weights will be from one iteration to the next [13, 14]. It should be sufficiently large to facilitate a rapid learning process, yet small enough to ensure its effectiveness. This hyperparameter is optimized based on validation set error [20]. Typically, this hyperparameter assumes values in the following format: $1e^{-n}$, where $n \in \mathbb{N}$.

The Nadam optimizer was selected for its effectiveness in optimizing neural network models [32].

4.1.2.2 Number of Epochs and Batch Size

To train the network, the dataset has to be seen by the network at least once. During the training of the network, an **epoch** corresponds to a complete iteration through the entire training dataset, while the **batch size** represents the number of training examples processed in each iteration. Essentially, an epoch consists of multiple batches, and the model's parameters are updated based on the error calculated from each batch.

Determining the appropriate number of training epochs can be a arduous task due to the need to find the balance between underfitting and overfitting the model. Underfitting is normally found when the validation error increases. On the other hand, overfitting is defined by the consistent decreases of the training error over time.

In response, certain techniques have been developed to address this issue. **Early Stopping** is one such method, where a parameter is applied at the lowest point of the validation error. At each instance of an error increase, a copy of the parameters at that particular time is created. When the training process concludes, these parameters are updated. The model ceases training when no further improvement is observed in the validation set error. When employing this method, it is important to consider the number of epochs as a hyperparameter [12].

4.2 Evaluation Metrics and Performance Assessment

As previously mentioned, to achieve optimal optimization, it is imperative to fine-tune hyperparameters with the objective of minimizing the **loss function**. In the context of regression tasks, several loss functions are at our disposal, including **Mean Squared Error** (**MSE**), **Mean Absolute Error** (**MAE**) and the **Huber Loss**.

For this dissertation, we employed the Huber Loss as our chosen loss function, which can be mathematically defined as follows [19]:

$$\begin{cases} \frac{1}{2} \times (y_i - \hat{y}_i)^2 & \text{, for } (y_i - \hat{y}_i) \leq \delta \\ \delta \times \left(|y_i - \hat{y}_i| - \frac{1}{2} \times \delta \right) & \text{, otherwise} \end{cases} \tag{4.10}$$

Where y_i represents the true values, and $\hat{y}_i$ represents the predicted values. When discussing the Huber Loss, it is common practice to take δ as the threshold, usually set to 1. The function shows quadratic behaviour if $error < \delta$. On the other hand, it exhibits a linear behavior if $error > \delta$. This feature makes this loss function more susceptible to outliers compared to other loss functions, such as Mean Squared Error.

Other ways of evaluating the model include applying metrics such as accuracy if the data is categorical or MSE, MAE and **Root Mean Squared Error** (**RMSE**) if the data is numerical [13, 14]. These metrics can be calculated using the following formulas:

$$MSE = \frac{1}{n} \times \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{4.11}$$

$$MAE = \frac{1}{n} \times \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{4.12}$$

$$RMSE = \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{4.13}$$

Even though applying metrics is the optimal way to assess the training of the model, another way to see if the hyperparameters are tuned correctly is by observing the **Learning Curve**. In a machine learning context it's a way to visualize how the performance of the model is affected by the training. It can help in assessing whether hyperparameters are correctly tuned and provide insights into the model's convergence and capacity (see Figure 15) [27].

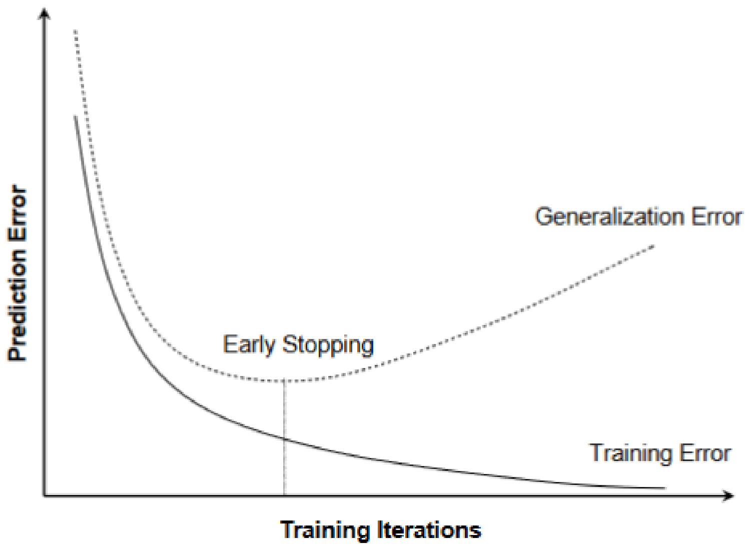


Figure 15: In this figure, two types of learning curves are represented. The first one depicts how a curve should look after training. The second one illustrates what happens when a model becomes underfit, resulting in an increase in validation error. It also indicates the point at which training would have stopped if Early Stopping had been applied. Source: [27]

In Garlick and Bradley's study, they reported achieving the highest accuracy using a three hidden-layered architecture with 450, 200, and 200 neurons in the first, second and third hidden layer, respectively, determined through a grid search method [10]. In this dissertation, our objective is to simplify the model and determine the ideal number of hidden layers and neurons per layer. In the latter, methodologies will be applied. Additionally, we will also perform hyperparameter fine-tuning with the aim of enhancing the model's performance and drawing conclusions from the results. Lastly, the programming language of choice was **python** and the following libraries were used:

- **Numpy**, **Pandas** and **Scikit-Learn** to process the data before inserting it into the model.
- **Tensorflow:** to construct, train and optimize the model.

Chapter 5

Results and Discussion

This chapter presents the results obtained from the investigation of various aspects of model parameterization, including batch size, foresight, learning rate, and hidden layer configurations. Furthermore, it includes the results of exploring the impact of extended epochs (up to 100) and varying foresights (30 and 100) on predictive model performance.

5.1 Determining Batch Size, Foresight and Learning Rate

In this study, the initial experiment aimed to determine the optimal batch size. This test was conducted with a neural network that consisted of three hidden layers with 450, 200, and 200, respectively, as mentioned in Garlick and Bradley's article [10]. A variety of values were experimented with, spanning from 8 to 64, in increments of 8. It was found that using a large batch size significantly increased the training error and, in some cases, it halted the variation of error across epochs. It was determined that the optimal batch size was 8.

Different learning rates were tested in this architecture, starting with a learning rate value of 0.001. The discovery of the right value for the learning rate is crucial because it directly affects the computational time required for training the neural network and the model's performance. The optimal learning curve was determined by monitoring the loss function through learning curves. The following values were tested: 0.001, 0.0001, 0.00001, and 0.000001 (see Figures 16, 17, 18 and 19).

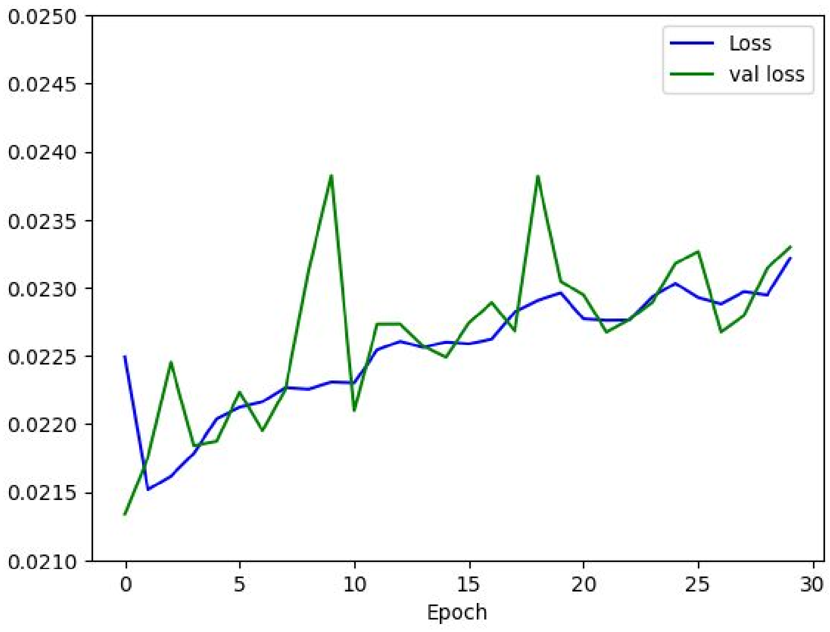


Figure 16: Learning rate curves of a 3 hidden layer neural network with 450, 200 and 200 neurons, respectively, and a learning rate of 0.001.

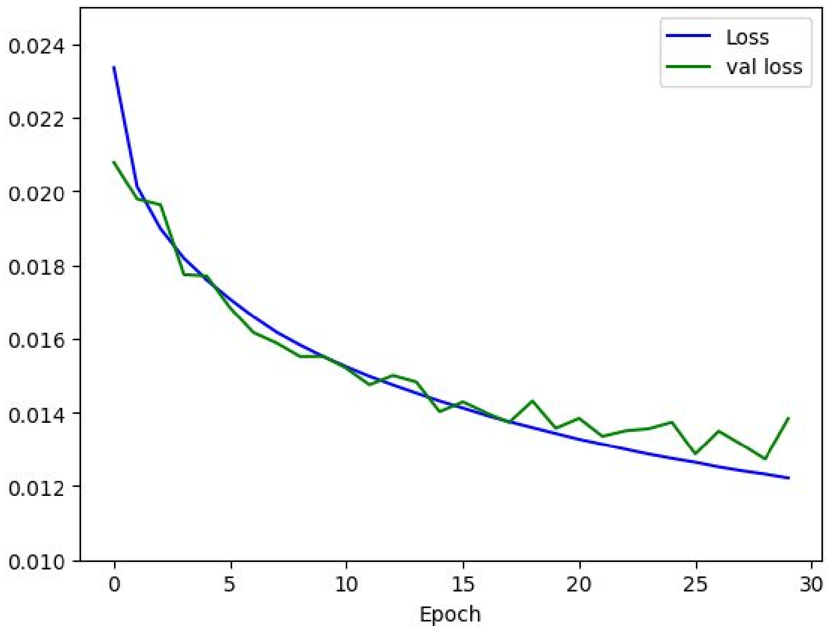


Figure 17: Learning rate curves of a 3 hidden layer neural network with 450, 200 and 200 neurons, respectively, and a learning rate of 0.0001.

