POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Behaviour cloning of a Model Predictive Controller for Path Tracking applications

Supervisors

Prof. Alessandro VIGLIANI

Prof. Umberto MONTANARO

Prof. Manuel FERRE PÉREZ

Candidate

Marino Massimo COSTANTINI

March 2024

Politecnico di Torino

1859

UNIVERSITY OF SURREY

UNIVERSIDAD POLITÉCNICA DE MADRID

Summary

Over the past few years, the University of Surrey has undertaken successful research in vehicle systems controls. Particularly, investigations have focused on implementing control strategies using scaled sensor-rich autonomous vehicles (SSRAVs) as experimental platforms, specifically emphasising path-tracking tasks. The preexisting controller architectures developed before the commencement of this study are essentially three:

- PID
- LQR
- MPC

Among these architectures, the Model Predictive Control (MPC) architecture has demonstrated superiority in terms of the key performance indicators (KPIs) taken into account. However, it is worth noting that a notable drawback of the MPC architecture is the considerable computational effort required. Indeed, to let this architecture work in real-time too, some simplifications got to be necessary. Real-time simplifications led to lower KPIs yet MPC was still the best architecture. In brief, the focus of this Master's thesis is to assess the viability of replacing MPC with an MPC-inspired Artificial Neural Network (ANN) to enhance computational efficiency and, consequently, achieve a higher level of overall real-time performance. The type used in this work is the Multi-Layer Perceptron (MLP) which is relatively straightforward compared to some other types of neural networks, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs). Nonetheless, CNNs and RNNs solutions may be considered as potential future developments of this work, since they might lead to comparable results.

Acknowledgements

To my family, my friends and whoever spontaneously and positively enriches this marvellous life.

"Audentes fortuna iuvat"
Publius Vergilius Maro

Table of Contents

List of Tables

List of Figures

Acronyms

ACADO	Toolkit for Automatic Control and Dynamic Optimisation
ANN	Artificial Neural Network
CoG	Centre of Gravity
DAE	Differential-Algebraic Equation
DNN	Deep Neural Network
e.g.	Exempli Gratia
FB	Feed-Back
FF	Feed-Forward
GD	Gradient Descent
GUI	Graphical User Interface

IACA	Integral of the Absolute Value of Control Action Delta
i.e.	Id Est
KPI	Key Performance Indicator
LQR	Linear Quadratic Regulator
LMPC	Linear Model Predictive Control
LTI	Linear Time Invariant
ME	Maximum Lateral Error
MLP	Multi Layer Perceptron
MPC	Model Predictive Control
NMPC	Non-linear Model Predictive Control
NNBO	Neural Network Based Optimisation
ODE	Ordinary Differential Equation
PID	Proportional Integrative Derivative

PP	Path Planning
PT	Path Tracking
RMSE	Root Mean Square Error of the Lateral Error
SSRAV	Scaled Sensor-Rich Autonomous Vehicle
w.r.t.	With Respect To

List of Definitions

ACADO Toolkit

ACADO Toolkit is a software environment and algorithm collection for automatic control and dynamic optimisation. It provides a general framework for using a great variety of algorithms for direct optimal control, including model predictive control, state and parameter estimation and robust optimisation. ACADO Toolkit is implemented as self-contained C++ code and comes along with user-friendly MATLAB interface. The object-oriented design allows for convenient coupling of existing optimisation packages and for extending it with user-written optimisation routines. Learn more about the features of ACADO Toolkit.[1]

Multi-Layer Perceptron

The simplest type of feed-forward ANN. The units are arranged into a set of layers, and each layer contains some number of identical units. Every unit in one layer is connected to every unit in the next layer; we say that the network is fully connected. The first layer is the *input layer*, and its units take the values of the input features. The last layer is the *output layer*, and it has one unit for each value the network outputs. All the layers in between these are known as *hidden layers*, because we don't know ahead of time what these units should compute, and this needs to be discovered during learning. The units in these layers are known as input units, output units, and hidden units, respectively. The number of layers is known as the *depth*, and the number of units in a layer is known as the *width* [2].

Overfitting

Overfitting occurs when an algorithm fits too closely or even exactly to its training data, resulting in a model that can't make accurate predictions or conclusions from any data other than the training data [3].

Path

Spatial construction that describes a sequence of steps that a physical body lies in or should lie in. It does not provide specific information about the timing at which each step is occupied. In this work the SSRAV will follow a path since there are no information concerning the time-scheduling.

Trajectory

The path that an physical body follows or should follow through space as a function of time. In this work, the trajectory will be the result of the path-tracking task that the SSRAV has performed thanks to the control architecture.

Underfitting

Underfitting is a scenario in data science where a data model is unable to capture the relationship between the input and output variables accurately, generating a high error rate on both the training set and unseen data [4].

Chapter 1

Introduction

The primary goal of this study is to explore a straightforward approach to address the real-time limitations of a Model Predictive Control (MPC) in a Path Tracking (PT) task applied to a Scaled Sensor-Rich Autonomous Vehicle (SSRAV). The intention is to evaluate this approach in both simulation and real-time deployment scenarios as done in chapters 6 and 7. The motivation for this investigation arose when it was recognised that the MPC, initially developed and fine-tuned in a simulation environment through sensitivity analysis, might face challenges in real-time application due to the hardware limitations as shown in the chapter 3.
Specifically, the computational real-time in-feasibility emerged when the prediction horizon encountered limitations imposed by the SSRAV hardware. This was despite the internal model, briefly described in the subsection 3.1.1, being very light.
Thus the computational limitations of the hardware in the SSRAV prompted a necessary downsizing of the optimisation problem. Indeed, despite the initial suggestion from sensitivity analysis for an optimal prediction horizon of 40 steps, practical constraints led to a reduction to 20 steps as resumed in the table 3.2. This downsizing further required the linearisation of the prediction model, with the exclusion of longitudinal speed from the control variables, resulting in a Linear Time-Invariant (LTI) internal model. This LTI model is valid only in the proximity of the chosen linearisation speeds, effectively transforming the controller into an LMPC.
The primary objective of this study is to replicate the non-implementable MPC using a Neural Network-Based Optimisation (NNBO) technique. Following a concise review of the state of the art in Section 2.4, a pure regression technique utilising a Multi-Layer Perceptron (MLP) was chosen to execute basic input-output behaviour cloning. The goal is to minimise the regression error and, consequently, demonstrate comparable behaviour with the expert controller through the key performance indicators (KPIs) detailed in Subsection 3.2.

As elaborated in chapter 4, a systematic approach involved the development of an entire toolkit from scratch. This toolkit serves as an easy-to-use program designed to guide users through the entire process. The steps can be summarised as follows:

I. Creation of the Set of LMPCs: For each chosen working speed a dedicated Linear MPCs (LMPC) was created. It is crucial that the set comprises LMPCs with prediction horizons of the same length, ensuring consistency among dataset rows. A discussion about the length of the final prediction horizons and working speeds chosen is presented in section 3.1.3.

II. Creation of the Dataset: Data was saved by running the expert LMPCs in closed-loop simulations along various representative paths called 'training paths'. The criteria guiding the path selection are explained in Section 3.5.

III. Creation, Setup, and Training of MLPs: Special attention was given to speed up the process, as detailed in Section 4.1. Here, a general tool chain was introduced for the swift training and validation of MLPs using MATLAB and a Python environment integration. This tool-chain can be also employed for general purposes, indeed it makes an additional outcome of this work. Further insights into the algorithm it utilises are provided in Section 3.7.

IV. Simulation Deployment of MLPs: The MLPs were deployed in a simulation environment, and subsequent KPI evaluations were conducted.

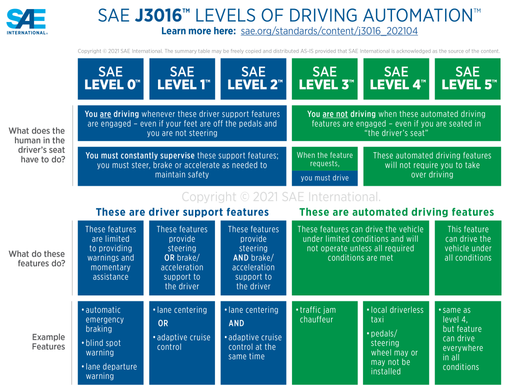
V. Experimental Deployment of MLPs: The MLPs were deployed experimentally, and subsequent KPI evaluations were conducted.

The development of the aforementioned comprehensive toolkit proved to be indispensable, particularly when unexpected results emerged during the experiments. This toolkit enabled a swift and efficient redesign of the experimental setup, significantly streamlining the process. Consequently, the ability to quickly iterate on the setup not only facilitated smoother experimentation but also expedited the identification and resolution of issues encountered.
Furthermore, as outlined in section 3.4, meticulous attention was devoted to reviewing past work. This thorough examination uncovered errors in calculations and provided valuable insights for improvement. Unfortunately, lack of time limited the implementation of all suggested fixes.

Chapter 2

State of the Art

Autonomous driving is a highly demanding sector in the automotive industry, requiring the exploration of innovative techniques that leverage advanced sensor arrays. The Society of Automotive Engineers (SAE) has categorised autonomous vehicles into six levels, spanning from Level 0 (requiring full manual operation) to Level 5 (fully autonomous operation). As automation levels advance, ensuring the safety of autonomous vehicles within dynamic traffic environments becomes a top priority for stakeholders across academia, industry, research, and policymaking. Engineers and researchers are actively working to enhance safety measures in response to these concerns. Notably, the Safety of the Intended Functionality (SOTIF) guidelines detailed in ISO/PAS-21448 cover both the hardware and software aspects of autonomous vehicles. These guidelines aim to achieve comprehensive situational awareness and mitigate the potential for system failures, thereby promoting safer autonomous driving experiences. [5].



SAE **J3016**™ LEVELS OF DRIVING AUTOMATION™
Learn more here: sae.org/standards/content/j3016_202104

	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	**You <u>are</u> driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You <u>are not</u> driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 2.1: SAE Levels of Driving Automation™ Refined for Clarity and International Audience [6].

2.1 Scaled Sensor-Rich Autonomous Vehicle

Mobile robotic systems have garnered significant interest from industries, research institutions, and governments worldwide, driven by the widespread adoption of Industry 4.0 [7]. These systems find application across diverse sectors, including laboratories, industries, warehousing, and transportation [8]. SSRAVs are notably employed as assessment platforms for autonomous driving, as highlighted in [9], [10], and [11]. They offer several advantages:

I. They enable cost-effective testing, leading to significant cost savings compared to experiments involving full-sized vehicles.

II. They provide a repeatable testing environment, facilitating more precise tuning of control algorithms.

III. They enhance safety, as scaled robotic cars entail less severe consequences in the event of system failures or collisions.

Consequently, these mobile robots serve as versatile platforms for evaluating Path Planning (PP) [12], [13], Path Tracking (PT) [14], and vehicle platooning algorithms [15]. PT controllers, in particular, serve as intermediaries between vehicle dynamics and the path planner of automated vehicles. They define actuator commands, typically steering angle and traction/braking torques, to follow predefined paths while adapting to dynamic and uncertain real-world conditions [16], [17], [18]. SSRAVs play a crucial role in experimentally evaluating the effectiveness of various PT solutions within this framework.

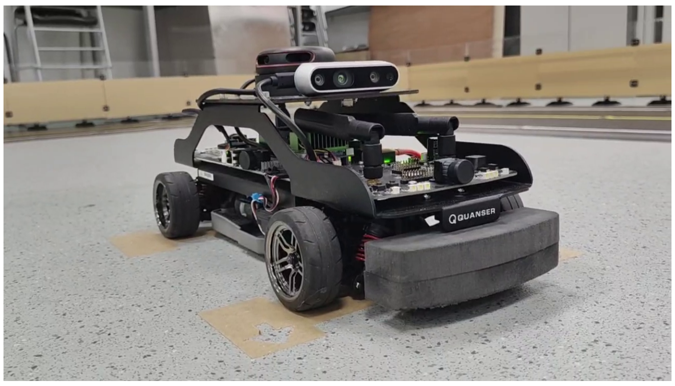


Figure 2.2: One of the QCars that the laboratory of the University of Surrey is provided with.

2.2 Path tracking control solutions

Since the performance of PT controllers depends on factors such as the complexity of the vehicle model and the accuracy and robustness of the control scheme, the design and implementation of PT controllers for automated vehicles have been the subject of numerous studies [18] [19]. Proposed control strategies in the literature range from geometry-based (pure pursuit and Stanley controller) to robust controllers (sliding mode controller and H) and optimisation-based controllers (LQR and MPC) [18] [19].

MPC techniques, among the latter two categories, are the most widely used for trajectory tracking due to their ability to handle multi-variable problems and systematically consider constraints on states and control actions, as well as predict the expected future behaviour of the system [17]. In the realm of model-based strategies, examples in the literature include optimal robust linear matrix inequality-based MPC [20] focusing on minimising battery consumption, and MPC based on a Space-Time Model (STM) introduced in [21] [22] to optimise vehicle speed profiles and the reference path. Additionally, an efficient and robust MPC for trajectory tracking of a small-scale autonomous bulldozer is presented in [23], demonstrating the capability of the proposed controller to track target trajectories with low processing time and small tracking errors.