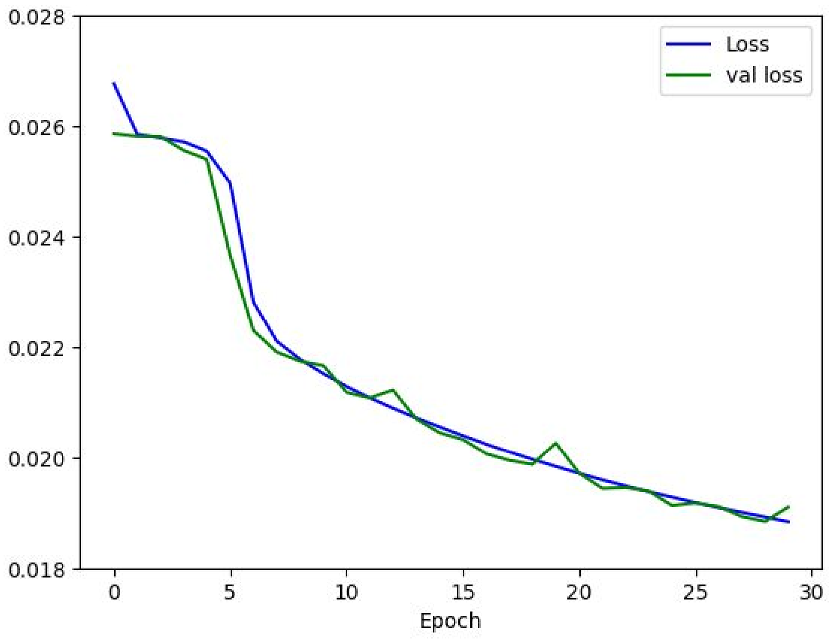


Figure 18: Learning rate curves of a 3 hidden layer neural network with 450, 200 and 200 neurons, respectively, and a learning rate of 0.00001.

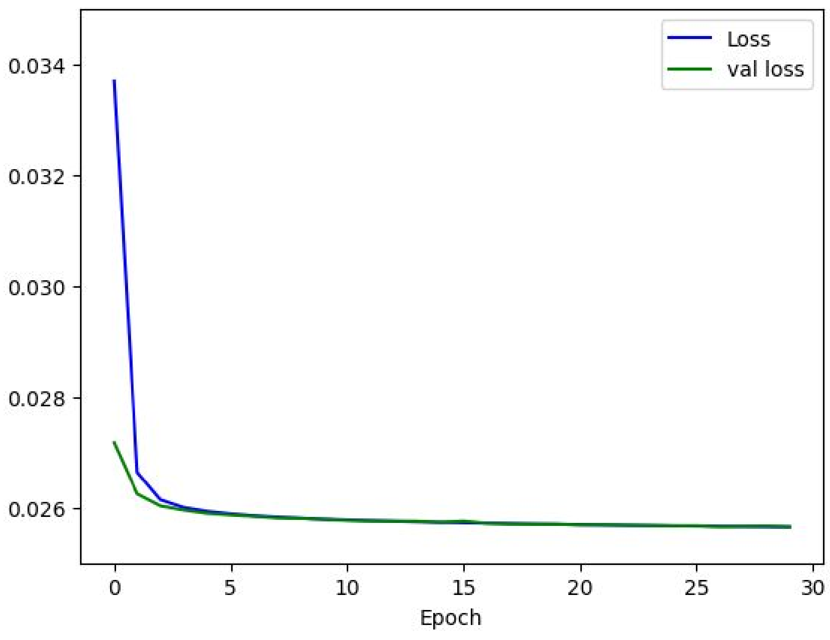


Figure 19: Learning rate curves of a 3 hidden layer neural network with 450, 200 and 200 neurons, respectively, and a learning rate of 0.000001.

After evaluation of the aforementioned learning curves, it is evident that the most stable model is achieved with a learning rate of 0.000001. This value was applied for all testing.

Furthermore, prior to performing any model simplifications, it was necessary to determine the optimal value of another parameter, namely, the foresight (introduced in Section 3.1.3). This parameter is of high importance as it will dictate how many normal lines will exist within a window and this data will subsequently be incorporated into the neural network for training.

We tested seven possible values and performed analysis based on Huber Loss, MSE, MAE and RMSE (see Table 2).

Table 2: For each value tested for foresight, an examination of Huber Loss, MSE, MAE, and RMSE was carried out to determine the best value. An assessment of the number of windows acquired was also made.

Huber Loss	MSE	MAE	RMSE	Foresight	Number of Windows
0.029022	0.058043	0.192914	0.240921	20	4322105
0.025614	0.051228	0.179624	0.226335	30	4262105
0.023794	0.047587	0.172711	0.218145	40	4202105
0.022421	0.044842	0.167096	0.211759	50	4142105
0.021631	0.043263	0.163940	0.207996	60	4082105
0.020932	0.041863	0.160797	0.204606	70	4022105
0.019619	0.039237	0.155134	0.198083	100	3842105

It is evident that there is a inverse correlation between the size of foresight and the values of the Huber Loss, MSE, MAE and RMSE. In all conducted experiments, the model achieved the lowest error when a foresight value of 100 was used. However, with our goal being the reduction of the complexity of the model as much as possible and the error shows a minor decline as the foresight value goes up, we initially performed all testing with a foresight value of 30. Afterwards, carried out testing using the best model identified, both for this foresight value and for a value of 100.

5.2 Evaluating different hidden layers configurations

In some cases a model with one neural network is enough to achieve a good performance. Therefore, in a first step, we considered only one hidden layer and applied the following methodology to determine the number of hidden neurons in the layer: Rule of Thumb, Pyramid Rule and Sheela and Deepa. Moreover, given the substantial volume of data, training is time-consuming, and as a result, all training was performed using 30 epochs.

The first round of testing was performed for a model with one hidden layer and methodology was applied to determine the number of neurons (see Table 3).

Table 3: Three different tests were performed for a model with one hidden layer. The differentiation among the tests lay in the rules applied to determine the number of neurons. Similar to the analysis conducted for foresight, an observation of Huber Loss, MSE, MAE, and RMSE was carried out.

Network	Huber Loss	MSE	MAE	RMSE	Neurons
Pyramid Rule	0.02415	0.04830	0.16834	0.21976	40
Rule of Thumb	0.02427	0.04854	0.16901	0.22032	131
Sheela and Deepa	0.02483	0.04966	0.17456	0.22283	4

After performing a comparison between the error showed in Table 3, it is immediately seen a decrease in error with a simpler model.

Further testing involved increasing the number of hidden layers and applying the previously mentioned techniques to determine the number of neurons per layer, to see if there is any correlation between increasing the number of hidden layers and the decrease of error.

For models with more than one hidden layer, two types of experiments were conducted:

1. All layers consist of an equal number of neurons, and the quantity of neurons in each layer is determined by the methodology applied to a model containing a single hidden layer.
2. Rules extracted from methods such as the Pyramid Rule and the Rule of Thumb were applied to each hidden layer. However, due to a lack of information, the technique presented by Sheela and Deepa did not provide any rules for models with two or more hidden layers. Consequently, no further testing was conducted regarding that methodology.

The following results were achieved with a model featuring two hidden layers (refer to Table 4 and 5):

Table 4: Model with two hidden layers with no variance in the number of neurons

Network	Huber Loss	MSE	MAE	RMSE	N_{h1}	N_{h2}
Pyramid Rule	0.02546	0.05092	0.17832	0.22565	40	40
Thumb Rule	0.02569	0.05137	0.17935	0.22665	131	131
Sheela and Deepa	0.02482	0.04964	0.17449	0.22280	4	4

Table 5: Model with two hidden layers with variance in the number of neurons

Network	Huber Loss	MSE	MAE	RMSE	N_{h1}	N_{h2}
Pyramid Rule	0.02492	0.04983	0.17463	0.22323	67	25
Thumb Rule	0.02553	0.05105	0.17905	0.22595	66	66

For three hidden layers a similar logic was applied (refer to Tables 6 and 7):

Table 6: Model with three hidden layers with no variance in the number of neurons

Network	Huber Loss	MSE	MAE	RMSE	N_{h1}	N_{h2}	N_{h3}
Pyramid Rule	0.02542	0.05084	0.17799	0.22547	40	40	40
Thumb Rule	0.02571	0.05141	0.17999	0.22674	131	131	131
Sheela and Deepa	0.02583	0.05166	0.18051	0.22729	4	4	4

Table 7: Model with three hidden layers with variance in the number of neurons

Network	Huber Loss	MSE	MAE	RMSE	N_{h1}	N_{h2}	N_{h3}
Pyramid Rule	0.02567	0.05135	0.17984	0.22660	86	41	19
Thumb Rule	0.02565	0.05130	0.17957	0.22649	43	43	43

Considering all the results obtained from testing, there is a clear association between the raise of error and the complexity of the model. Additionally, it is worth noting that the increment in model complexity also led to a substantial increase in computational time. Therefore, we arrived to the conclusion that the optimal model structure consists of an input layer, a single hidden layer with 40 hidden neurons and the output layer.

5.3 Evaluating predictive model performance: extended epochs and varied foresight

To ensure the model's optimal predictive performance, two additional tests were conducted. The number of epochs was extended to 100, and the model was tested for foresight of both 30 and 100 (see Table 8).

Table 8: Two final tests were performed with the optimal model found. The distinguishing factor between these tests was the value of foresight.

Network	Huber Loss	MSE	MAE	RMSE	Foresight	Neurons	Epochs
Pyramid Rule	0.02277	0.04555	0.15726	0.21341	30	40	100
Pyramid Rule	0.01410	0.02819	0.11650	0.16791	100	40	100

We also observed the loss learning curves for both training and validation of each model (see Figures 20 and 21).

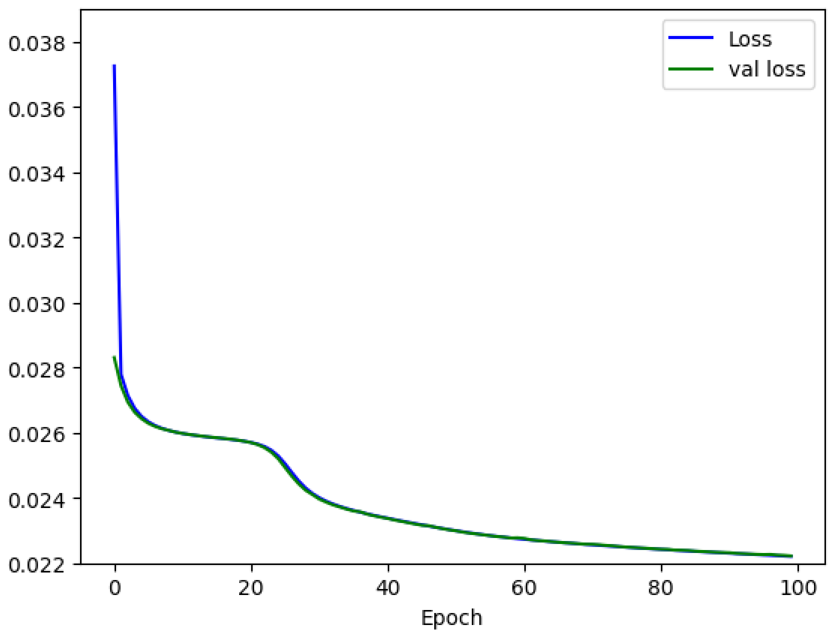


Figure 20: Model with Foresight value of 30.