Despite the significant advancements in path tracking performance achieved by these control methods, they still face several challenges [18] [24]:

I. Design complexity is related to the necessity of incorporating vehicle non-linearities.

II. Difficulty in obtaining a suitable set of tuning parameters ensuring consistent performance across a broad range of potential scenarios.

III. Demanding state and parameter estimation requirements.

IV. Robustness with respect to variations in vehicle and environmental parameters. Moreover, the real-time implementation of implicit MPC formulations is computationally demanding.

To address these challenges (i-iv) recent machine-learning solutions have been proposed. Among these, imitation learning (IL) leverages the universal function approximation capability of artificial neural networks (ANN) [25] [26]. It involves training networks to imitate the control action computed by another control strategy. The resulting ANN can possess the same control capability and strong robustness as the original controller but with much lower computational complexity. This work aims to address the limitations of MPCs, as reviewed in the next section.

2.3 MPC & associated issues

Presently, the attributes of MPCs, encompassing both their merits and limitations, are widely acknowledged by the majority. MPC was already emerging into the process industries during the 1980's. Today it is commonplace and, in some sense, mature [27]. Moreover, commercial packages are available on a variety of platforms and "do-it-yourself" implementations are also readily found (e.g. ACADO toolkit). In short, MPC is a controller that makes predictions thanks to an internal model of the system to control. A more specific and common formulation of the optimisation problem is as follows:

Find a set of manipulated variables that minimises a loss function of future predicted control errors subject to constraints on both manipulated and controlled variables.

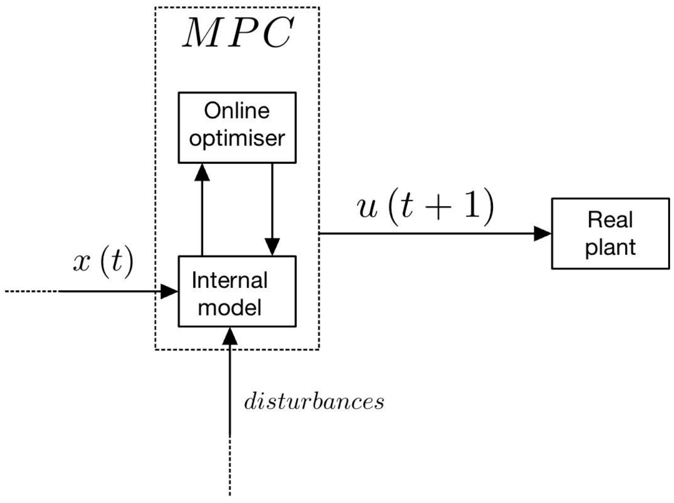


Figure 2.3: MPC schematic. The MPC is implicit, then the optimisation is performed online. This is a non-single evaluation algorithm then the computational cost increases together with the number of iteration that the algorithm takes.

The general formulation of MPC involves minimising a cost function J over a finite time horizon N:

$$J = \sum_{k=0}^{N-1} \ell(x_k, u_k) + V(x_N), \tag{2.1}$$

where x_k is the state vector, u_k is the control input, ℓ is the stage cost function, and V is the terminal cost function.
The stage cost function $\ell(x_k, u_k)$ is often defined as a quadratic form:

$$\ell(x_k, u_k) = (x_k - x_{\text{ref},k})^\top Q(x_k - x_{\text{ref},k}) + u_k^\top R u_k, \tag{2.2}$$

where Q is the state cost matrix, R is the control input cost matrix, and $x_{\text{ref},k}$ is the reference state at time k.
The terminal cost function $V(x_N)$ can be similarly defined as:

$$V(x_N) = (x_N - x_{\text{ref},N})^\top P(x_N - x_{\text{ref},N}), \tag{2.3}$$

where P is the terminal state cost matrix, and $x_{\text{ref},N}$ is the reference state at the terminal time N.
Subject to the system dynamics constraint:

$$x_{k+1} = f(x_k, u_k), \tag{2.4}$$

and input and state constraints, which can be either hard or soft:

$$u_k \in \mathcal{U}, \quad \text{(Hard or Soft)} \tag{2.5}$$

$$x_k \in \mathcal{X}, \quad \text{(Hard or Soft)}, \tag{2.6}$$

where $\mathcal{U}$ and $\mathcal{X}$ are the input and state constraint sets, respectively.
Hard constraints must be satisfied at every time step, while soft constraints contribute to the cost function and can be violated within a specified tolerance.
Additionally, output constraints can be considered:

$$y_k = Cx_k, \tag{2.7}$$

with constraints:

$$y_k \in \mathcal{Y}, \quad \text{(Hard or Soft)}, \tag{2.8}$$

where C is the output matrix and $\mathcal{Y}$ is the output constraint set.

The optimisation problem is typically solved online (implicit MPC) at each time step, and only the first control input is applied to the system. The process is then repeated at the next time step with updated measurements.
Still in brief, summarising the features of MPCs in comparison with conventional controllers: [28]

- Benefit: A well-designed MPC excels in effectively operating within constraints of the real actuator, model uncertainty, and non-linearity.
- Drawback: The complexity of MPC's algorithm results in a longer processing time compared to other controllers, primarily due to real-time optimisation challenges .

The aforementioned drawback poses an inevitable constraint on the hardware suitable for deploying MPC. Specifically, when employing a micro-controller, the feasible set of MPCs for real-time applications is restricted and contingent upon two key factors:

1. Power of the hardware
2. Computational cost imposed by the MPC

As a consequence, to address this limitation, the choice of hardware must either lean towards more powerful alternatives, or the optimisation problem must be re-evaluated and streamlined (e.g., by shortening the prediction horizon, utilising a more efficient internal model, reducing the number of iterations in the optimisation process, or relaxing certain hard constraints).

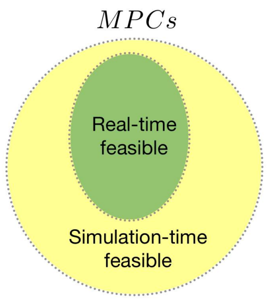


Figure 2.4: MPC real-time feasibility.

2.4 Neural Network-Based Optimisation

The literature offers numerous strategies to overcome these limitations, with a particular focus on methods that avoid the necessity of online optimisation, a significant contributor to real-time impracticality. Many of these strategies leverage machine learning techniques during the offline phase. Among them, a straightforward approach involves training an NN through pure regression to mimic the outputs of a traditional optimiser, commonly referred to as imitation learning (IL).

At its essence, imitation learning aims to acquire expertise by observing and learning from expert demonstrations. Recent advancements in data-driven MPC often utilise a basic form of imitation learning known as behaviour cloning. This technique seeks to train controllers that replicate the performance of MPC by sampling trajectories from the closed-loop MPC system in real-time. However, behaviour cloning is recognised for its inefficiency in data utilisation and susceptibility to distribution shifts.

In [29], the use of deep neural networks (DNNs) is simulated, where a DNN controller is trained on simulated input-output data from a well-designed MPC. Subsequently, the expert controller is substituted by the artificial neural network itself. Similarly, in [30], an MPC law is approximated by a DNN that is computationally efficient for online evaluation and has a small memory footprint, making it suitable for embedded applications. In another study [31], it is proposed to learn the optimal control policy defined by a complex model predictive formulation using deep neural networks offline, enabling the online use of the learned controller requiring only the evaluation of a neural network. Another example is provided by [32], where the MPC regulator is approximated offline using a feed-forward MLP.

However, the aforementioned pure regression methods overlook the complexity of preserving the theoretical properties and guarantees of MPC and necessitate careful validation through simulation and experimental validations. The common issue with approaches lacking guarantees is the inability to ensure the safety of the controller by design.

Despite their differences, all the aforementioned methodologies converge on addressing a regression task. NNs are recognised as universal function approximators [33]. However, achieving precise regressions practically proves to be challenging or even infeasible, inevitably resulting in regression errors. With a sufficiently complex network architecture and abundant training data, it becomes feasible to significantly mitigate these regression errors.

Nevertheless, as highlighted by Akesson et al. [34], the regression error can escalate over time when the network operates in closed-loop control. Consequently, the performance of an approximate MPC controller may substantially deviate from that of an exact controller. Hence, approaches lacking assurances necessitate additional safety mechanisms operating concurrently with the controller. These mechanisms undertake tasks such as controlling action saturation, detecting constraint violations, identifying instability, and executing emergency stops. While such setups may suffice for certain practical applications, such as managing cost-effective unmanned systems in low-risk environments, they may not meet the requirements of more critical systems where the theoretical assurances of MPC are indispensable.

The literature also explores methodologies capable of providing guarantees. For instance, the approach by Ahmed et al. [35] ensures input constraint satisfaction through saturation and stability at equilibrium. However, guarantees for state and output constraint satisfaction, as well as stability during regulation, remain unassured. In this study, instead of training the NMPC policy via regression, the authors developed an ANN loss function aligned with the cost function of a regulating NMPC controller. This enabled training the ANN via back-propagation, facilitated by computing the necessary derivatives using block partial derivatives. Notably, the authors introduced a crucial constraint to ensure stability at the equilibrium. Specifically, they aimed to train a network that produces no output when the system is at equilibrium. They achieved this by designing the ANN architecture as DNN devoid of bias terms and solely employing the tanh activation function. Thus, if all inputs to the network were zero, the outputs would also be zero, similar to a linear control law with no offset.

Figure 2.5 illustrates a typical implementation of Neural Network-Based Optimisation (NNBO) to replace an MPC controller.

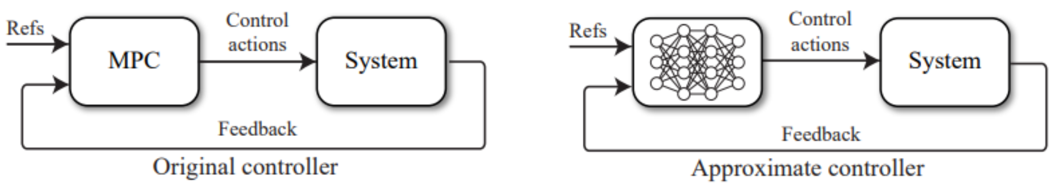


Figure 2.5: Typical implementation of NNBO to replace an MPC controller.

Chapter 3

Methodologies

This chapter outlines the methodologies employed for each aspect of this Master's thesis. The specifications of both benchmark MPCs and expert MPCs are listed in tables 3.1 and 3.2, including the internal model taken into account, which is shown in subsection 3.1.1, together with a small introduction to the ACADO toolkit (i.e., the external toolkit used to formulate the MPC optimisation problem) in 3.1.2. Moreover, the KPIs used will be described, and their use will be motivated in section 3.2. Also, particular attention is reserved for the set of expert LMPCs created and the shape of the dataset, respectively in sections 3.1.3 and 3.7. One of the goals was to keep all the code clear; in this regard, the MathWorks Automotive Advisory Board (MAAB) [36] was taken into account. Additionally, all the previous code made by previous candidates was reviewed, now knowing the mentioned guide. Finally, the algorithms used in the toolchain to build the MLP will be expressed in section 3.7.

3.1 Model Predictive Problem formulation

As the Model Predictive architecture is not the primary focus of this work, this subsection will be kept brief. The table below presents the features of the downsized version of the MPC deployed in the experimental validation. In this study, the real-time feasible LMPC is referred to as the benchmark LMPC.

Real-time feasible LMPC (benchmark)	
Internal Model:	Single-Track dynamical lateral model
Sample time internal model:	$0.01s$
Discretisation time to solve equations:	$0.001s$
Prediction Horizon:	20 steps
Disturbances (online data):	Curvature of the path along the prediction horizon $\mathbf{k} = \begin{bmatrix} k_0 & k_1 & k_2 & ... & k_{19} & k_{20} \end{bmatrix} m^{-1}$
Number of Iterations:	3
Linearisation speeds (one for each LMPC):	$\mathbf{v} = \begin{bmatrix} 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1.0 & 1.2 & 1.5 \end{bmatrix} \frac{m}{s}$

Table 3.1: Experimentally tested LMPCs configuration.

In this study, LMPCs impractical for real-time use are termed expert LMPCs. The data to train LMLPs is derived from these controllers. Without the need for real-time implementation, both the prediction horizon and the number of iterations (N) double from 3 to 6. Speeds considered range from 0.5 m/s to 1.5 m/s in 21 values with a fixed 0.05 m/s step, aiming to precisely capture linear model variations from the linearisation process.

Real-time feasible LMPC (benchmark)	
Internal Model:	Single-Track dynamical lateral model
Sample time internal model:	$0.01s$
Discretization time to solve equations:	$0.001s$
Prediction Horizon:	40 **steps** $\rightarrow$ **doubled**
Disturbances (online data):	Curvature of the path along the prediction horizon $\mathbf{k} = \begin{bmatrix} k_0 & k_1 & k_2 & \dots & k_{39} & k_{40} \end{bmatrix} m^{-1}$
Number of Iterations:	9 $\rightarrow$ **tripled**
Linearization speeds (one for each LMPC):	$\mathbf{v} = \begin{bmatrix} 0.50 & 0.55 & 0.60 & \dots & 1.40 & 1.45 & 1.50 \end{bmatrix} \frac{m}{s}$

Table 3.2: Simulated tested LMPCs configuration.