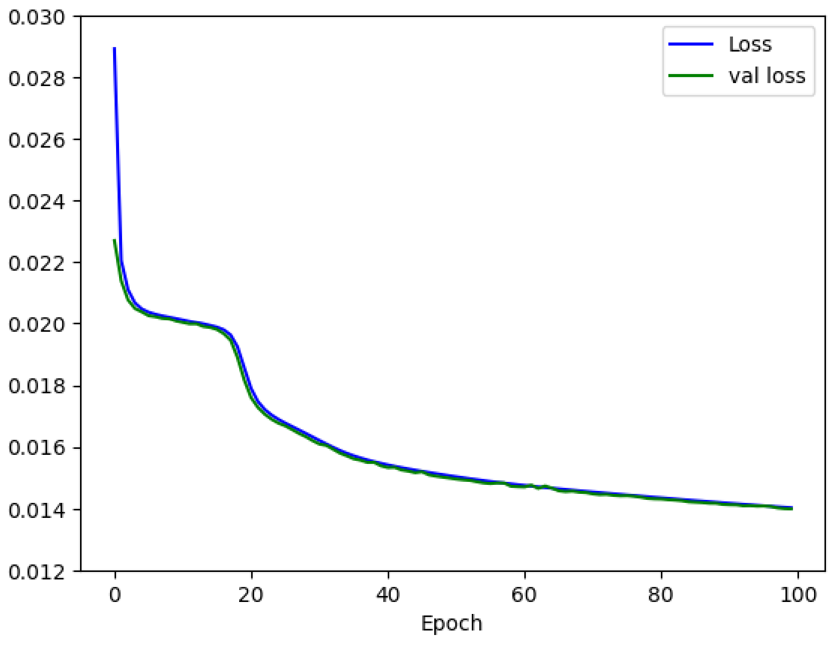


Figure 21: Model with Foresight value of 100.

An evident connection exists between the increase of foresight and the decrease of error. Nonetheless, we will make predictions using both models to determine if there is a significant difference in prediction error. We selected three different tracks for prediction, each varying in difficulty based on the number of bends (refer to Figures 22, 23 and 24).

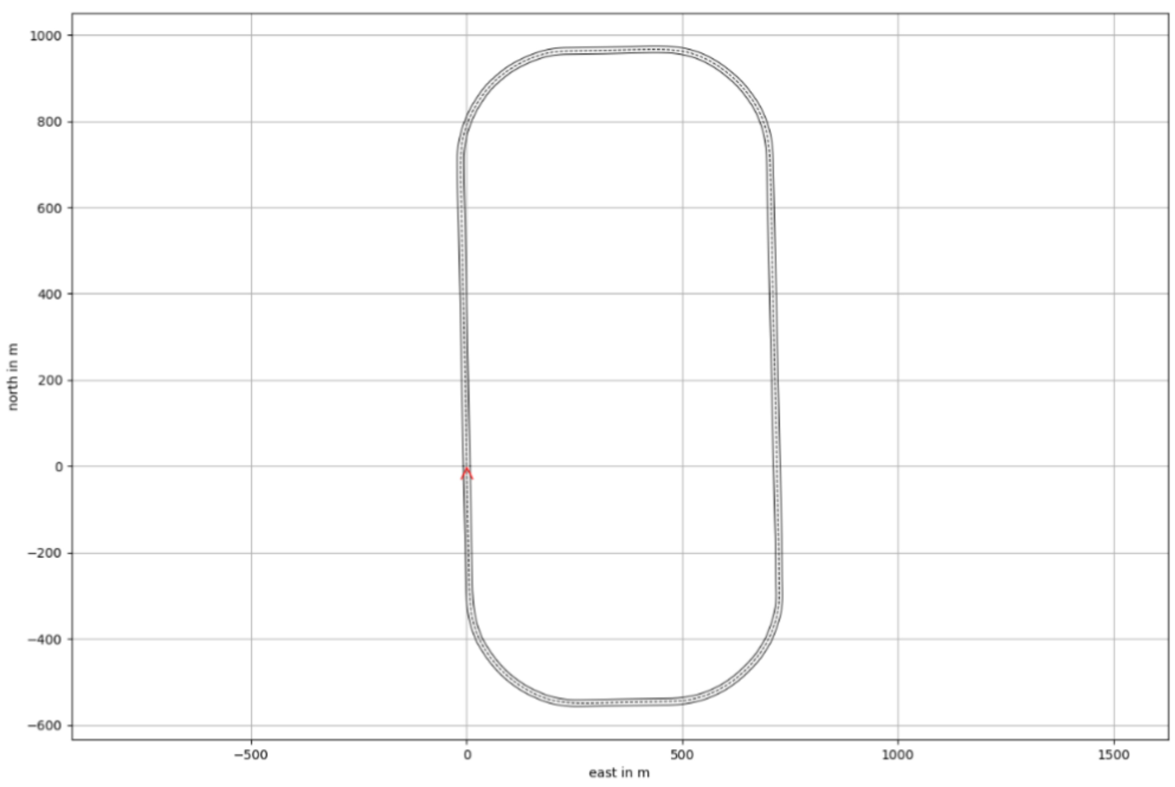


Figure 22: Ims Track.

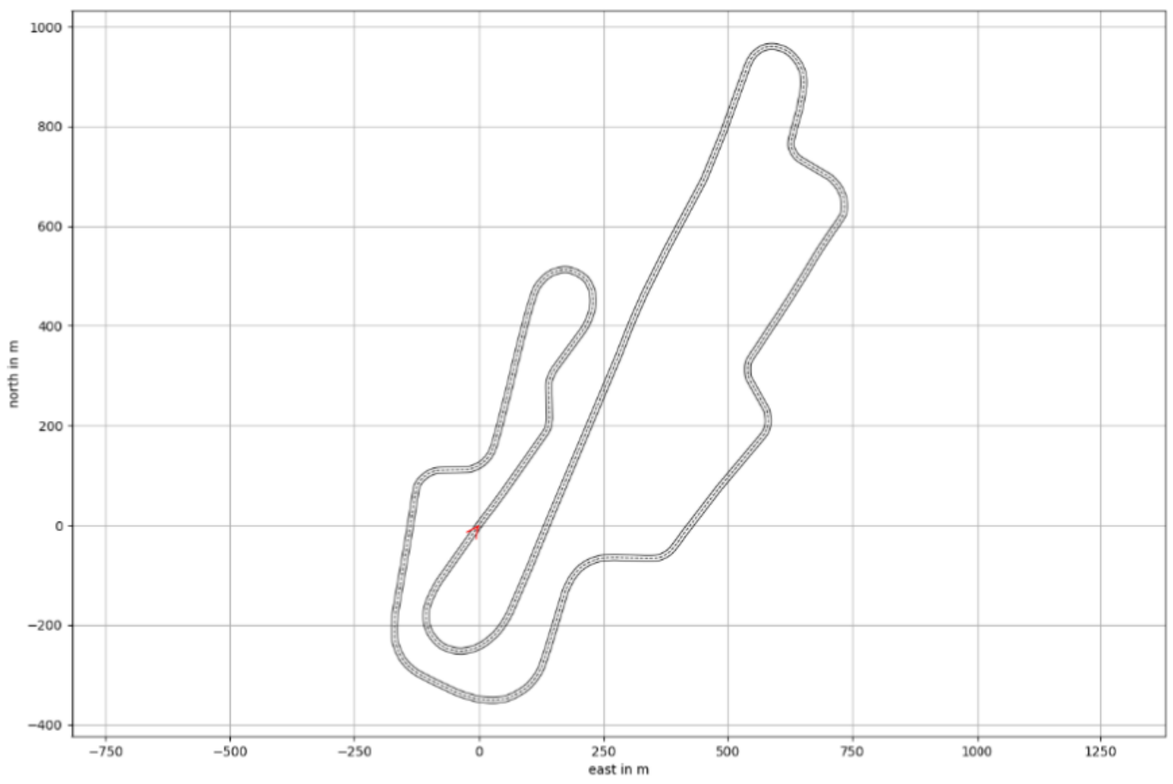


Figure 23: Mugello Track.

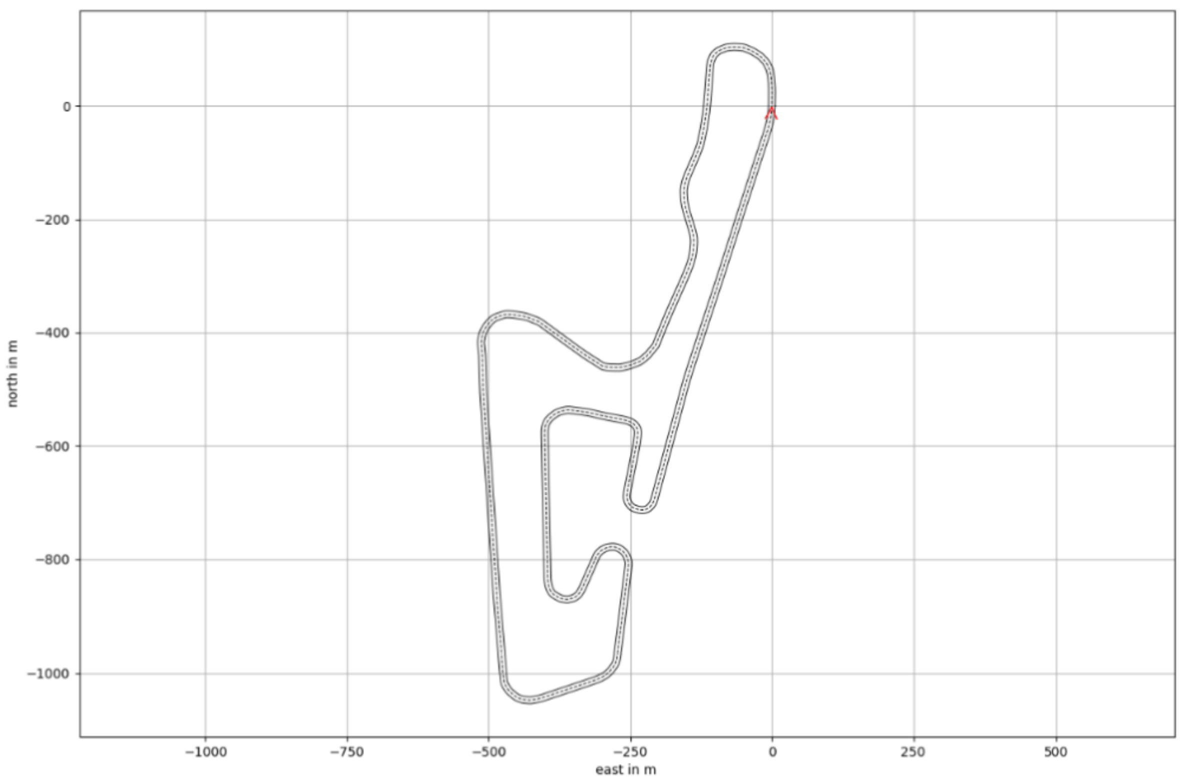


Figure 24: Okayama Track.

During the prediction process, we tracked the MSE, MAE, and RMSE (see Table 9).

Table 9: MSE, MAE and RMSE values for two foresights values on three different tracks

Track	MSE	MAE	RMSE	Foresight	Epochs
Ims	0.07630	0.25264	0.27623	30	100 Epochs With Early Stopping
Mugello	0.26110	0.43087	0.51098	30	100 Epochs With Early Stopping
Okayama	0.08460	0.27033	0.29086	30	100 Epochs With Early Stopping
Ims	0.08759	0.22427	0.29596	100	100 Epochs With Early Stopping
Mugello	0.11021	0.28554	0.33197	100	100 Epochs With Early Stopping
Okayama	0.07691	0.25605	0.27732	100	100 Epochs With Early Stopping

It is important to mention that for complex tracks like Mugello and Okayama, there is a subtle reduction in error. However, for simpler tracks like Ims, there is a small increase. Furthermore, when considering a higher foresight, computational requirements increase exponentially, taking nearly three times longer compared to a lower foresight.

Lastly, we will present the plots for the most complex track - Mugello (see Figures 25 and 26). Refer to Appendice A for a more detailed view of the results.

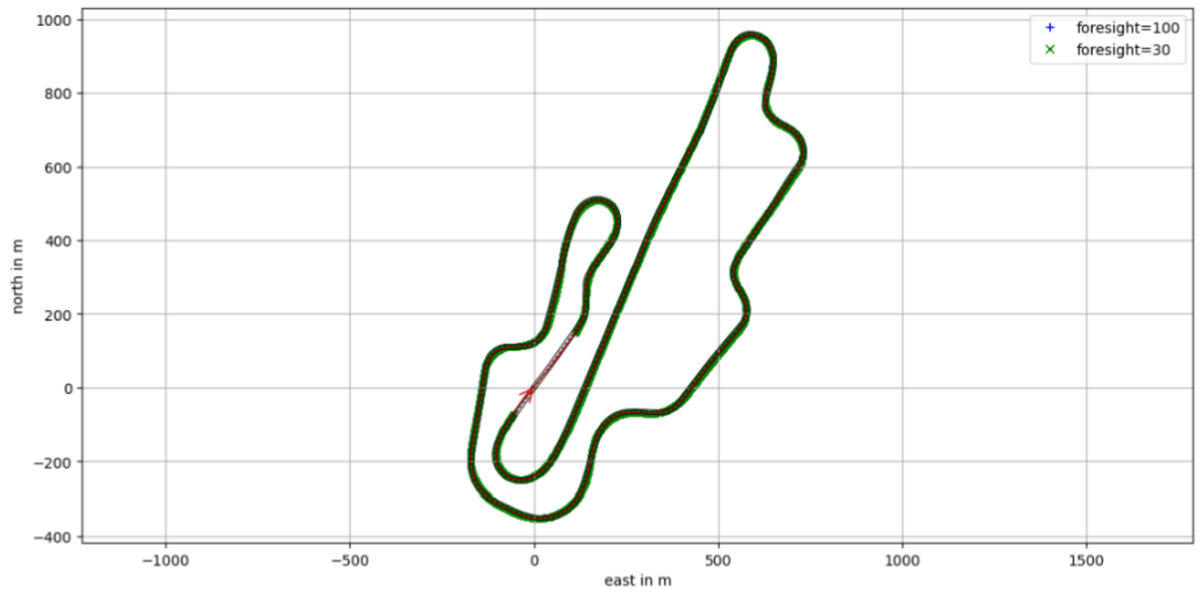


Figure 25: Prediction of Mugello track with both values of foresight.

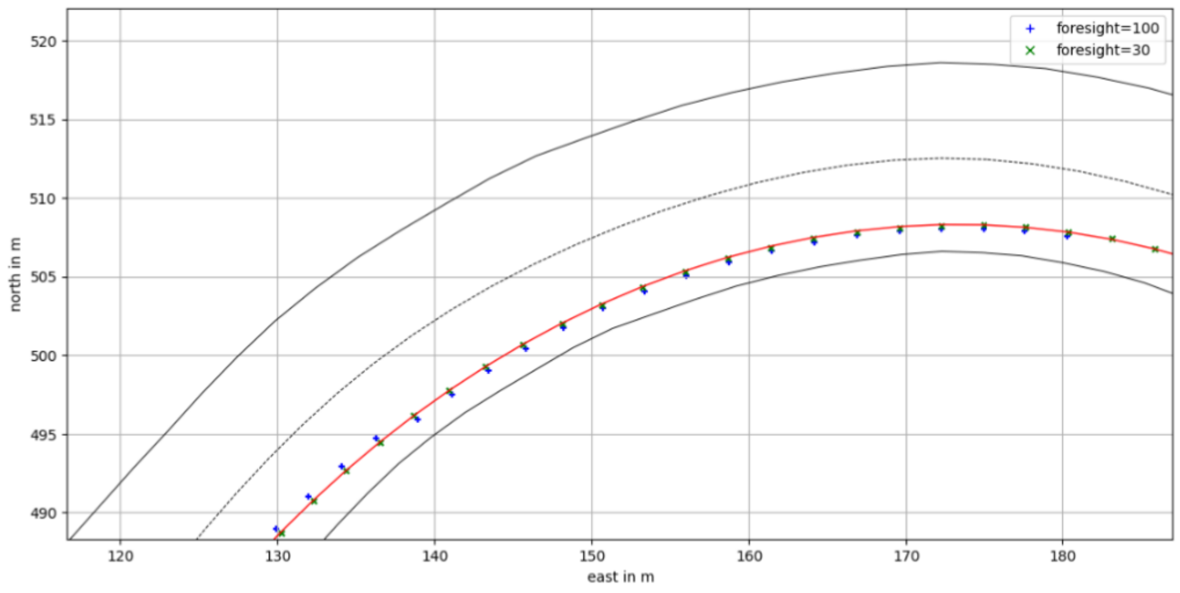


Figure 26: The plot depicting the prediction of the Mugello track for both values of foresight and the OCP line has been enhanced. A clear approximation is observed between the values predicted by the neural network and the OCP line.

After observing and comparing the OCP line with the prediction points acquired from the neural networks, it is evident that there is a proximity to the OCP line. Even though the models with different foresight exhibit varying prediction errors, they are not high enough to generate less favorable results.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This investigation presented a machine learning approach to the problem of discovering the optimal trajectory for a given track. A detailed explanation was made about neural networks, giving insight on how they function, particularly concerning topics such as weights, learning rates, epochs, metrics, and other critical components of the model.

A dataset comprising 76 Formula 1 and DTM tracks was created and expanded using augmentation techniques, which involved shearing, reversing, scaling, and combinations of these methods. Subsequently, a traditional approach was employed to determine the optimal racing line, which is crucial to determine waypoint location. Furthermore, each track was segmented into windows containing either normal or pseudo-normal lines. Feature extraction was required to obtain information such as circuit width and details about the circuit's bends along each normal line. After encoding all features, including circuit width, angular changes between normals, offset angles, and waypoint locations, these data points will serve as inputs to the network. Furthermore, during a pre-processing stage, data scaling was applied to ensure consistency across the dataset.

Experiments were conducted with the aim of simplifying a model and achieving high accuracy with low computational time. The investigation involved a primary step: a model with one hidden layer. Three methodologies were applied to determine the number of hidden neurons: the Pyramid Rule, the Rule of Thumb, and Sheela and Deepa. Furthermore, more testing was performed by increasing hidden layers, which can be split into two categories: models where all layers have the same number of neurons, meaning there is no variation on a layer-by-layer basis, and models with variance in the number of neurons, with the latter being dictated by rules established by the scientific community. However, the Sheela and Deepa method suffers from a severe lack of information and, as a result, no further testing was conducted.

Taking into consideration the results obtained and computational time available, we were able to respond to the following question: How many layers should the model comprise, and what should be the number of neurons per layer?

The model with best accuracy was the simpler model, with one hidden layer and the application of the pyramid rule for the number of neurons and it is structured as follows: an input layer with 183 neurons, one hidden layer with 40 neurons and, lastly, the output layer with 9 neurons. This outcome aligns with the consensus in the scientific community that a single hidden layer is typically sufficient to address most machine learning problems, reinforcing this established theory [11].

In addition, one more experiment was performed to observe the variation of error in a neural network with a higher foresight. The number of epochs was increased in an attempt to further minimize the error. It was observed that not only did the error decrease with a higher number of epochs, but it decreased even more with a higher foresight.

Lastly, predictions of the optimal path were made for two models with different values of foresight. It was seen that a lower value of foresight results in smaller MSE error for simpler tracks, like IMS. Conversely, it generates higher MSE errors for more complex tracks, namely Mugello and Okayama. On the other hand, a model with higher foresight gives a higher MSE error in simpler tracks and a smaller one in more complex ones. It is important to note that even though a more complex model produces lower prediction errors, it exhibits low variance and requires a longer computational time.

In conclusion, this dissertation has demonstrated the efficacy of utilizing a Feed Forward Neural Network to predict optimal trajectories for Formula 1 and DTM tracks. By creating a model with a single hidden layer comprising 40 neurons, we achieved remarkable predictive performance.

Furthermore, our experiments shed light on the sensitivity of model performance to parameters such as foresight, emphasizing the need for adaptability based on the specific track's complexity. This adaptability not only reduces errors but also lays the foundation for achieving more accurate predictions in real-world scenarios, where track layouts vary widely.

The findings from the experiments provide valuable insights for practitioners and researchers in the fields of trajectory optimization, machine learning, and deep learning. The adaptability of the model, as demonstrated in our experiments, underscores the importance of tailoring predictive models to the unique characteristics of the task at hand.

6.2 Future Work

The pursuit of an optimal trajectory remains a compelling challenge, and our work signifies a significant step towards realizing its full potential through the application of machine learning and deep learning techniques. To enhance our research, the first step would be to use the full dataset initially created, instead of only half. Moreover, one of the main challenges of this dissertation was conducting a significant number of tests with a neural network that is time-consuming. For the most complex architecture, with three hidden layers, it took more than a day to train the model. All training was made with a MacBook Air M1. Therefore, an improvement could be made by exploring other ANN models. Furthermore, it would be advantageous, whenever possible, to harness the virtual environments available on platforms such as Google Cloud Platform or Microsoft Azure Machine Learning Studio. These platforms have the potential to expedite and streamline the computational aspects of our research, thereby advancing our exploration of optimal trajectories.

Bibliography

[1] N. Bianco, L. Roberto, and M. Gadola. Minimum time optimal control simulation of a gp2 race car. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 232, 2017. doi: 10.1177/0954407017728158.

[2] N. Bianco, E. Bertolazzi, F. Biral, and M. Massaro. Comparison of direct and indirect methods for minimum lap time optimal control problems. *Vehicle System Dynamics*, 57:1–32, 2018. doi: 10.1080/00423114.2018.1480048.

[3] D. Butler and V. Karri. Race car chassis tuning using artificial neural networks. pages 866–877, 2003. doi: 10.1007/978-3-540-24581-0_74.

[4] P. Cai, X. Mei, L. Tai, Y. Sun, and M. Liu. High-speed autonomous drifting with deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5:1247–1254, 2020. doi: 10.1109/LRA.2020.2967299.