3.1.1 Internal model

When dealing with path-tracking tasks, lateral dynamics play a fundamental role. Here, the single-track lateral dynamical model is used as an internal model of the LMPC. Moreover, a change of variables has been made to better address the automatic lane-keeping task.
The following figures illustrate the system of reference and how the mobile frame is attached to the SSRAV. It is also important to define the system of reference

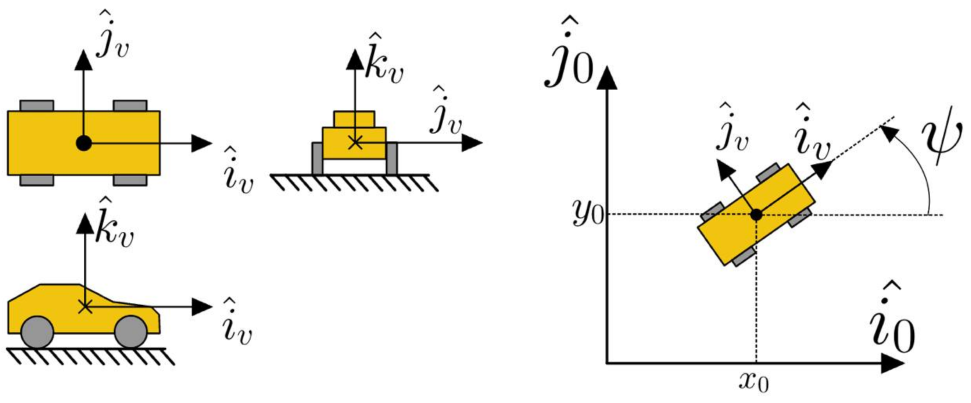


Figure 3.1: Vehicle frame and fixed frame.

for the wheels, especially the steering ones. This will prove useful in defining the single-track model used as the internal model reference inside the MPC to solve the optimal problem.

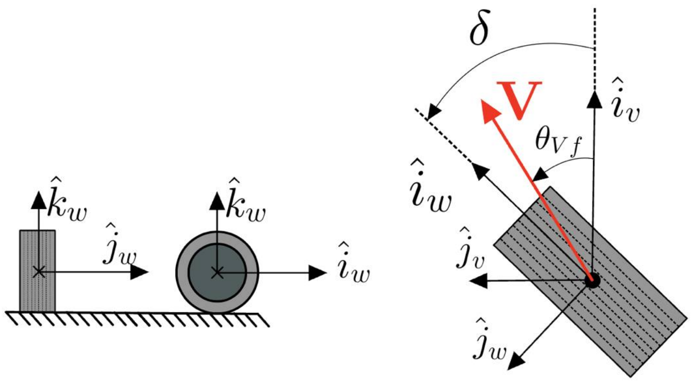


Figure 3.2: Wheel frame and vehicle frame.

In the context of the single-track model, the SSRAV is conceptualised as a single-body mass capable of motion in a three-degree-of-freedom plane. The system is characterised by a state vector $\mathbf{x}$ with dimensions 4×1, encompassing the lateral displacement (y), lateral velocity ($\dot{y}$), yaw angle (ψ), and yaw rate ($\dot{\psi}$). Notably, this model omits the inclusion of longitudinal states (x and $\dot{x}$). This omission is founded on the assumption of independence between longitudinal and lateral dynamics. Consequently, we can presume either a constant longitudinal velocity or utilise an alternative model to describe the longitudinal dynamics of the SSRAV. Thus the state vector $\mathbf{x}$ is mathematically expressed as:

$$\mathbf{x} = \begin{bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{bmatrix}$$

where:

- y is the lateral displacement w.r.t. the SSRAV frame.
- $\dot{y}$ is the lateral velocity w.r.t. the SSRAV frame.
- ψ is the yaw angle represented in figure 3.1.
- $\dot{\psi}$ is the yaw rate.

While the input $\mathbf{u}$ is mathematically expressed as:

$$\mathbf{u} = \begin{bmatrix} \delta \end{bmatrix}$$

where:

- δ is the steering angle represented in figure 3.2.

The dynamics of the linearised system then can be represented by the following linear state space:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

At this juncture, as previously indicated in the section's preamble, a change of variables becomes instrumental for an enhanced control strategy. Particularly in the context of PT problems, where a predefined path is given, it is advantageous to express all quantities in terms of errors with respect to the path. In this case, the control variables are defined as follows: lateral error (e_1), the derivative of lateral error ($\dot{e}_1$), yaw error (e_2), and yaw rate error ($\dot{e}_2$).

The error variables are given by:

$$e_1 = (\mathbf{p} - \mathbf{p}_{\text{ref}}) \cdot \hat{j}_v$$

$$e_2 = \psi - \psi_{\text{ref}}$$

Here, $\mathbf{p}$ represents the vehicle position in the fixed frame, $\mathbf{p}_{\text{ref}}$ is the projected point of the vehicle from the lateral perspective with respect to the path (i.e., the position the SSRAV should occupy at that instant), ψ is the yaw angle w.r.t. the fixed frame 3.1, and ψ_{ref} is the desired yaw angle with respect to the fixed frame, referenced to the inclination of the path at the point $\mathbf{p}_{\text{ref}}$.
These error variables provide a more intuitive representation for the control problem, aligning with the objectives of path tracking and facilitating a clearer understanding of the SSRAV's positional and orientation deviations from the desired trajectory.

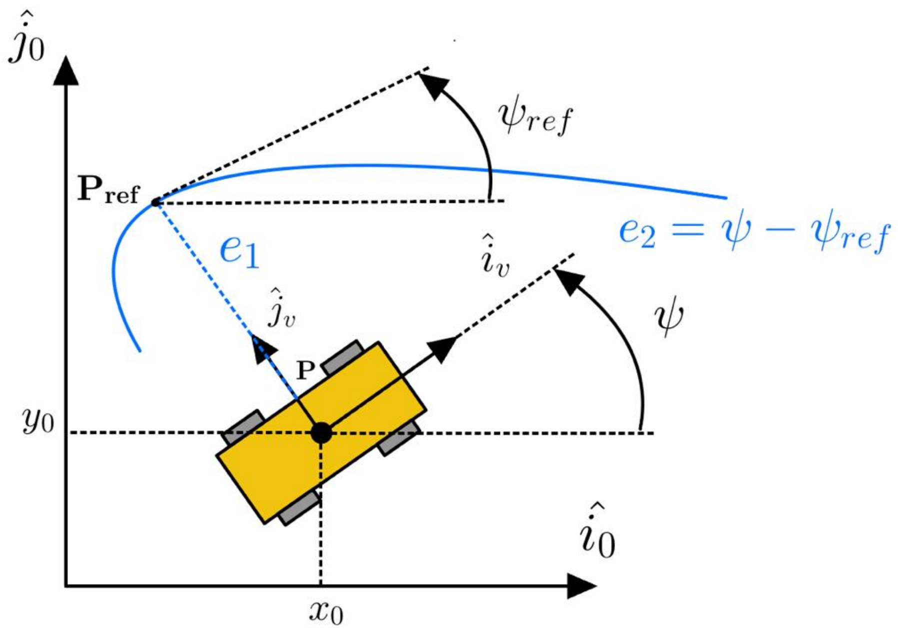


Figure 3.3: Vehicle frame and fixed frame with error state representation.

Thus the new state vector $\mathbf{e}$ is mathematically expressed as:

$$\mathbf{e} = \begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix}$$

While the input $\mathbf{u_e}$ is mathematically expressed as:

$$\mathbf{u_e} = \begin{bmatrix} \delta \\ \dot{\psi}_{ref} \end{bmatrix}$$

The dynamics of the linearised system then can still be represented by the following linear state space:

$$\dot{\mathbf{e}} = A\mathbf{e} + B\mathbf{u_e}$$

Below, the matrices A and B are explicitly defined. Passages are omitted for brevity as modelling is considered beyond the thesis scope.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{af}+2C_{ar}}{mV_x} & \frac{2C_{af}+2C_{ar}}{m} & \frac{-2C_{af}l_f+2C_{ar}l_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2C_{af}l_f-2C_{ar}l_r}{I_xV_x} & \frac{2C_{af}l_f-2C_{ar}l_r}{I_x} & -\frac{2C_{af}l_f^2+2C_{ar}l_r^2}{I_zV_x} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ \frac{2C_{af}}{m} & -\frac{2C_{af}l_f-2C_{ar}l_r}{mV_x} - V_x \\ 0 & 0 \\ \frac{2C_{af}l_f}{I_z} & -\frac{2C_{af}l_f^2+2C_{ar}l_r^2}{I_zV_x} \end{bmatrix}$$

Where:

- C_{af} represents the front tire cornering stiffness [N/rad].
- C_{ar} represents the rear tire cornering stiffness [N/rad].
- m is the vehicle mass [kg].
- V_x denotes the longitudinal speed (linearisation speed) [m/s].
- I_z stands for the vehicle inertia [kg·m^2].
- l_f is the distance from the Centre of Gravity (CoG) to the front tyre [m].
- l_r is the distance from the CoG to the rear tyre [m].

3.1.2 ACADO tool-kit

The implementation of the presented MPC was carried out in the Matlab/Simulink environment, leveraging the capabilities of the ACADO toolkit and its MATLAB interface.
The ACADO Toolkit stands as a comprehensive software environment and algorithm collection, coded in C++, tailored for dynamic optimisation and control design. It offers a versatile framework, accommodating various algorithms for direct optimal control, encompassing model predictive control, Runge-Kutta, and BDF integrator for simulating Ordinary Differential Equations (ODEs) and Differential-Algebraic Equations (DAEs).
Designed with openness and user-friendliness in mind, ACADO Toolkit is an open-source solution that operates in a self-contained manner, eliminating the need for external packages. To facilitate seamless integration with MATLAB/Simulink, ACADO for MATLAB serves as an interface. This interface facilitates the incorporation of ACADO integrator and algorithms for direct optimal control into the Matlab environment.
Key features of ACADO for Matlab include:

- Same properties as ACADO Toolkit: The interface inherits all the properties and functionalities of the ACADO Toolkit, introducing no new algorithms or additional features.

- No C++ knowledge required, MATLAB familiarity: Users do not need to possess any knowledge of C++—including its syntax or compiling rules—to utilise the interface. This aspect makes ACADO for MATLAB an accessible entry point for those comfortable with MATLAB but lacking C++ expertise. The interface adopts MATLAB-style notations and allows direct utilisation of variables and matrices stored in the Matlab workspace.

- Integration with Matlab black box models: While the ACADO Toolkit supports a symbolic syntax for expressing differential and algebraic equations, the interface enables the seamless connection of existing MATLAB black box models to the ACADO Toolkit.

- Cross-platform compatibility: The interface is compatible with various operating systems, including Windows, Mac, and Linux.

3.1.3 Creation of the set of LMPCs

The initial LMPCs, each with a 20-step prediction horizon, were developed to operate within a restricted range of specific speeds, detailed in Equation (3.1). It is noteworthy that the prediction horizon was halved from 40 to 20, emphasising the need for more efficient computational strategies.

$$\mathbf{v} = \begin{bmatrix} 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1.0 & 1.2 & 1.5 \end{bmatrix} \frac{m}{s} \tag{3.1}$$

The primary goal is to create a unified NNBO controller capable of continuous operation across the entire speed spectrum, from 0.5 m/s to 1.5 m/s, for enhanced usability. To achieve this, a model is proposed based on a set of LMPCs. This strategy enables the controller to emulate optimal performance across the complete operational range. Additionally, we aim to utilise the initially computed 40-step prediction horizon, identified as the most effective, to further enhance the controller's performance.
In essence, the set of LMPCs is configured to cover the entire specified speed range with a step size of 0.05 m/s, resulting in the creation of 21 expert controllers. The toolkit developed for this purpose, leveraging ACADO, proves instrumental in efficiently generating diverse MPC sets tailored to specific speed intervals.
How to deal with not extracted Q and R

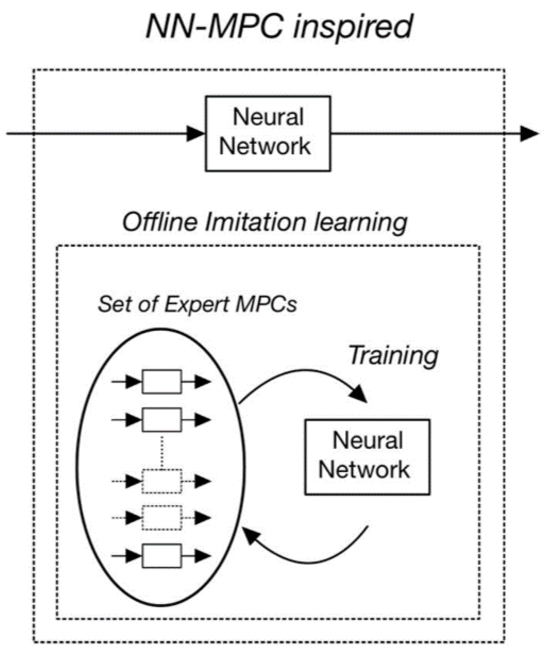


Figure 3.4: NNBO controller schematic.