[5] E. Capo and D. Loiacono. Short-term trajectory planning in torcs using deep reinforcement learning. pages 2327–2334, 2020. doi: 10.1109/SSCI47803.2020.9308138.

[6] D. Chindamo and M. Gadola. Estimation of vehicle side-slip angle using an artificial neural network. *MATEC Web of Conferences*, 166:02001, 2018. doi: 10.1051/matecconf/201816602001.

[7] K. Djordjevic, M. Jordovic Pavlovic, Z. Cojbasic, S. Galovic, M. Popovic, M. Nesic, and D. Markushev. Influence of data scaling and normalization on overall neural network performances in photoacoustics. *Optical and Quantum Electronics*, 54, 2022. doi: 10.1007/s11082-022-03799-1.

[8] L. Fridman, L. Ding, B. Jenik, and B. Reimer. Arguing machines: Human supervision of black box ai systems that make life-critical decisions. pages 1335–1343, 2019. doi: 10.1109/CVPRW.2019.00173.

[9] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Duerr. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6:4257–4264, 2021. doi: 10.1109/LRA.2021.3064284.

[10] S. Garlick and A. Bradley. Real-time optimal trajectory planning for autonomous vehicles and lap time simulation using machine learning. *Vehicle System Dynamics*, 60:1–21, 2021. doi: 10.1080/00423114.2021.2011929.

[11] P. Gaurang, A. Ganatra, Y. Kosta, and D. Panchal. Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers. *International Journal of Computer Theory and Engineering*, 3:332–337, 2011. doi: 10.7763/IJCTE.2011.V3.328.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[13] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, pages 41–290. O'Reilly Media, Inc., 2019.

[14] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2 edition, 1999.

[15] Jeff T. Heaton. *Introduction to Neural Networks with Java*. Heaton Research, Inc., 2005.

[16] C. Ibnu, J. Santoso, and K. Surendro. Determining the neural network topology: A review. *ICSCA '19: Proceedings of the 2019 8th International Conference on Software and Computer Applications*, pages 357–362, 2019. doi: 10.1145/3316615.3316697.

[17] Nitin R. Kapania. Trajectory planning and control for an autonomous race vehicle. pages 1–52, 2016.

[18] B. Lenzo and V. Rossi. A simple mono-dimensional approach for lap time optimisation. *Applied Sciences*, 10:1498, 2020. doi: 10.3390/app10041498.

[19] Gregory P. Meyer. An alternative probabilistic interpretation of the huber loss. pages 5261–5269, 2021. doi: 10.48550/arXiv.1911.02088.

[20] M. Moreira and E. Fiesler. Neural networks with adaptive learning rate and momentum terms. Technical report, Idiap, 1995.

[21] J. Muñoz, G. Gutierrez, and A. Sanchis de Miguel. A human-like torcs controller for the simulated car racing championship. pages 473–480, 2010. doi: 10.1109/ITW.2010.5593318.

[22] M. Narkhede, P. Bartakke, and M. Sutaone. A review on weight initialization strategies for neural networks. *Artificial Intelligence Review*, 55:1–32, 2022. doi: 10.1007/s10462-021-10033-z.

[23] V. Nichoga, I. Prudyus, and L. Vashchyshyn. Process of building artificial neural network for automatic detection of signals from transverse cracks in the rail head. *Problemy Kolejnictwa - Railway Reports*, pages 59–62, 2017. doi: 10.36137/1758e.

[24] Hadeel S. Obaid, Saad A. Dheyab, and Sana S. Sabry. The impact of data pre-processing techniques and dimensionality reduction on the accuracy of machine learning. pages 279–283, 2019. doi: 10.1109/IEMECONX.2019.8877011.

[25] P. Ongsulee. Artificial intelligence, machine learning and deep learning. pages 1–6, 2017. doi: 10.1109/ICTKE.2017.8259629.

[26] A. Pano-Azucena, E. Tlelo-Cuautle, S. Tan, B. Ovilla-Martinez, and L. de la Fraga. Fpga-based implementation of a multilayer perceptron suitable for chaotic time series prediction. *Technologies*, 6:90, 2018. doi: 10.3390/technologies6040090.

[27] C. Perlich. Learning curves in machine learning. 2011. doi: 10.1007/978-0-387-30164-8_452.

[28] J. Quadflieg, M. Preuss, O. Kramer, and G. Rudolph. Learning the track and planning ahead in a car racing controller. pages 395–402, 2010. doi: 10.1109/ITW.2010.5593327.

[29] M.A. Reche and K. Stratis. Teaching a vehicle to autonomously drift: A data-based approach using neural networks. *Knowledge-Based Systems*, 153:12–28, 2018. doi: 10.1016/j.knosys.2018.04.015.

[30] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Review*, 65:386–408, 1958. doi: 10.1037/h0042519.

[31] K. Gnana Sheela, Subramaniam N. Deepa, et al. Review on methods to fix number of hidden neurons in neural networks. *Mathematical problems in engineering*, 2013, 2013.

[32] D. Timothy. Incorporating nesterov momentum into adam. 2016.

[33] TUMFTM. Dataset. `https://github.com/TUMFTM/racetrack-database`. Accessed on 2022-10-30.

[34] M. Veneri and M. Massaro. A free-trajectory quasi-steady-state optimal-control method for minimum lap-time of race vehicles. *Vehicle System Dynamics*, 58:1–22, 2019. doi: 10.1080/00423114.2019.1608364.

[35] T. Vujičić, T Matijevi, J. Ljucović, A. Balota, and Z. Ševarac. Comparative analysis of methods for determining number of hidden neurons in artificial neural network. *Central European conference on information and intelligent systems*, 219, 2016.

[36] R. Yu, Z. Shi, H. Chaoxing, T. Li, and Q. Ma. Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle. pages 4958–4965, 2017. doi: 10.23919/ChiCC.2017.8028138.

Part I

Appendices

Appendix A

Details of results

The following images correspond to the prediction plots of the three predicted tracks, each with varying levels of complexity.

First, we begin with a simpler circuit, the IMS, focusing on an initial overall prediction of the full track. Subsequently, we zoom in on the sections where bends occur (see Figures 27 to 30).

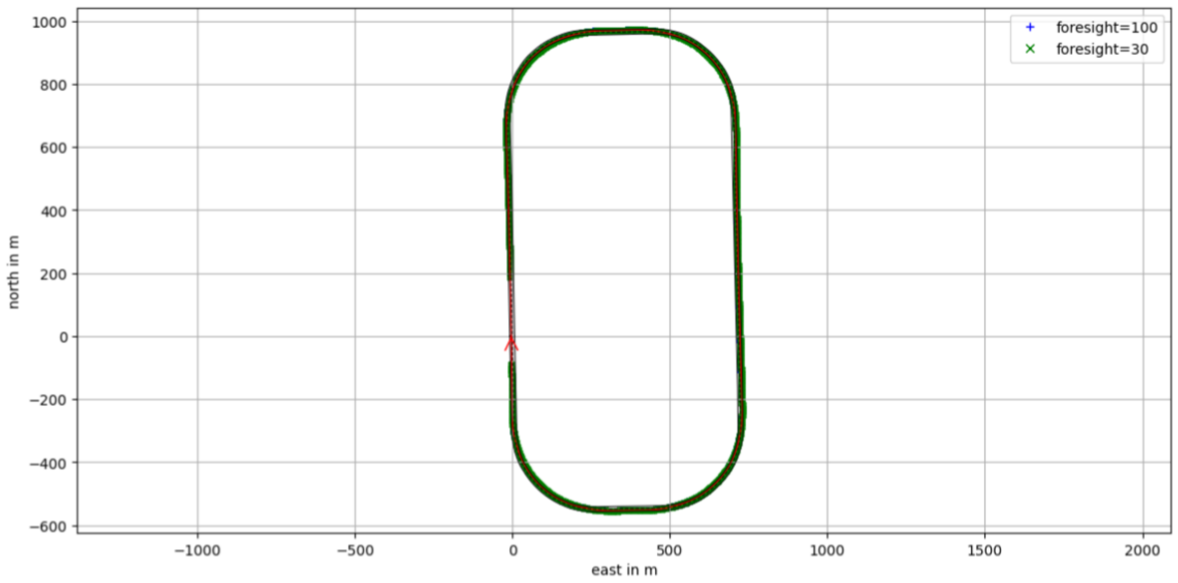


Figure 27: Ims Prediction.

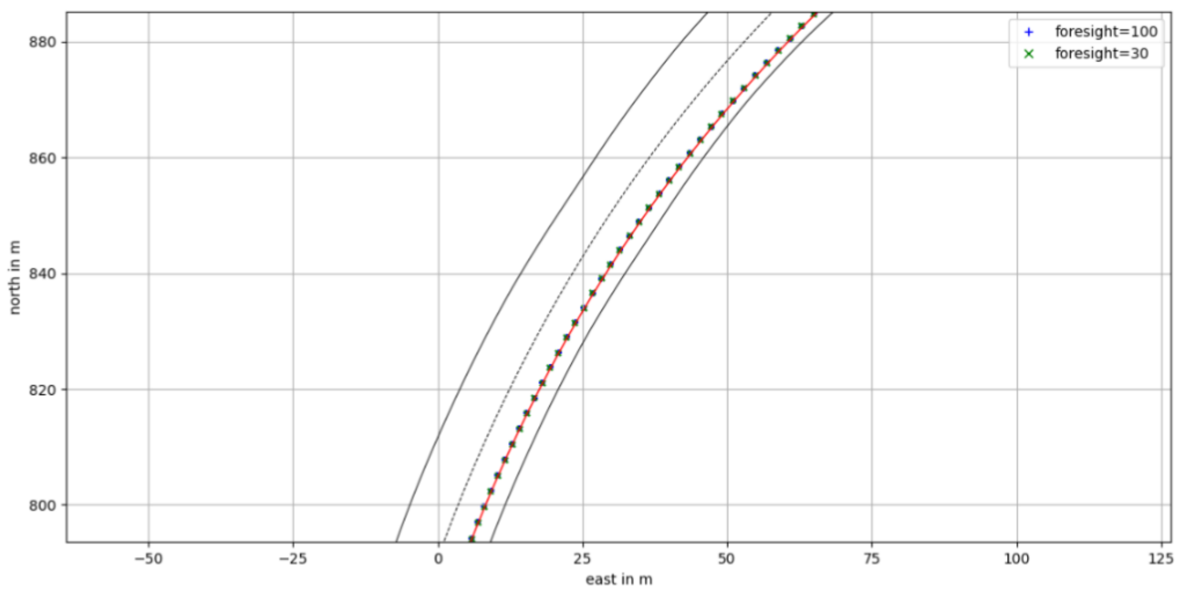


Figure 28: lms with zoom 1.

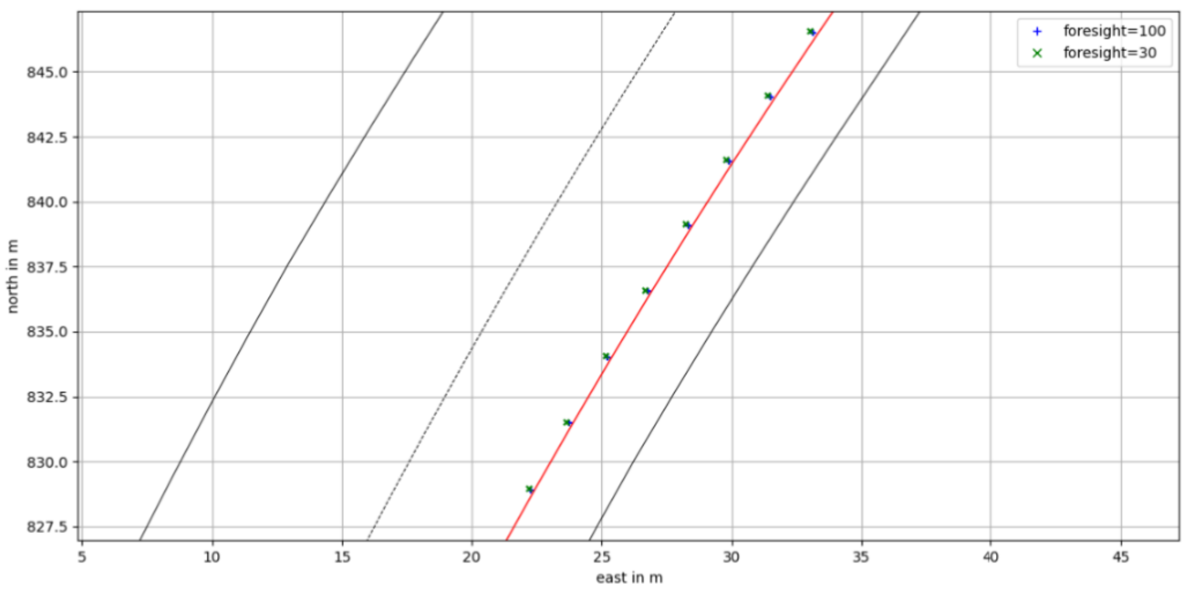


Figure 29: lms with zoom 2.

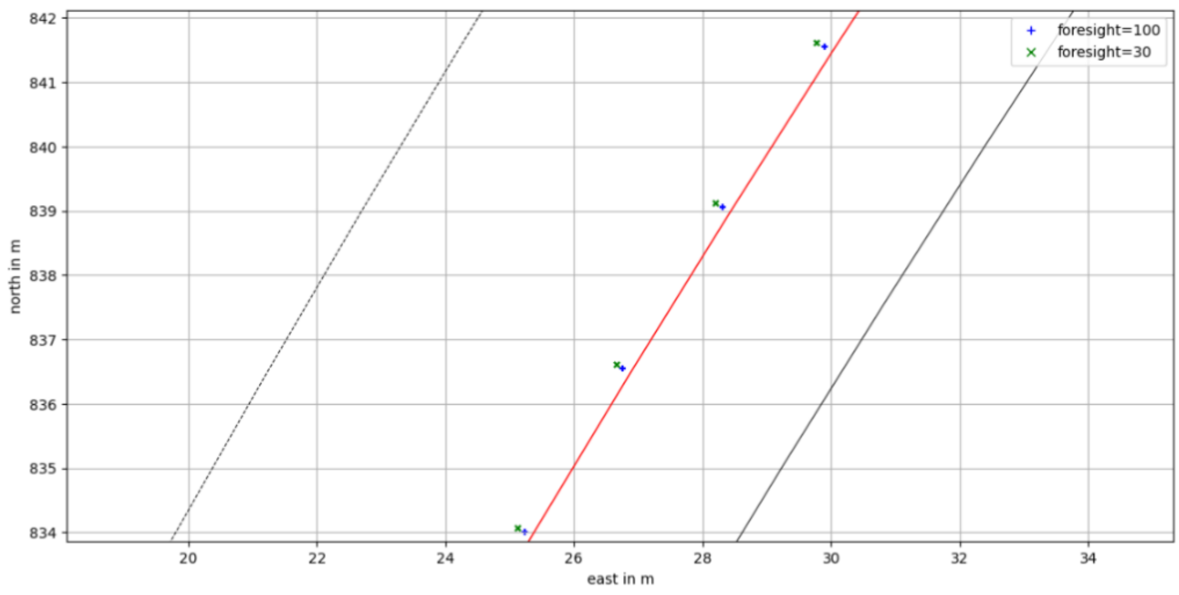


Figure 30: Ims with zoom 3.

Secondly, we present predictions for a more complex track, Okayama, while sequentially applying the same logic of zooming in on regions where bends occur, as we did for the IMS track (see Figure 31 to 37).

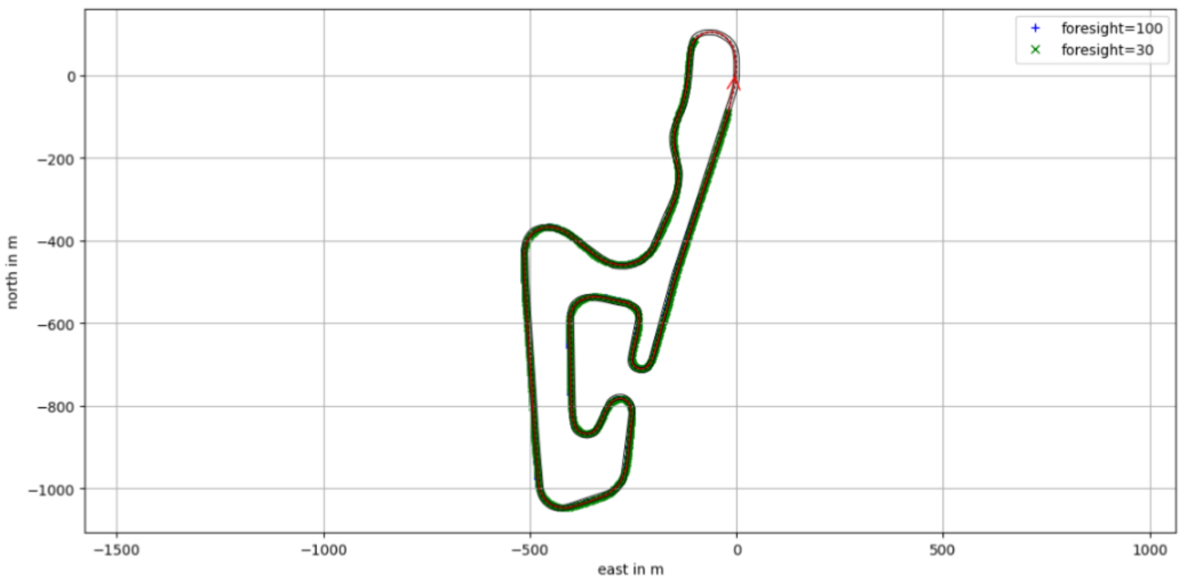


Figure 31: Okayama Prediction.

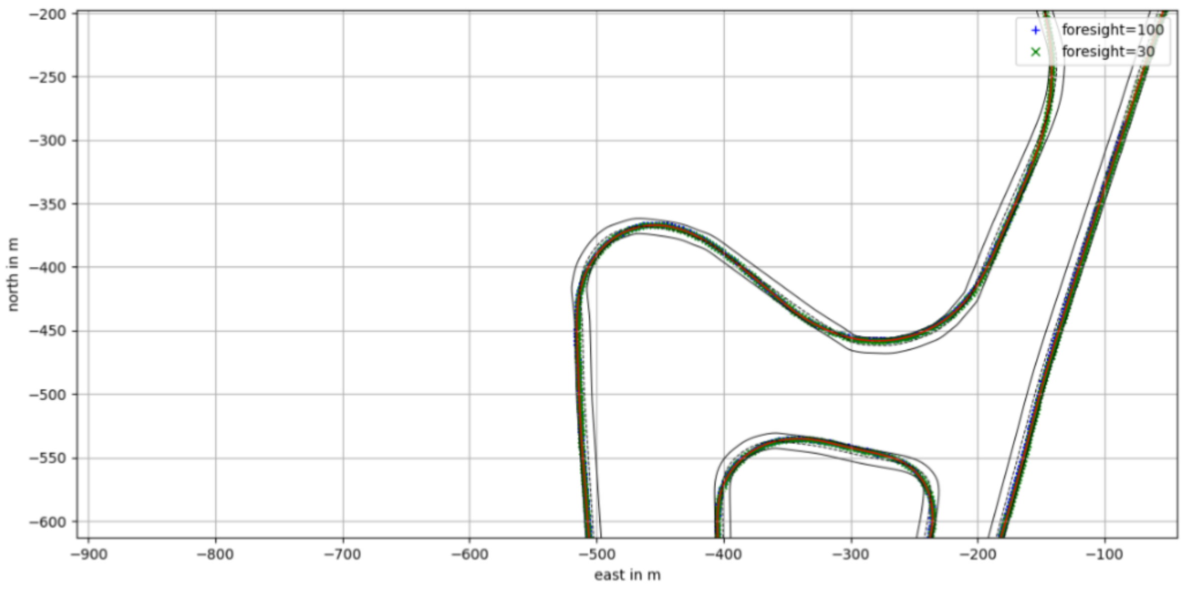


Figure 32: Okayama with zoom 1.

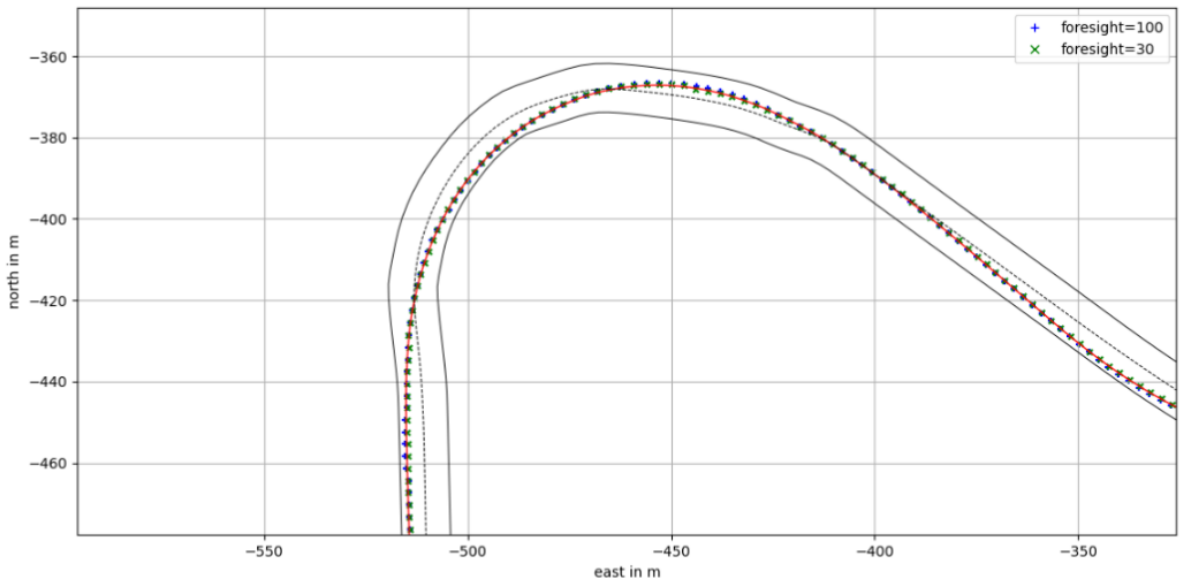


Figure 33: Okayama with zoom 2.