3.2 Key Performance Indicators

Key Performance Indicators (KPIs) play a pivotal role in evaluating the efficacy of a control strategy. This significance is heightened, particularly when undertaking comparisons between diverse strategies, as is the focus of this study. In the context of PT tasks involving error states, the following KPIs are commonly utilised:

- Maximum Lateral Error (ME)
  This parameter is highly sensitive to any spikes that may occur within the system. Essentially, it measures the minimum smoothness of the signal, specifically the lateral error concerning the path.
- Root Mean Square Error of the Lateral Error (RMSE)
  Conversely, this parameter is more representative as it pertains to an entire set of measurements. In this case, the root mean square is intended to provide a value that represents the goodness of the signal overall. It is less sensitive to outliers such as spikes compared to the previous KPI.
- Integral of the Absolute Value of Control Action Delta (IACA)
  This value pertains to the displacement of the actuator. Similar to the RMSE, it is based on the entire set of signals. However, this value is also influenced by the type of path. For instance, if the path is a circle, the steering wheel necessarily needs to be displaced. Consequently, even if the path is well-tracked, the value of this KPI increases due to the integral being performed.

They are mathematically defined as follows:

$$\text{ME} = \max_{t \in [0, T_f]} |\Delta y(t)| \tag{3.2}$$

$$\text{RMSE} = \sqrt{\frac{1}{T_f} \int_0^{T_f} |\Delta y(t)|^2 \, dt} \tag{3.3}$$

$$\text{IACA} = \frac{1}{T_f} \int_0^{T_f} |\delta(t)| \, dt \tag{3.4}$$

where T_f is the duration of the manoeuvre.

3.3 MAAB Guidelines

The MathWorks Automotive Advisory Board (MAAB) guidelines align seamlessly with this work [36]. Although the project was led by a Master's student and not shared with others, adhering to standardisation practices remains advantageous and illustrates good habits, particularly given the project's automotive focus. MathWorks underscores the significance of these guidelines for success and teamwork, both internally and when collaborating with partners or subcontractors. Therefore, following these guidelines is essential for achieving:

- System integration without problems
- Well-defined interfaces
- Uniform appearance of models, code, and documentation
- Reusable models
- Readable models
- Problem-free exchange of models
- A simple, effective process
- Professional documentation
- Understandable presentations
- Fast software changes
- Cooperation with subcontractors
- Successful transitions of research or pre-development projects to product development

Here is the hyperlink to download the MAAB guidelines provided by ETH Zurich: MAAB Guidelines

3.4 Review of the previous Simulink model

Given the origin of this thesis from a previous study where MPC was developed and tested, it was essential to conduct a comprehensive review, particularly focusing on the computational aspects. Despite the presence of an existing infrastructure to leverage, significant challenges were encountered during the evaluation process. Below is a concise overview of the issues identified and observations made during this assessment:

3.4.1 Reference distance along the path

Let's define $\mathbf{s_v}$ as the actual distance travelled by the vehicle and $\mathbf{s_p}$ as the distance along the reference path.
In a PT task, knowing the vehicle's location only is not sufficient; in fact, understanding which part of the path we're referencing is crucial. This is where the parameter $\mathbf{s_p}$ becomes vital. It provides awareness of the current characteristics of the reference path, as it's used to parametrically define various path features such as $\mathbf{X_{ref}}$, $\mathbf{Y_{ref}}$, $\mathbf{k}$, and $\psi_{\mathbf{ref}}$. However, calculating the distance travelled by the vehicle doesn't directly translate to the distance travelled along the path due to path-tracking errors. Consequently, $\mathbf{s_p}$ and $\mathbf{s_v}$ can differ significantly in the reality. In the previous model though, the current path position was computed in a cumulative way shown below:
It was determined by calculating the lateral projection of the vehicle's Centre

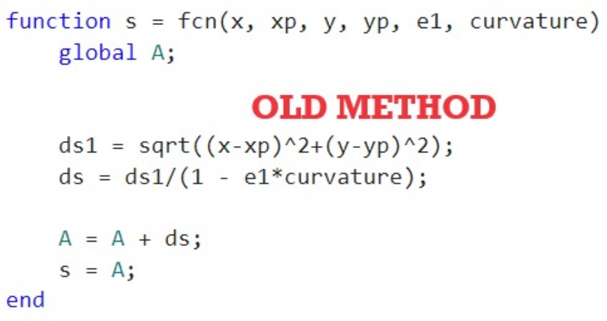

```
function s = fcn(x, xp, y, yp, e1, curvature)
    global A;

                OLD METHOD
    ds1 = sqrt((x-xp)^2+(y-yp)^2);
    ds = ds1/(1 - e1*curvature);

    A = A + ds;
    s = A;
end
```

Figure 3.5: Old code affected by error.

of Gravity (Cog) onto the path. However, this method only corrected the path travelled by the vehicle and then it was susceptible to cumulative errors, particularly on longer paths.

The new method 3.7, on the other hand, is an uninformed research algorithm that minimises discrepancies by iteratively computing the closest points along the path. Essentially, it geometrically computes the reference point while making assumptions about maximum distances from the path.

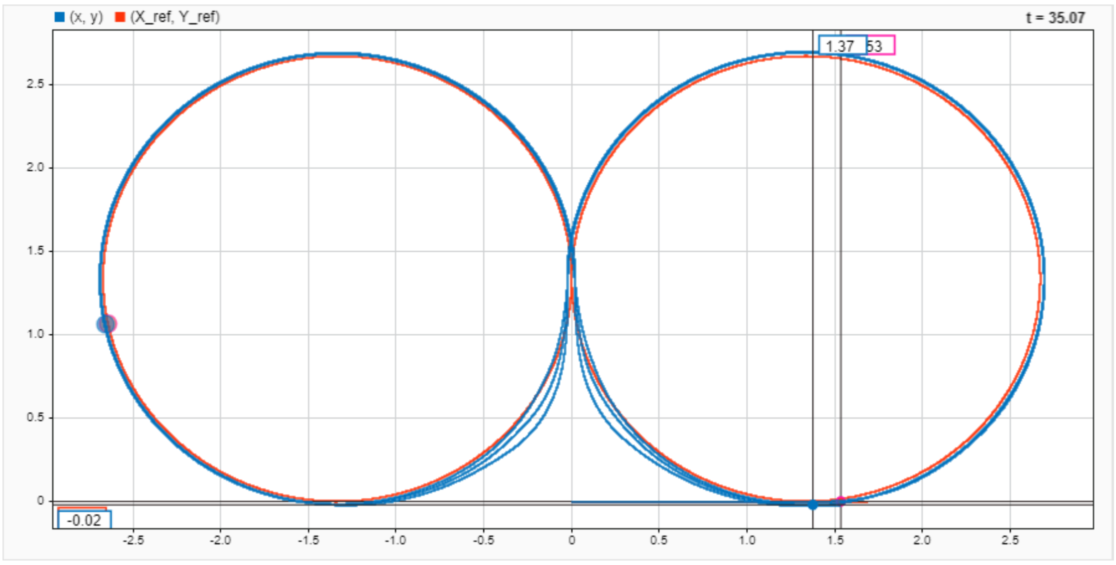


Figure 3.6: Data inspector scope on the entity of the distance-travelled error after 3 rounds in simulation.

The aforementioned effect becomes even more visible in the case of a real experiment due to the increased general magnitude of the errors. Moreover, the forward movement of the reference has the collateral effect of causing the controller to behave as if a look-ahead method were implemented, even if there is not. In this case, the look-ahead distance becomes larger and larger as the experiment progresses, causing the SSRAV to start steering before the curves.

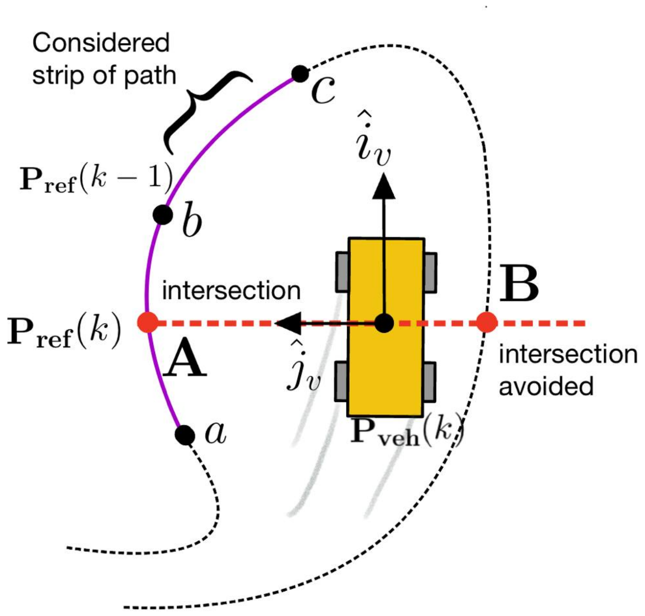


Figure 3.7: Graphical representation of the geometry behind the new algorithm developed.

The new algorithm is based on evaluating two sets of points: the SSRAV's research horizon (red line A-B), whose length is chosen a priori according to the magnitude of the respected errors, and the strip of path considered (violet line) at the given instant. Indeed, the algorithm evaluates a strip of path that is in the neighbourhood of the previously detected position. Despite being heavier than the previous one, this algorithm is very precise, as it is necessary to prevent errors over $\mathbf{s_p}$ from accumulating. [insert graphic of the algorithm]

3.4.2 Reference yaw

Similarly to the online computation of the reference yaw as before, this approach may encounter issues, particularly with longer paths. To address this, instead of employing a heavier online algorithm for research, a simpler solution involves computing the reference yaw offline. It's important to note that the reference yaw should be treated as a cumulative value and thus not constrained between $-\pi$ and π due to the `atan` domain problem.
To achieve this offline, a lookup table is created, taking the current path position $\mathbf{s_p}$ as input and yielding ψ_{ref} as output. One challenge faced was extrapolating the cumulative yaw, especially when using the tangent function.
The previous method computed the reference yaw as follows:

$$\dot{\psi}_{ref} = v_x k \tag{3.5}$$

The equation above is valid when $e_2 << 1$, meaning that the longitudinal axis of the SSRAV can be considered parallel to the reference path. In other words, the assumption here was that v_x is perpendicular to the path, which holds true only when the heading error is null and of course, this condition is not always satisfied. Moreover, integrals should be avoided as they accumulate errors as was done before the changes.

$$\dot{\psi} = \int \dot{\psi}_{ref} dt \tag{3.6}$$

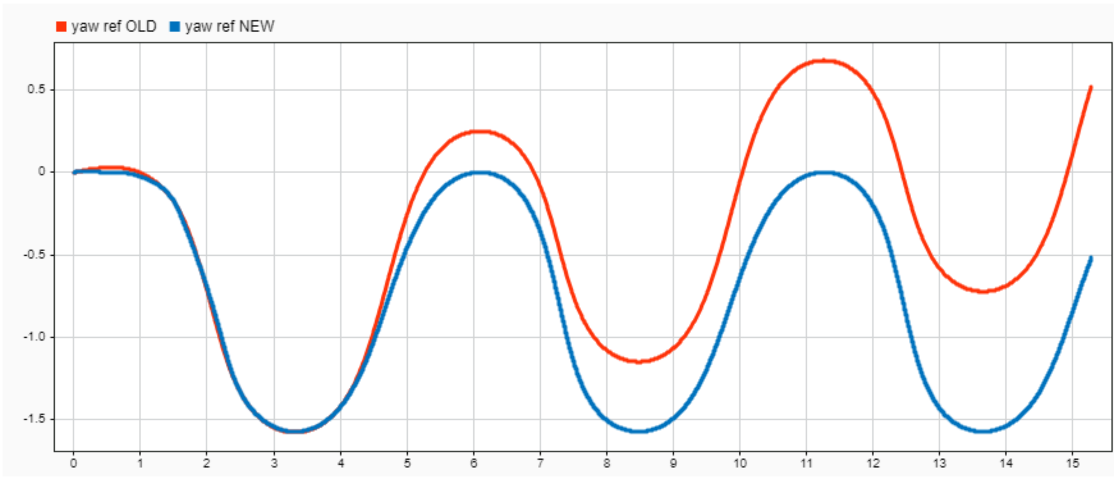


Figure 3.8: Scope of the accumulated error to compute ψ_{ref} after 15 s.

In this regard, below is the code for the nested function that computes the yaw angle (ψ). Note that it employs incremental calculus to confine the pose within the range of $-\pi$ to π:

3.4.3 Simulink modifications According to MAAB Guidelines

In particular, modifications made in the previous model were:

- Providing variables with more understandable names. As the MAAB guideline tells us, "Adoption of a naming convention is recommended. A naming convention guides naming blocks, signals, parameters, and data types". Naming conventions frequently cover issues such as:
  - Compliance with the programming language and downstream tools
    - Length
    - Use of symbols
  - Readability
    - Use of underscores
    - Use of capitalisation
    - Encoding information
    - Use of "meaningful" names
    - Standard abbreviations and acronyms
  - Data type
    - Engineering units
    - Data Ownership
    - Memory type
- Avoiding the abuse of `goto` and `from` blocks. The guide is explicit about this: their use must be limited for two reasons: to speed up the code and improve readability.
- Especially in the case of the model for the simulation, there was excessive use of sum, product, and operational blocks. In this case, it is better to use MATLAB function calls that summarise the operations in a bunch of lines of code with proper comments.

3.5 LMPC dataset

Due to the spatial dimension of the problem, the creation of the dataset was a bit tricky. This is because the unseen data that the MLP is going to process is no longer just the state (actually augmented state since now also the speed is considered), but the whole prediction horizon (what was called disturbances in the ex-MPC problem). Thus, the size in terms of memory of the dataset strictly depends on the prediction horizon. Let's provide a bunch of numbers:

Basically, the input layer will have the following structure:

$$\overbrace{\overbrace{\begin{bmatrix}k_0 & k_1 & \dots & k_{N-1} & k_N\end{bmatrix}}^{\text{Prediction Horizon}}\overbrace{\overbrace{\begin{bmatrix}e_1 & \dot{e_1} & e_1 & \dot{e_1} & \delta\end{bmatrix}}^{\text{State}}\begin{bmatrix}v_x\end{bmatrix}}^{\text{Augmented state}}}^{\text{Input vector (unseen data)}} \tag{3.7}$$

Moreover, for the training dataset, it is necessary to add a column regarding the input to guess. Then the final structure of the dataset will have rows with the following shape:

$$\overbrace{\overbrace{\begin{bmatrix}k_0 & \dots & k_{N-1} & k_N & e_1 & \dot{e_1} & e_1 & \dot{e_1} & \delta & v_x\end{bmatrix}}^{\text{Input}}\overbrace{\begin{bmatrix}\dot{\delta}\end{bmatrix}}^{\text{Label}}}^{\text{Training vector}} \tag{3.8}$$

The shape of the dataset (e.g., the input column(s) placed as the last element of the vector) must respect the requirements that the toolchain developed before requires. They will be faced in the subsection 4.1.1.

To generate a sufficiently representative dataset, two data-saving strategies were considered:

a. Utilising a uniform distribution via an open-loop simulation.

b. Employing a closed-loop simulation with a non-uniform distribution, dependent on the chosen trajectory and the system's state response.

Option (a.) was swiftly discarded due to two primary reasons:

1. Randomly combining inputs led to a dataset with an inaccurate distribution, resulting in poorly trained areas (e.g., combinations of infeasible curvatures or unrealistic velocity and position pairings). The distribution became normal, causing combinations with differing probabilities to have equal probabilities (statistical inaccuracy).

2. The problem's scale quickly escalated, resulting in a large dataset. For instance, assuming a vector length of M, each element with 5 possible values (a highly restrictive assumption), the resulting number of rows would be 5^M.

Ultimately, option (b.) was chosen. However, it comes with the drawback that the dataset's outcome is closely tied to the modelled system and the trajectory traversed by the simulated SSRAV (e.g., a sinusoidal path yields different outcomes than a circular one).

3.6 Chosen paths

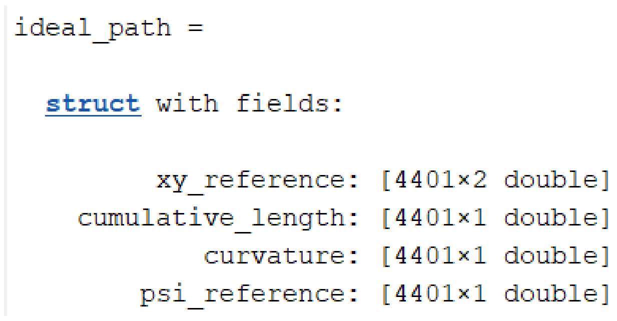
It is necessary to distinguish between two kinds of paths: those used for the validation phase, which should not be included in the training, and those used for training. Since the MPC architecture was already tested over some paths, the latter were also used for the validation set. The `.mat` files where they are saved are structures that have stored:

- Cumulative length s_p sampled each 0.001 m.
- XY reference parameterised over Cumulative length s_p,

$$P_{\text{ref}}(s_p) = \Big[X_{\text{ref}}(s_p); \quad Y_{\text{ref}}(s_p)\Big] \tag{3.9}$$

- Curvature $k(s_p)$ still parameterised over cumulative length s_p
- Yaw reference $\psi_{\text{ref}}(s_p)$ still parameterised over cumulative length s_p

Below is how the structure looks like in MATLAB:



```
ideal_path =

  struct with fields:

          xy_reference: [4401×2 double]
     cumulative_length: [4401×1 double]
             curvature: [4401×1 double]
         psi_reference: [4401×1 double]
```

Figure 3.9: Data structure for the ideal path.

3.6.1 Validation paths

The validation paths consist of basically three trajectories. They are chosen to fit the laboratory environment and are directly saved in the proper folder of the toolkit (refer to Section 4.2). They consist of an S-shaped trajectory, an O-shaped one, and an eight-shaped one. Their respective aliases are `S_path`, `O_path`, and `OO_path`.

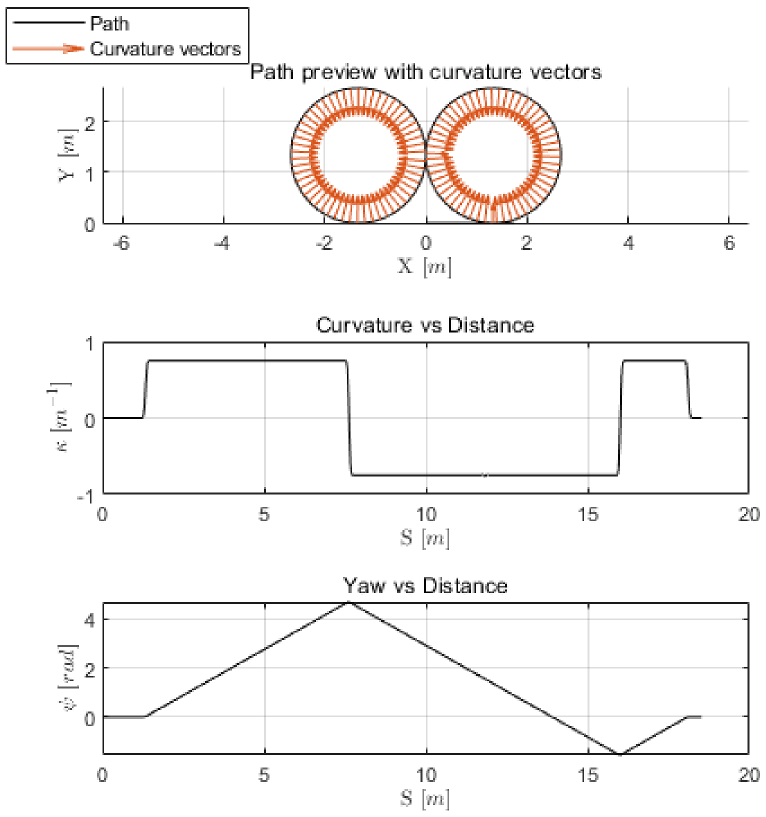


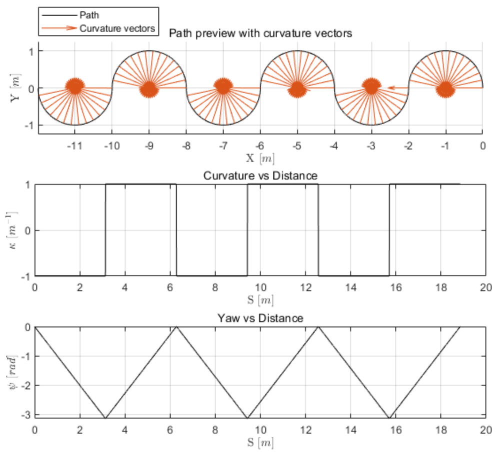
Figure 3.10: Eight validation path

.

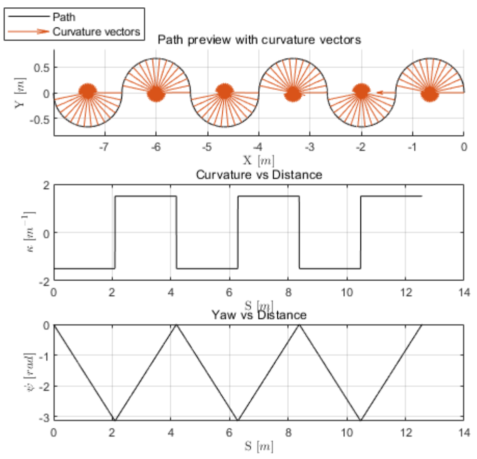
3.6.2 Training paths

Here, the difficult part arises. This is because, as briefly mentioned earlier, the training paths will serve as the foundation for the MLP training process. Naturally, the variety of training paths in the spectrum will also affect the hyperparameters of the MLP, and indeed, numerous trials were conducted. Eventually, two kinds of training paths were utilised:

- **Repeated-S paths**: These paths consist of repeated semi-circles. They are symmetrical to ensure that the MLP is not trained with uncentred data. Along the path, the curvature can assume 3 values: -k, 0, and +k, where k is a parameter decided beforehand. Nine repeated-S paths were created for nine different values of k. Below there are two instances:



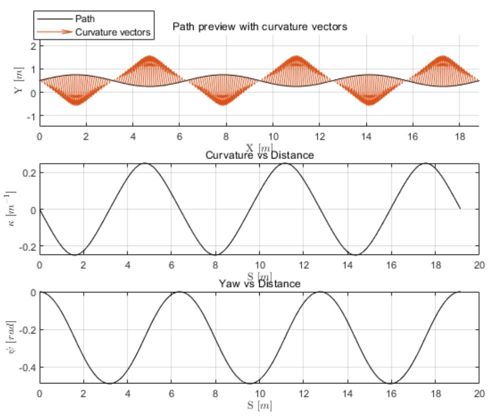
(a) S-shaped path with a radius of 1 m.



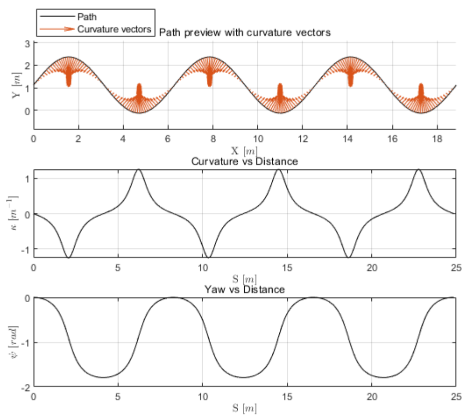
(b) S-shaped path with a radius of 0.5 m.

Figure 3.11: S-shaped trajectory example. These plots depict not only the xy path but also the curvature over the distance $\mathbf{s_p}$ and the yaw reference $\psi_{\mathbf{ref}}$, still parameterised as a function of the distance $\mathbf{s_p}$. Notice how the curvature is defined: its "bang-bang" shape will make the training easier due to the limited values that the curvature can take.

- **Sinusoidal paths**: These paths consist of a sine wave. The sine wave adopts a continuous range of curvatures from -k to +k. Consequently, this type of path makes the training process more complex. Similarly, specific k references were chosen for this solution. Below is an instance of a sinusoidal path:



(a) Sinusoidal path whose amplitude is 0.75 m.



(b) Sinusoidal path whose amplitude is 2.25 m.

Figure 3.12: Sinusoidal path example. As in the previous plots, these ones depict not only the xy path but also the curvature over the distance $\mathbf{s_p}$ and the yaw reference $\psi_{\mathbf{ref}}$, still parameterised as a function of the distance $\mathbf{s_p}$. Notice instead how the curvature is variable. This will make the training more complex but will allow the MLP to better generalise the PT task.

3.6.3 Data Cleaning & Data Augmentation

- To avoid biased training, two equal training datasets were created. One dataset traverses the path from left to right, while the other traverses the path in the opposite direction. This approach ensures symmetry in the training process. Formally, data augmentation was performed.
- Another methodology employed to prevent higher speeds from resulting in fewer measurements was to manually limit the simulation time. By imposing a limit, each velocity received an equal distribution of rows, ensuring consistent sampling despite variations in speed. Subsequently, the dataset was cleaned to remove non-uniform rows.

3.7 Learning Algorithms for MLP

As highlighted in the introduction, the choice of the MLP as the selected ANN architecture comes from the will to prove its effectiveness even if its simplicity is compared to more complex ANN structures. In the following subsection 3.7.1, we provide a brief overview of the MLP and discuss the Adam optimiser utilised in the learning process. Subsequently, in subsection 3.7.2, we delve into a methodology for conducting trial and error experiments concerning training, focusing on the selection of optimal hyper-parameters and activation functions.

3.7.1 Multi-Layer Perceptron & Adam Optimiser

Multi-Layer Perceptron (MLP) stands out as the simplest form of Artificial Neural Network (ANN). Comprising layers with varying quantities of artificial neurons (referred to as layer width), an MLP exhibits a sequential structure. The initial layer, known as the input layer, and the concluding layer, referred to as the output layer, play pivotal roles. The output layer shapes the final output of the network. Intermediate layers, termed hidden layers, serve as the computational core of the ANN.