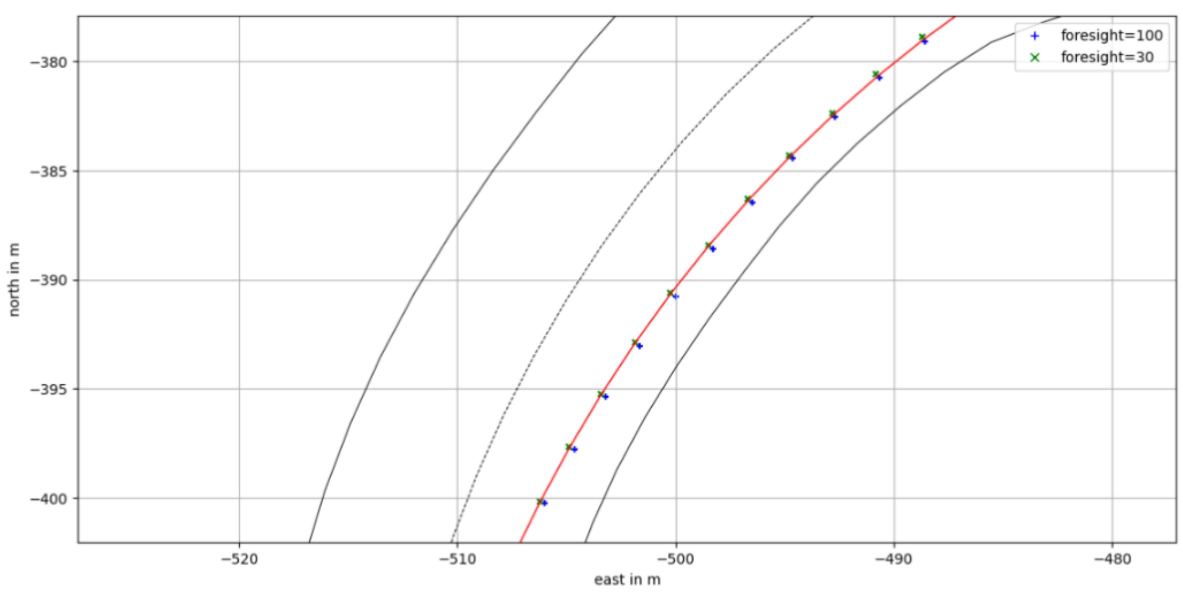


Figure 34: Okayama with zoom 3.

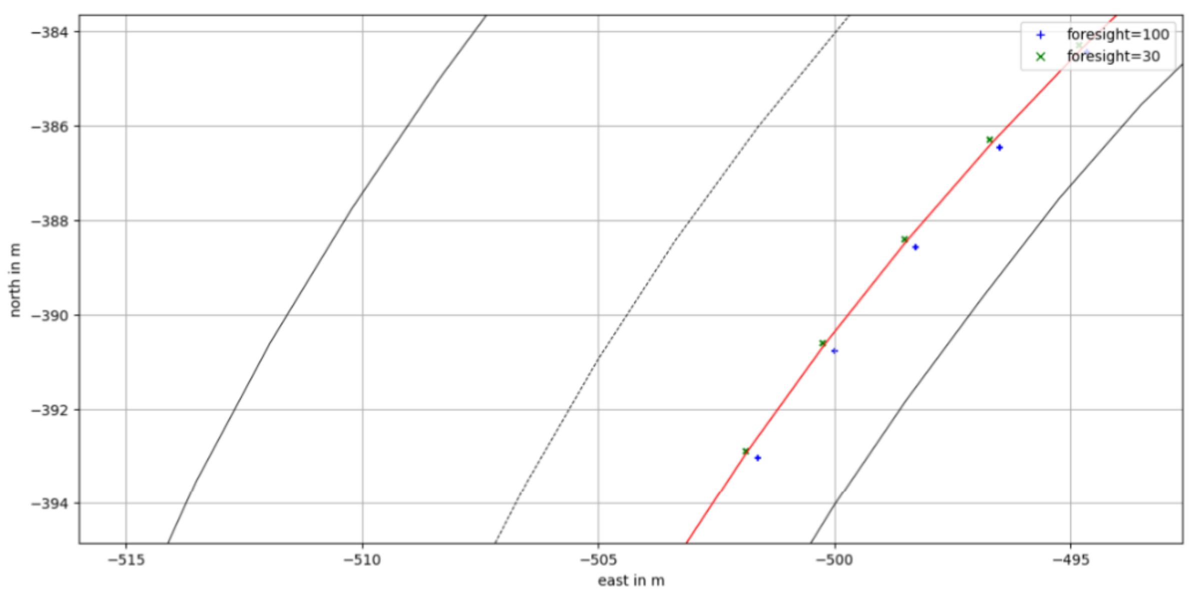


Figure 35: Okayama with zoom 4.

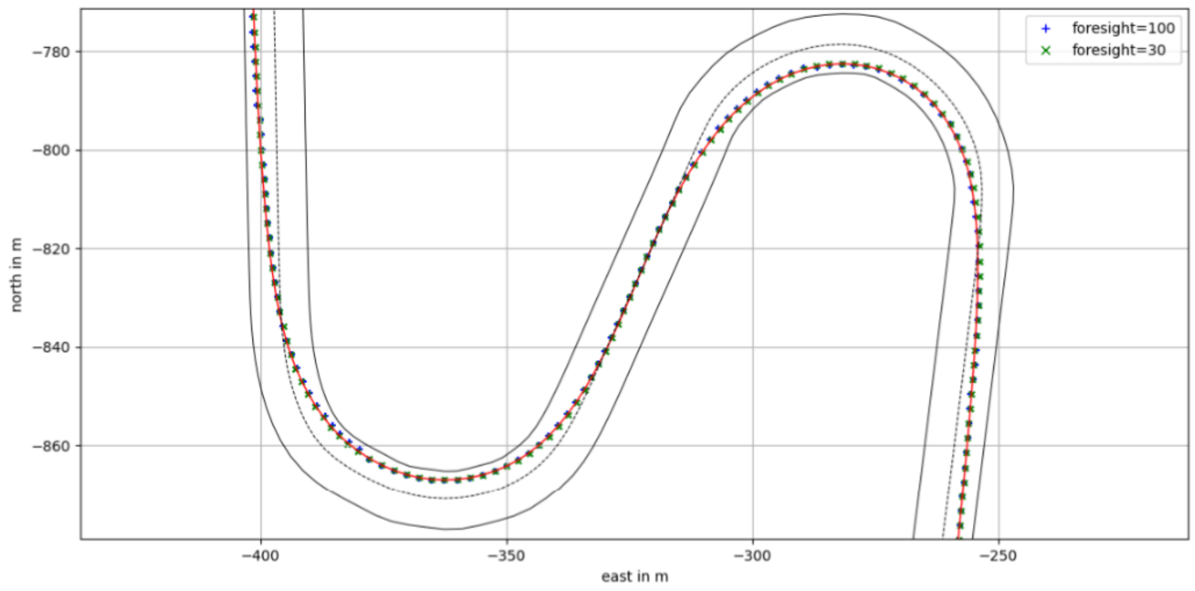


Figure 36: Okayama with zoom 5.

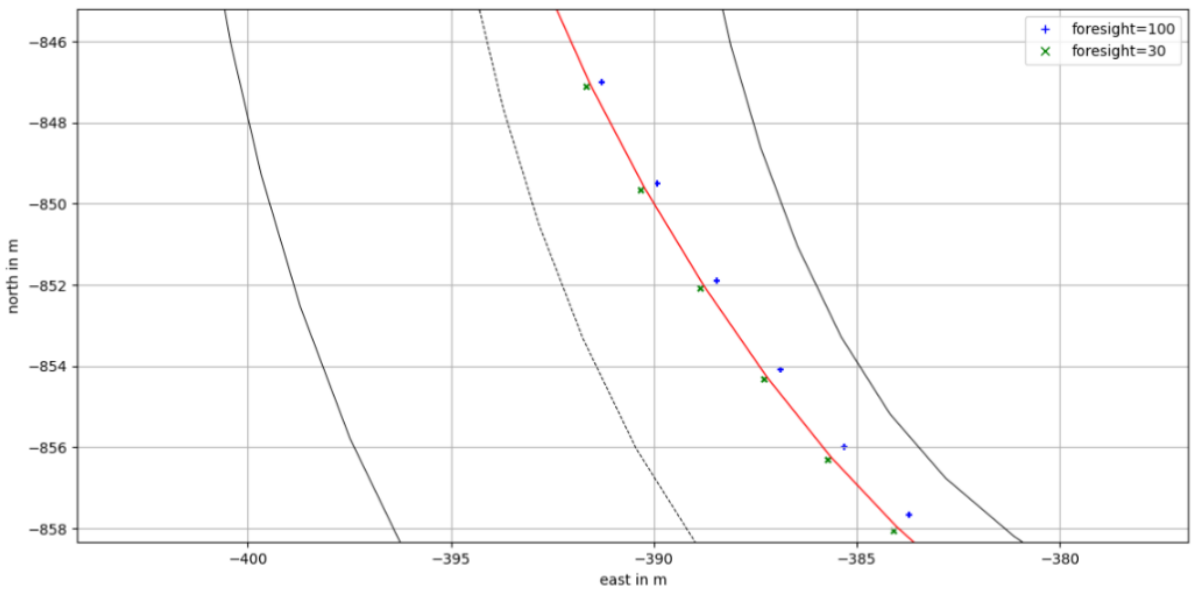


Figure 37: Okayama with zoom 6.

Lastly, additional plots of previously mentioned Mugello track are presented, showcasing predictions in different bends of the circuit (see Figures 38 to 42).