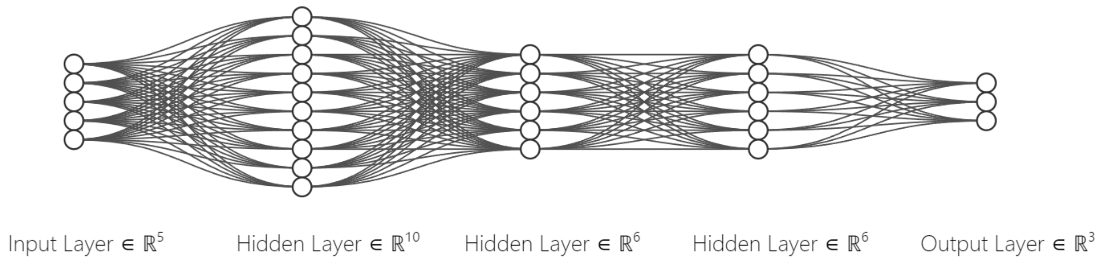

Input Layer ∈ $\mathbb{R}^5$
Hidden Layer ∈ $\mathbb{R}^{10}$
Hidden Layer ∈ $\mathbb{R}^6$
Hidden Layer ∈ $\mathbb{R}^6$
Output Layer ∈ $\mathbb{R}^3$

Figure 3.13: Representation of a generic MLP. The drawing of this MLP has been created thanks to Neural Network SVG Drawer, which allows rendering them [37].

The architecture of an MLP is deemed 'dense' due to its characteristic of full connectivity between neurons of successive layers, forming a directed graph. Consequently, data flows unidirectional, from the input layer to the output layer. Each connection is associated with a parameter called weight $\mathbf{w}$, which multiplies the value originating from the connected neuron. Additionally, each neuron features a parameter known as bias $\mathbf{b}$, added to the value of the associated neuron. As a result, the number of weights exceeds the number of biases. The shape vector of the MLP, representing the width of each layer, dictates the total number of biases and weights and how to arrange them for the deployment of the MLP itself. This will be fundamental for the development of the tool-chain described in section 4.1.

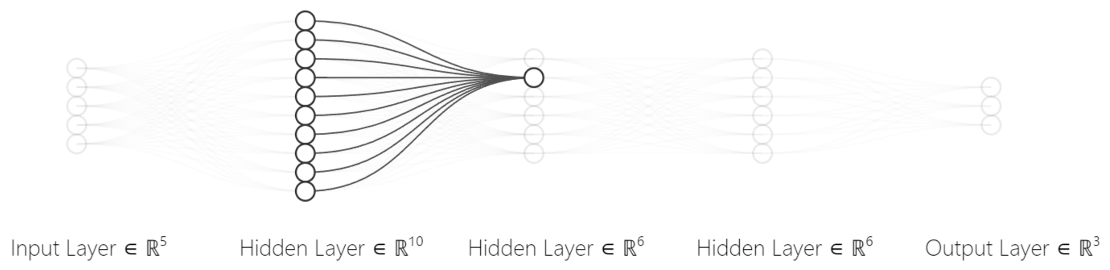

Input Layer ∈ $\mathbb{R}^5$
Hidden Layer ∈ $\mathbb{R}^{10}$
Hidden Layer ∈ $\mathbb{R}^6$
Hidden Layer ∈ $\mathbb{R}^6$
Output Layer ∈ $\mathbb{R}^3$

Figure 3.14: Highlight of neuron connections. The drawing of this MLP has been created thanks to Neural Network SVG Drawer, which allows rendering them [37].

In general, the output and input widths are predetermined. In this application, the input width depends on the prediction horizon $\mathbf{p_h}$ and state size affects the input width, while the output width depends on the actuator that in unitary (i.e. steering derivative $\dot{\delta}$). The inclusion of velocity in the online parameters expands the input state feedback from 5 to 6.

Activation functions, applied at the neuron level, contribute to the diversity of MLPs. Various types of activation functions exist, each with its unique advantages and drawbacks.

The training process of an MLP involves adjusting the weights and biases to minimise the difference between predicted and actual outputs. This optimisation is achieved through an algorithm, which adapts the learning rate for each parameter individually. The efficacy of the MLP depends on careful considerations of its architecture, activation functions, and optimisation techniques. This process is known as back-propagation.

In this application, the method used to optimise the update process for the parameters is the Adam optimiser. Before discussing Adam, it is essential to provide brief explanations of other optimisation algorithms.

Gradient Descent (GD): Utilising a fixed learning rate, GD's updating process is static, solely relying on the gradient of the loss function at the current point. However, this approach may lead to slow convergence and susceptibility to local minima.

GD with Momentum: While maintaining a fixed learning rate, GD with Momentum introduces a dynamic updating process by incorporating a decay coefficient. Picture this as a small ball rolling down the cost function landscape. The inertia and viscosity gained through the decay coefficient help the algorithm avoid local minima and accelerate convergence.

GD with RMSProp: Similar to GD with Momentum, GD with RMSProp employs a fixed learning rate but adjusts the updating process dynamically with a decay coefficient. It emphasises updating in flat zones and applies gentler updates in areas with abrupt gradients, making it effective in landscapes with varying gradients.

Adam: Adam, a compromise between Momentum and RMSProp, combines adaptive learning rates with momentum. It maintains two moving averages for each parameter: the first moment (mean) and the second moment (uncentred variance). The learning rate is adaptively adjusted based on historical gradients (Momentum) and their magnitudes (RMSProp). This adaptive mechanism overcomes the limitations of fixed learning rates in GD and the sensitivity to varying landscapes in RMSProp. Imagine Adam as a versatile explorer gracefully navigating the cost function landscape with a keen sense of both direction and speed, providing a balanced convergence speed and robustness.

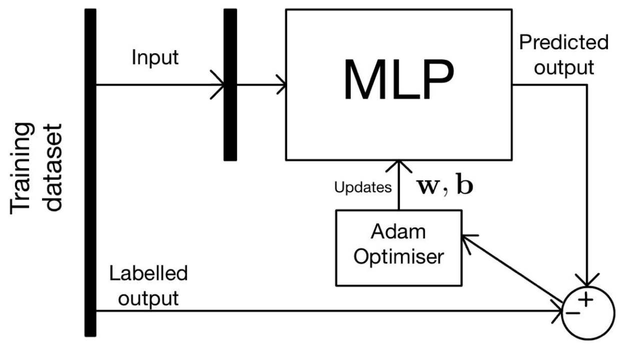


Figure 3.15: Block scheme of back-propagation training.

3.7.2 Hyperparameters & Activation Functions

Determining hyperparameters and activation functions, as well as the dimensions of any ANN, is inherently challenging as they depend on the shape of the function being approximated. However, there are techniques that can provide guidance. Below, we briefly describe the criteria for selecting hyperparameters such as the number of epochs, batch size, and learning rate.

Initially, it is necessary to list the data that are given and constrained by the problem. Thus, given a fixed dimension of the dataset and fixed input and output layers, determining the number of epochs becomes more manageable. Typically, a range of epochs can be considered. For this application, with the following data characteristics:

- The training dataset comprises approximately 500,000 rows.
  These rows were generated by simulating the MPC over the training path designed using the methodologies outlined in subsection 3.6.2, across various longitudinal speeds. Specifically, the validation speeds chosen are defined in equation 3.1.

- Similarly, the validation dataset consists of approximately 150,000 rows.
  The generation process mirrors that of the training dataset, albeit utilising benchmark paths instead.

- The input layer width is calculated as follows: 6 (for the augmented state) + 40 (for the prediction horizon) + 1 = 47.
  Notably, the input layer width is affected by the length of the prediction horizon. While a longer prediction horizon may enhance MPC optimisation, it also poses challenges for MLP training due to the increased volume of data involved. Additionally, the state is augmented to 6 dimensions to account for the longitudinal speed.

- Conversely, the output layer width remains fixed at 1 ($\dot{\delta}$).
  This fixed output is due to its dependence on the number of actuators, which remains unchanged regardless of the MLP's deployment.

A suitable number of epochs was found to be 100. If subsequent analysis indicates potential for improvement, further experimentation will be conducted, but the baseline for all tests will remain at 100 epochs.

In terms of batch size, the literature suggests a starting point of $\mathbf{B} = 32$. Although $\mathbf{B} = 32$ is more suitable for smaller datasets due to slower training with smaller batch sizes, the powerful workstation utilised in this study allowed for $\mathbf{B} = 32$ to be retained as the default batch size.

Similarly, considering the ample amount of available data, a smaller learning rate is deemed appropriate. Thus, the initial learning rate is set to $\alpha = 5.0000 \times 10^{-5}$. Notably, this low value is also chosen in accordance with the characteristics associated with the Adam optimiser implemented in the tool-chain developed for this work, as briefly discussed in Section 4.1.

Furthermore, it is essential to mention two features connected with the Adam optimiser: the momentum, denoted as $\beta_1 = 0.9800$, and the sum squared weights, denoted as $\beta_2 = 0.9700$, both contributing to the optimisation process.

Table 3.3: Default values for hyperparameters.

Hyperparameter	**Default Value**
Number of epochs	100
Batch size ($\mathbf{B}$)	32
Learning rate (α)	5.0000×10^{-5}
Momentum (β_1)	0.9800
Sum squared weights (β_2)	0.9700

3.8 Experimental Approach

Throughout the experimentation phase, a meticulous approach was taken to ensure the reliability and consistency of the results obtained. The goal was to ensure the experimental environment remained comparable test by test. Some taken precautions are taken below:

- Use of the same SSRAV (QCar) for all trials, despite the availability of multiple units within the laboratory. By maintaining this uniformity in the testing environment, it was possible to facilitate accurate comparisons between the performances of different controllers.

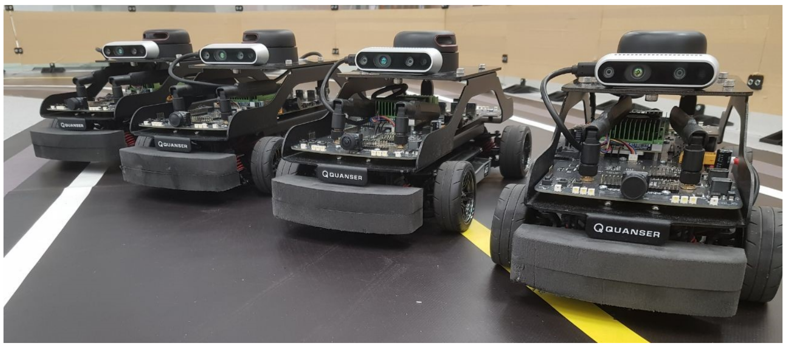


Figure 3.16: The 4 QCars available in the laboratory of University of Surrey.

- Battery charge check. To safeguard against any potential degradation in SSRAV performance, particular attention was paid to monitoring the battery charge level. A critical threshold of 11.5V was established, and measures were taken to ensure that the battery charge always exceeded this limit before each experiment commenced. This precautionary step aimed to mitigate any adverse effects on performance resulting from inadequate power supply.

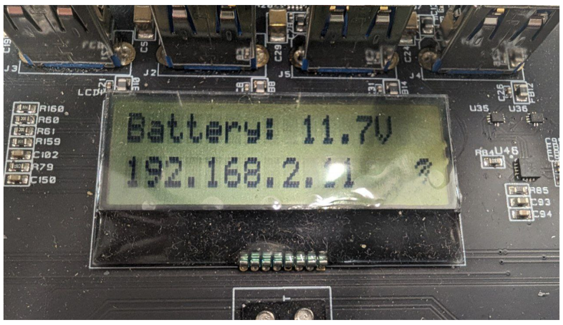


Figure 3.17: Display of the QCar showing battery charge.

- Precision in initial positioning and mapping. Despite the option to utilise previously scanned environmental data, it was decided to perform a new LIDAR scan for each trial. This approach ensured that the mapping of the environment was consistently precise in terms of the SSRAV's initial position and orientation since the SSRAV is manually placed test by test. By adhering to this practice, potential discrepancies in the experimental setup were minimised, thereby enhancing the validity and repeatability of the results obtained.

- The experimental approach also involved creating a dedicated environment. Specifically, a specialised track was utilised to simulate realistic friction between the road and the QCar wheels. The setup is depicted below.

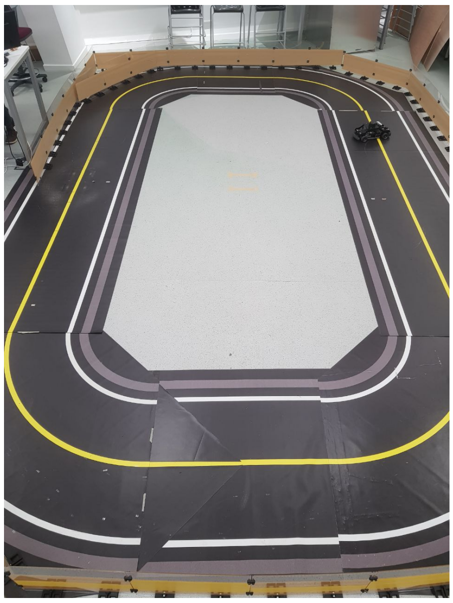


Figure 3.18: Track used for experiments.

- In order to ensure robustness and reliability of the data, multiple measurements were taken. Each controller was tested three times over the same path at the same speed. Following the completion of each trial, the mean and standard deviation of the measurements were calculated to quantify central tendency and variability, respectively. Subsequently, error bar graphs were constructed to visually represent the mean values along with the associated uncertainties, aiding in the interpretation and visualisation of the data.

Chapter 4

Development of programming tools

This thesis entailed the creation of a comprehensive workflow from scratch, necessitated by the considerable volume of data and the significant computational effort involved. A crucial aspect of this endeavour is the need for a meticulous and systematic approach to effectively navigate through extensive datasets and numerous LMPC instances. The process began with the management and storage of numerous MPC solutions, followed by a stage focused on training and storing MLP models derived from datasets originating from the MPCs. These models were then utilised in various simulations, and the resulting KPIs were saved and compared. Thus, a well-defined methodology was imperative to prevent data loss, optimise computational resources, and ensure time-work efficiency.

To tackle this significant challenge, an automated method for creating MPCs using ACADO was identified. Additionally, to expedite the training process, a hybrid approach was adopted: the MLP was configured in MATLAB but trained in Python, leveraging the Python implementation within MATLAB. This decision was made to optimise computational workload, with a substantial portion executed in Python. Furthermore, for MLP deployment, a Simulink function was developed. This integrated approach aimed to streamline the overall process and enhance the efficiency of both the MPC and MLP components in the study.

The development of the programs unfolded in two distinct but interlinked parts, forming a toolchain nested within a toolkit dedicated to NNBO applied to MPC problems. In this work, they are simply referred to as the "toolchain" and "toolkit." Their names will be refined in the future, after this thesis, as future developments progress. In fact, this work will continue beyond this thesis, and the programming tools will undergo improvements and modifications.

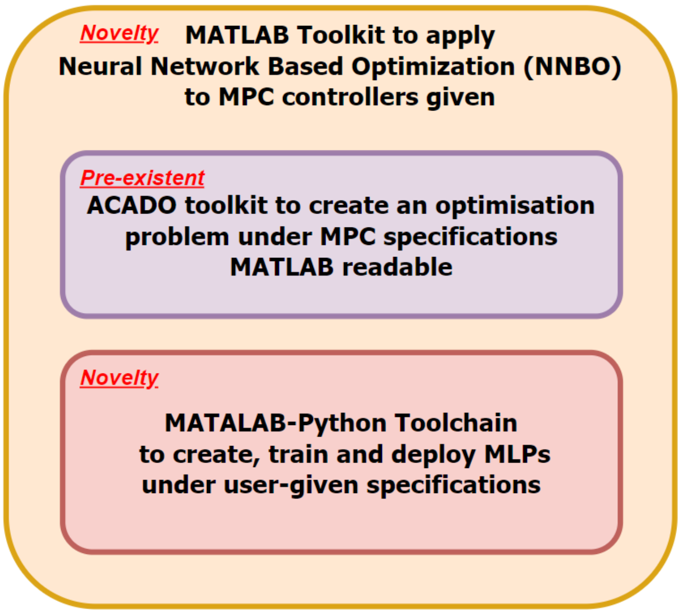

Novelty
MATLAB Toolkit to apply
Neural Network Based Optimization (NNBO)
to MPC controllers given
Pre-existent
ACADO toolkit to create an optimisation problem under MPC specifications MATLAB readable
Novelty
MATALAB-Python Toolchain
to create, train and deploy MLPs
under user-given specifications

Figure 4.1: The two novelty tools are the MLP-dedicated toolchain and the NNBO for MPCs-dedicated toolkit. ACADO toolkit is a pre-existent open source tool instead which is used exclusively for creating MPC-dedicated Optimisation Problems.

4.1 MLP builder toolchain

The aim of the first tool-chain is a general application. There was the need of a tool able to straightforwardly set up and train a MLP to then automatically deploy it on Simulink on a given plant. In order to perform the training fast, the best option was performing the former on Python rather than MATLAB. On the other hand, the usage of Python will make the difficulty in building the framework increase. Its structure can be summarised in the following way:

1. General requirements
2. Set up in **MATLAB**
3. Training & Validation in **Python**
4. Evaluation and Report file generation (`.mat`, `.txt`, `.fig`)
5. Deployment (forward propagation):
   (a) On Simulink through a block, useful for control task like in this work and in general for time-based simulations and real-time too.[1]
   (b) On Matlab through a function to call.
   (c) On Python, merely using the same script where the MLP was previously trained, since the forward propagation is included in the training.

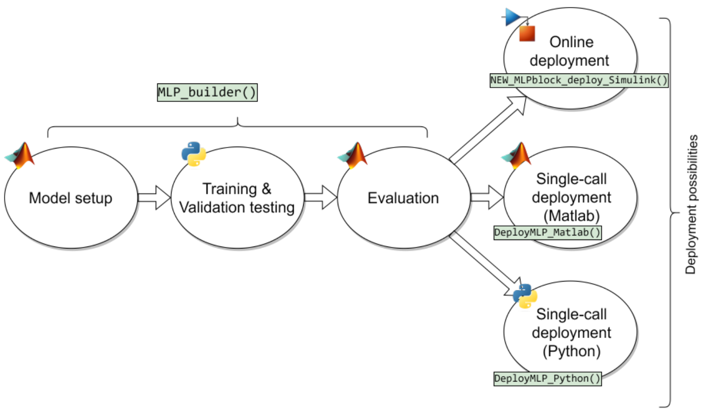

MLP_builder()
Model setup
Training & Validation testing
Evaluation
Online deployment
NEW_MLPblock_deploy_Simulink()
Single-call deployment (Matlab)
DeployMLP_Matlab()
Single-call deployment (Python)
DeployMLP_Python()
Deployment possibilities

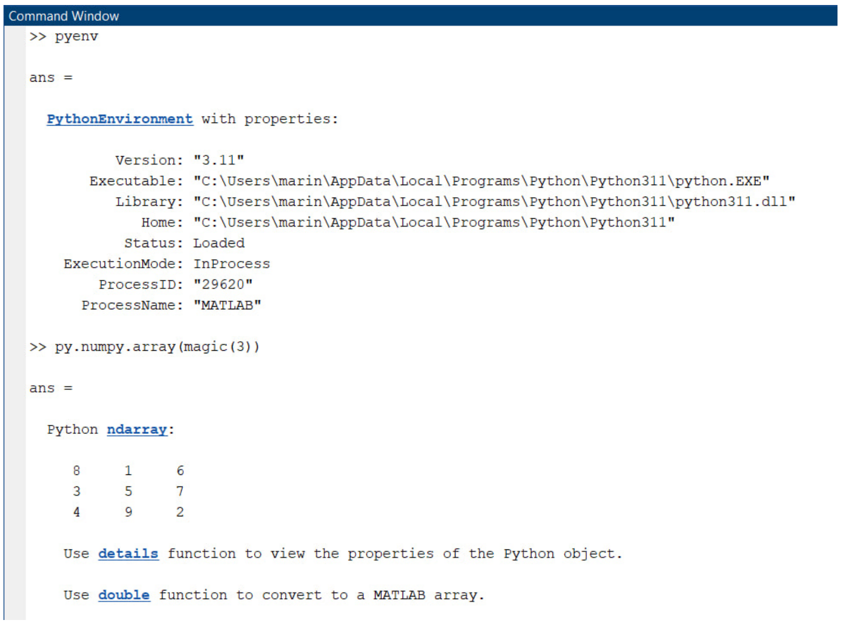
Figure 4.2: Stream of the developed toolchain.

[1]In this solution the MLP block may replace a controller or any other dynamic component if well trained.

4.1.1 Necessary Requirements

- Ensure that your system has a version of Python compatible with MATLAB. By visiting MATLAB Python Compatibility it is possible to verify the compatibility of your Python version. If your current version is incompatible, you can install a suitable version of Python directly from Python Downloads. Please pay attention to the specified version on the former link.

- MATLAB Python Integration: Configure MATLAB to interface with a compatible Python version. You can confirm that the integration is functioning by executing the following command in your command window: `>>pyenv`. You can also try to call basic Python function as `>>py.numpy.array(magic(3))`.The expected output structures are outlined below:



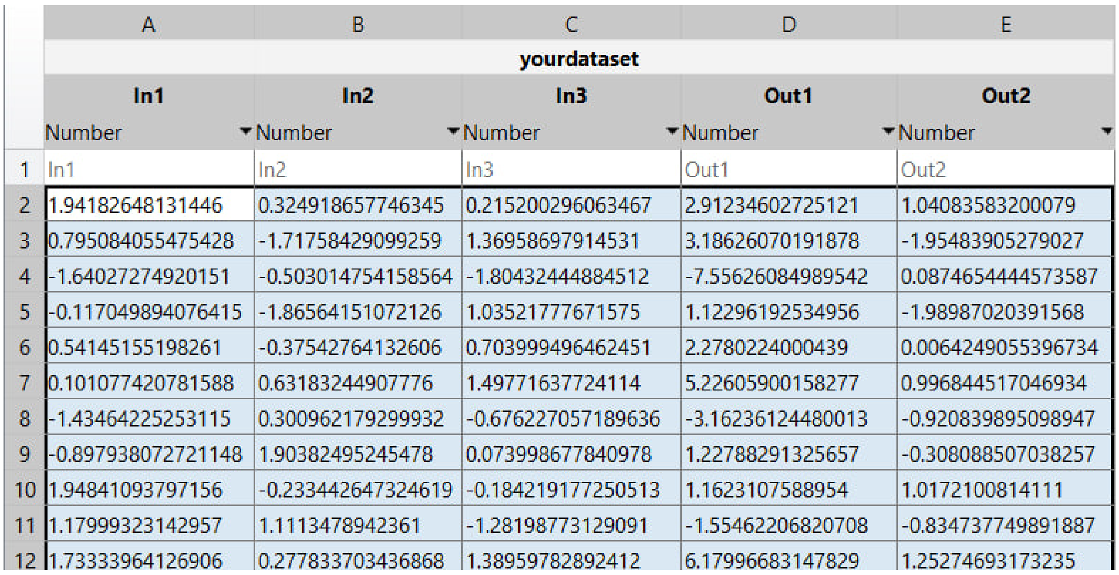
```
>> pyenv

ans =

  PythonEnvironment with properties:

          Version: "3.11"
       Executable: "C:\Users\marin\AppData\Local\Programs\Python\Python311\python.EXE"
          Library: "C:\Users\marin\AppData\Local\Programs\Python\Python311\python311.dll"
             Home: "C:\Users\marin\AppData\Local\Programs\Python\Python311"
           Status: Loaded
    ExecutionMode: InProcess
        ProcessID: "29620"
      ProcessName: "MATLAB"

>> py.numpy.array(magic(3))

ans =

  Python ndarray:

     8     1     6
     3     5     7
     4     9     2

    Use details function to view the properties of the Python object.

    Use double function to convert to a MATLAB array.
```

Figure 4.3: Expected output after performing Python environment check.

  You can find more information on MATLAB Python Integration.

- Ensure that the Python file `MLP_builder.py` and `yourfilename.csv` are present in the current folder, together with the function itself `MLP_builder.m`. Moreover, if you want to perform further deployment, ensure in your folder there are the corresponding functions and files: `DeployMLP_Matlab.m`, `DeployMLP_Python.m` plus `DeployMLP_system.slx` and the subsystem reference `DeployMLP.slx`.

- Ensure that the structure of the `.csv` file aligns with your requirements. You can create an empty table using the MATLAB command `T=table()` and then write the table to a CSV file using `writetable(T,'yourdataset.csv')`, similar to the operations performed in the MATLAB script `dataset_builder.m`. The first N columns should represent the inputs, while the remaining M columns should correspond to the outputs. It is important to verify that N and M are correctly defined based on your data and share the same length. An example is provided below for reference.



	A	B	C	D	E
	yourdataset				
	In1	In2	In3	Out1	Out2
	Number	Number	Number	Number	Number
1	In1	In2	In3	Out1	Out2
2	1.94182648131446	0.324918657746345	0.215200296063467	2.91234602725121	1.04083583200079
3	0.795084055475428	-1.71758429099259	1.36958697914531	3.18626070191878	-1.95483905279027
4	-1.64027274920151	-0.503014754158564	-1.80432444884512	-7.55626084989542	0.0874654444573587
5	-0.117049894076415	-1.86564151072126	1.03521777671575	1.12296192534956	-1.98987020391568
6	0.54145155198261	-0.37542764132606	0.703999496462451	2.2780224000439	0.0064249055396734
7	0.101077420781588	0.63183244907776	1.49771637724114	5.22605900158277	0.996844517046934
8	-1.43464225253115	0.300962179299932	-0.676227057189636	-3.16236124480013	-0.920839895098947
9	-0.897938072721148	1.90382495245478	0.073998677840978	1.22788291325657	-0.308088507038257
10	1.94841093797156	-0.233442647324619	-0.184219177250513	1.1623107588954	1.0172100814111
11	1.17999323142957	1.1113478942361	-1.28198773129091	-1.55462206820708	-0.834737749891887
12	1.73333964126906	0.277833703436868	1.38959782892412	6.17996683147829	1.25274693173235

Figure 4.4: Instance of how a suitable dataset should look like.

Useful sources:

- Explore the fundamental concepts of MLPs by referring to the content available at this link: MLP Features [38].

- If you find yourself uncertain about choosing an activation function for your neural network and seek in-depth explanations, I recommend consulting the following article: Activation Functions in Neural Networks.