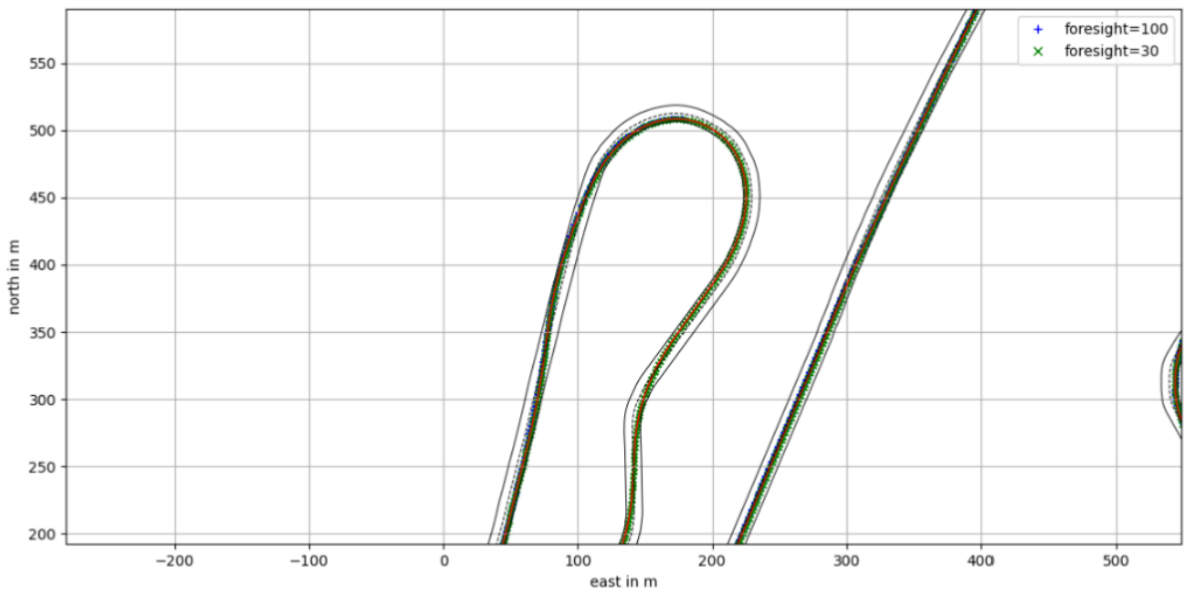


Figure 38: Mugello with zoom 1.

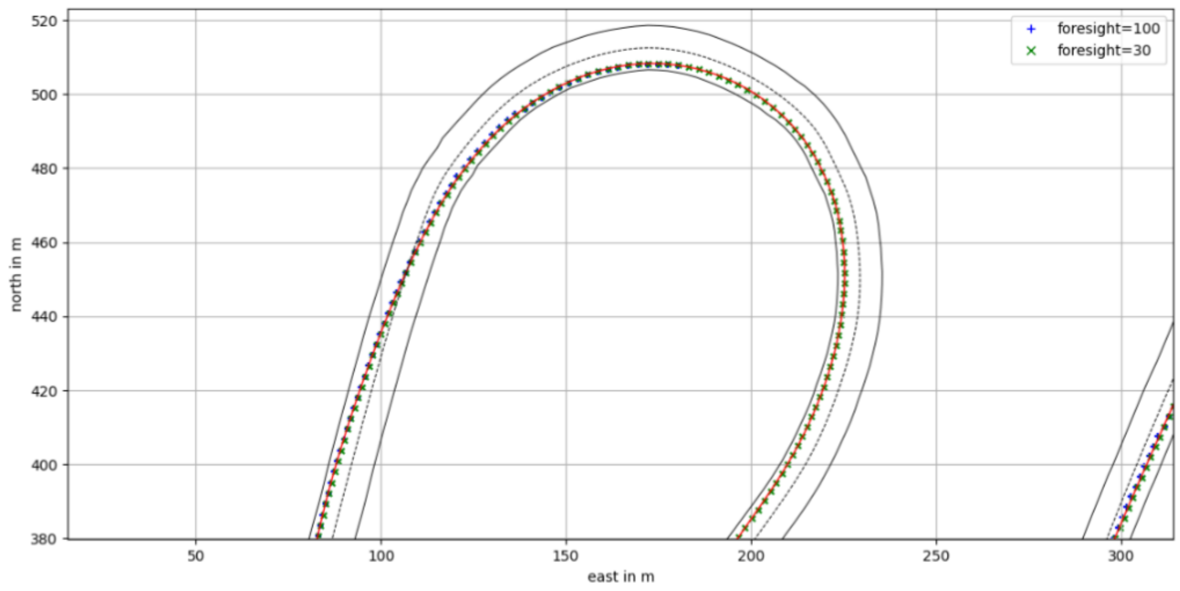


Figure 39: Mugello with zoom 2.

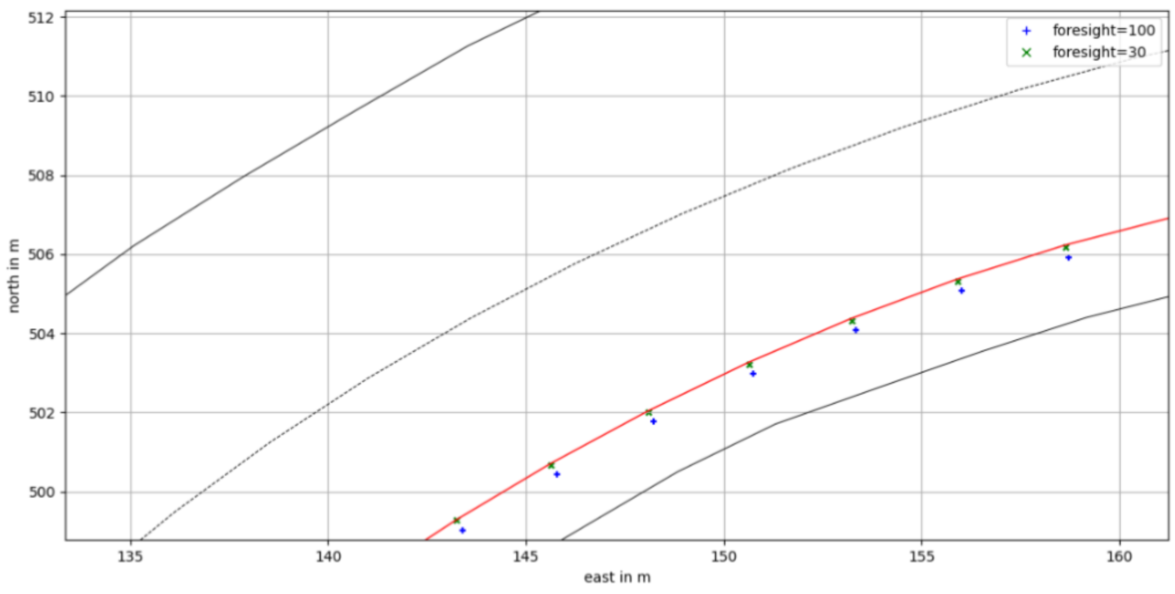


Figure 40: Mugello with zoom 3.

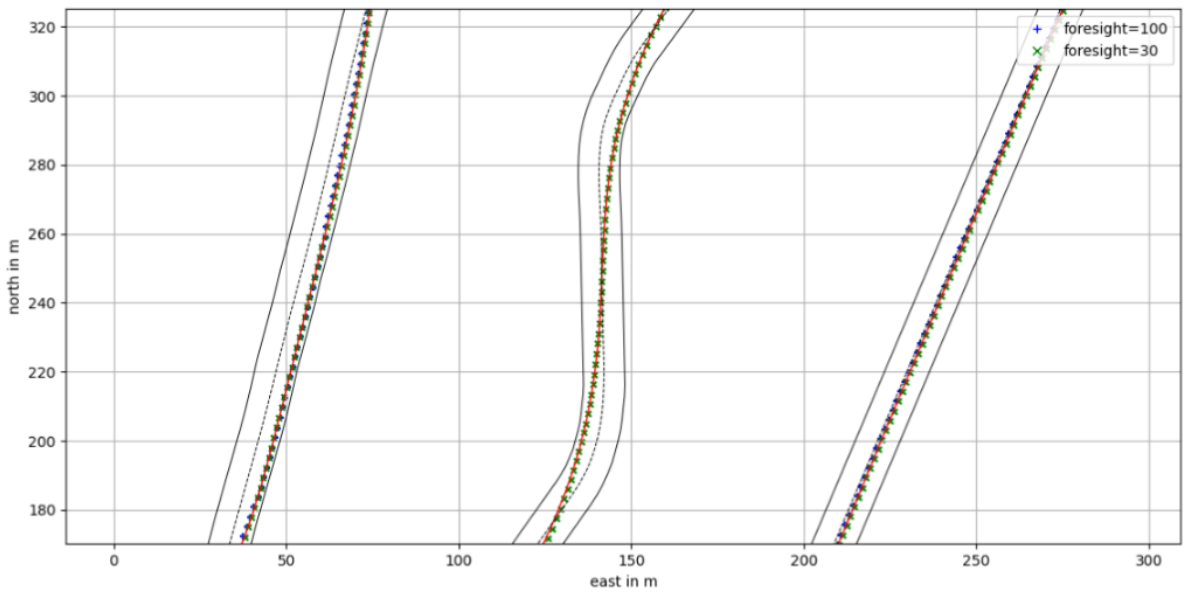


Figure 41: Mugello with zoom 4.

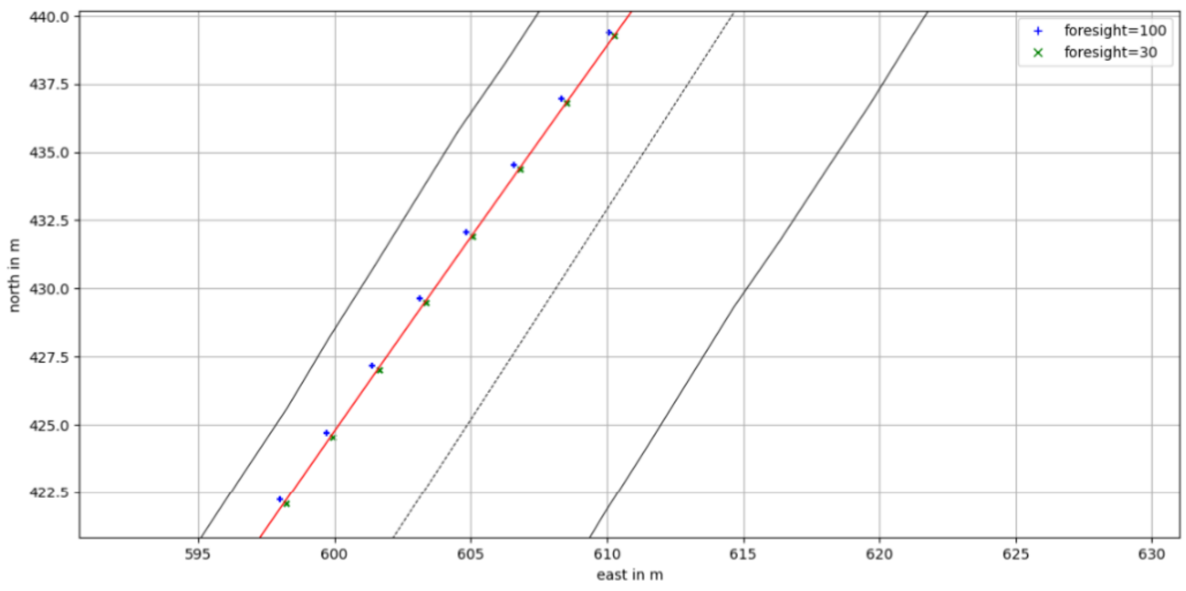


Figure 42: Mugello with zoom 5.