- For a detailed review of the foundation of the Adam optimiser used in this application, referring to Analytics Vidhya's comprehensive guide on deep learning optimises can be helpful: Comprehensive Guide on Deep Learning Optimises.

4.1.2 Setup on MATLAB

This MATLAB phase is ruled by the Matlab function `MLP_builder.m`. It was designed to create and train a MLP neural network using a specified dataset provided in a `.csv` file. The function provides custom options for parameters such as the dataset file, shuffling preferences, layer widths, hidden layer structure, activation functions, and some training options. This function interfaces with a Python file `MLP_builder.py` to construct and train the MLP according to the specified configuration. After training in Python is completed, the function extracts key features: weights and biases are saved into a Python dictionary named `'MLP_dict'`, scaling factors used for data normalisation are stored in a Python array, and the losses are recorded in a Python array of tuples.

What could be improved in future versions?

- Code generation to implement the deployment code into an embedded system that uses microcontrollers.
- Other ways of preprocess data rather than only normalisation

User interface

```matlab
% Initialize the structure to fill out
MyMLP = MLP_builder('initialize');

% Set training settings
MyMLP.DatasetFullPath          = 'datasets\MyDataset.csv';
MyMLP.InputWidth               = 10;
MyMLP.OutputWidth              = 1;
MyMLP.OutputActivationFun      = 'sine';
MyMLP.HiddenWidths             = [8 10 8 3];
MyMLP.HiddenActivationFun      = {'tanh','sine','relu','
   relu'};
MyMLP.Epochs                   = 1500;
MyMLP.LearningRate             = 0.001;
MyMLP.BatchSize                = 512;
MyMLP.Momentum                 = 0.9;
MyMLP.SumSquaredWeight         = 0.99;

% Train the MLP
MLP_builder('train', MyMLP);
```

The method used wanted to be easy for the user and light to run for MATLAB. The initialisation allows for the creation of a default `struct` which will be customised then by the user. It is important to note that the architecture of the MLP, with regard to both its widths and depth, is flexible and can vary. While this flexibility may seem straightforward initially, transitioning from Python to MATLAB in subsequent stages proved to be nontrivial. In fact, it was necessary to code the information stored in biases and weights into two long arrays and then decode them using the array that describe the structure of the MLP itself.

Due to the length and the complexity of the function `MLP_builder.m`, it is not reported here. The main code is available downloading the tool-chain.

Mathematical expression		**MATLAB alias**
step	$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	`'step'`
linear	$f(x) = x$	`'lin'`
ReLU	$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	`'relu'`
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad i = 1, ..., J$	`'softmax'`
tanh	$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	`'tanh'`
sine	$f(x) = \sin(x)$	`'sine'`
sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$	`'sigmoid'`

Table 4.1: Examples of activation functions, operating either element-wise or vector-wise, depending on the function

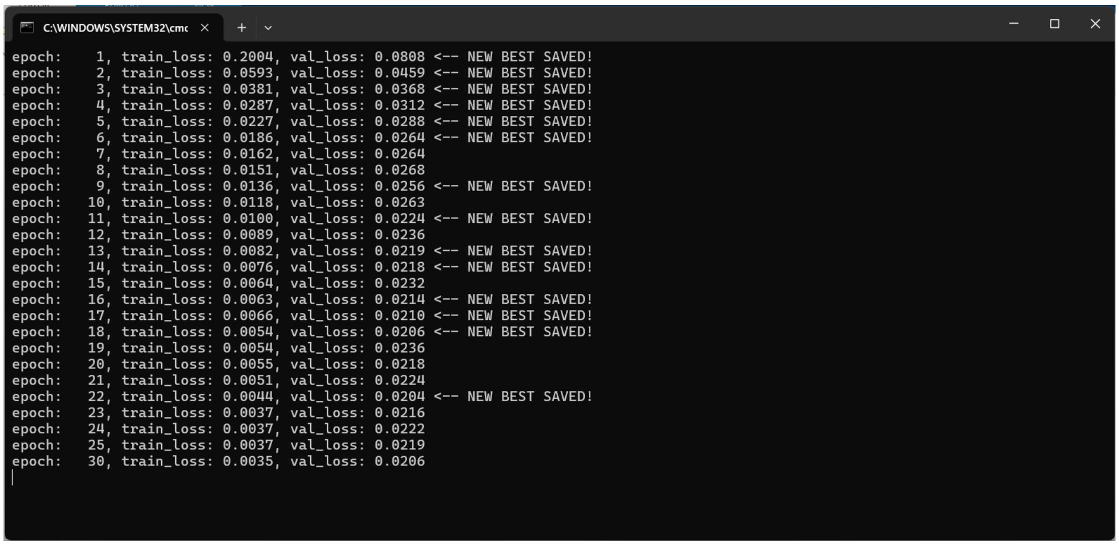
4.1.3 Training & Validation on Python

To keep the program lightweight, the only libraries utilised to construct the MLP were `pandas` and `numpy`. This approach allowed for the development of raw code optimised for our task while maintaining simplicity. While the complete code is omitted due to its length, a concise pseudo-code is provided below.

Algorithm 1 Python function called by MATLAB

```
Class Definitions:
    Class ADAMOptimiser:                       ▷ Class for the ADAM optimiser
    Class MultiLayerPerceptron:                ▷ Class for the Multi-Layer Perceptron

Input Activation functions, MLP structure, filepath, mode
if mode == "train" then
    Input train_dataset_name, valid_dataset_name
    Shuffle train_dataset_name and valid_dataset_name
    Perform consistency check between dataset and layer widths
    if consistency check passes then
        Normalise dataset
        Trainings using parameters from MATLAB
        Write report to provided filepath
        Save weights and biases of the best validation loss
    else
        Output "Consistency check failed"
    end if
else if mode == "deploy" then
    Input weights, biases
    Deploy MLP with provided weights and biases
else
    Output "Invalid mode. Please choose 'train' or 'deploy'."
end if
```

Another challenging aspect of the Python process was rendering the training state available online, which is not straightforward due to Python running in the process. The issue arises because the values of the losses computed epoch by epoch are returned as MATLAB output only at the end of the overall training. To address this, a workaround was implemented: the Python program creates and writes to a text file online, and a .exe function capable of reading the file row by row is used during the process. This setup ensures that when the training starts, a window similar to the one shown in the next figure automatically appears.

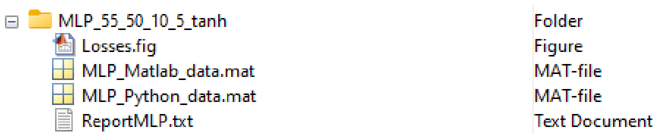


```
epoch:    1, train_loss: 0.2004, val_loss: 0.0808 <-- NEW BEST SAVED!
epoch:    2, train_loss: 0.0593, val_loss: 0.0459 <-- NEW BEST SAVED!
epoch:    3, train_loss: 0.0381, val_loss: 0.0368 <-- NEW BEST SAVED!
epoch:    4, train_loss: 0.0287, val_loss: 0.0312 <-- NEW BEST SAVED!
epoch:    5, train_loss: 0.0227, val_loss: 0.0288 <-- NEW BEST SAVED!
epoch:    6, train_loss: 0.0186, val_loss: 0.0264 <-- NEW BEST SAVED!
epoch:    7, train_loss: 0.0162, val_loss: 0.0264
epoch:    8, train_loss: 0.0151, val_loss: 0.0268
epoch:    9, train_loss: 0.0136, val_loss: 0.0256 <-- NEW BEST SAVED!
epoch:   10, train_loss: 0.0118, val_loss: 0.0263
epoch:   11, train_loss: 0.0100, val_loss: 0.0224 <-- NEW BEST SAVED!
epoch:   12, train_loss: 0.0089, val_loss: 0.0236
epoch:   13, train_loss: 0.0082, val_loss: 0.0219 <-- NEW BEST SAVED!
epoch:   14, train_loss: 0.0076, val_loss: 0.0218 <-- NEW BEST SAVED!
epoch:   15, train_loss: 0.0064, val_loss: 0.0232
epoch:   16, train_loss: 0.0063, val_loss: 0.0214 <-- NEW BEST SAVED!
epoch:   17, train_loss: 0.0066, val_loss: 0.0210 <-- NEW BEST SAVED!
epoch:   18, train_loss: 0.0054, val_loss: 0.0206 <-- NEW BEST SAVED!
epoch:   19, train_loss: 0.0054, val_loss: 0.0236
epoch:   20, train_loss: 0.0055, val_loss: 0.0218
epoch:   21, train_loss: 0.0051, val_loss: 0.0224
epoch:   22, train_loss: 0.0044, val_loss: 0.0204 <-- NEW BEST SAVED!
epoch:   23, train_loss: 0.0037, val_loss: 0.0216
epoch:   24, train_loss: 0.0037, val_loss: 0.0222
epoch:   25, train_loss: 0.0037, val_loss: 0.0219
epoch:   30, train_loss: 0.0035, val_loss: 0.0206
```

Figure 4.5: Window that automatically opens during the training to show online losses.

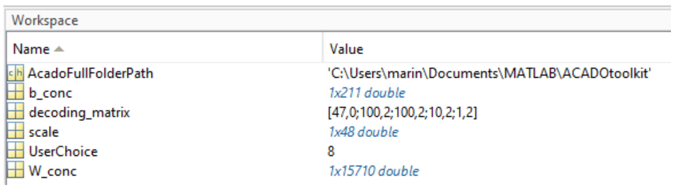
4.1.4 Evaluation on MATLAB

The evaluation phase pertains to the final outcome of the preceding training. Both validation and training losses are plotted, enabling a posterior analysis of issues such as overfitting, underfitting, loss shapes, etc. This analysis aids in fine-tuning the hyperparameters to optimise performance.



MLP_55_50_10_5_tanh	Folder
Losses.fig	Figure
MLP_Matlab_data.mat	MAT-file
MLP_Python_data.mat	MAT-file
ReportMLP.txt	Text Document

Figure 4.6: How the dataset folder should look like after a successful training.



Workspace

Name	Value
AcadoFullFolderPath	'C:\Users\marin\Documents\MATLAB\ACADOtoolkit'
b_conc	1x211 double
decoding_matrix	[47,0;100,2;100,2;10,2;1,2]
scale	1x48 double
UserChoice	8
W_conc	1x15710 double

Figure 4.7: MATLAB variables from training report, in particular from the file `.mat` that stores the whole information to deploy the MATLAB-function based MLP of the toolchain.

Notice how the MLP is represented concisely 4.8 as a vector of length N, where each element corresponds to the width of the respective layer. The first element represents the input layer, which, in the previous instance, was utilised for an image classification task using the MNIST dataset of handwritten digits. The parameter N denotes the depth of the MLP. Additionally, activation functions were encoded using string aliases, as detailed in Table 4.1. To facilitate communication between MATLAB and Python during deployment, each alias was mapped to an integer, which is utilised by the deployment function (see 4.1.5). Furthermore, for a comprehensive understanding of the training process, information regarding the Adam optimises settings is provided, including momentum decay (β_1), sum squared weight (β_2), batch size (**B**), and initial learning rate (α).

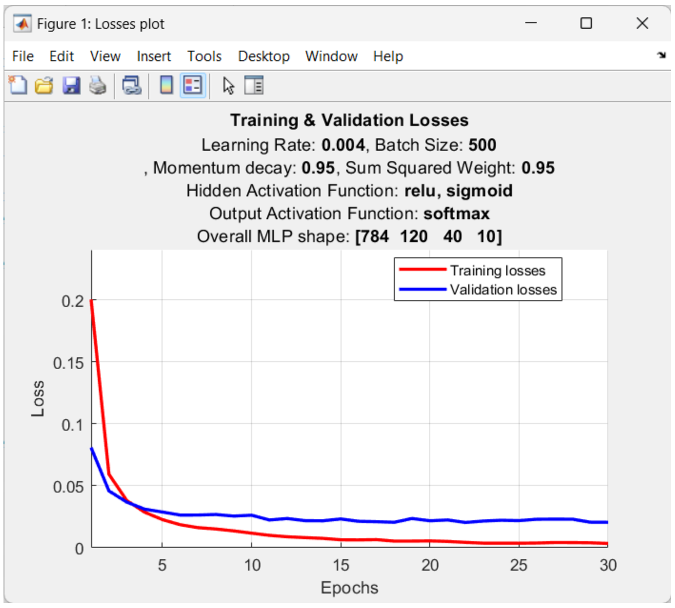


Figure 1: Losses plot

Training & Validation Losses
Learning Rate: **0.004**, Batch Size: **500**
, Momentum decay: **0.95**, Sum Squared Weight: **0.95**
Hidden Activation Function: **relu, sigmoid**
Output Activation Function: **softmax**
Overall MLP shape: **[784 120 40 10]**

Figure 4.8: Final plot window that automatically opens at the end of the training to show the overall concerning losses.

4.1.5 Deployment

The deployment will be the only section where some code is reported. This is because it is the core of the toolchain. The code that we decided to report is the one used to deploy in Simulink (that is basically the same of the one used in MATLAB, the only exception is that in MATLAB there were no limitation because of the variable-size signals that instead are limitational in Simulink.

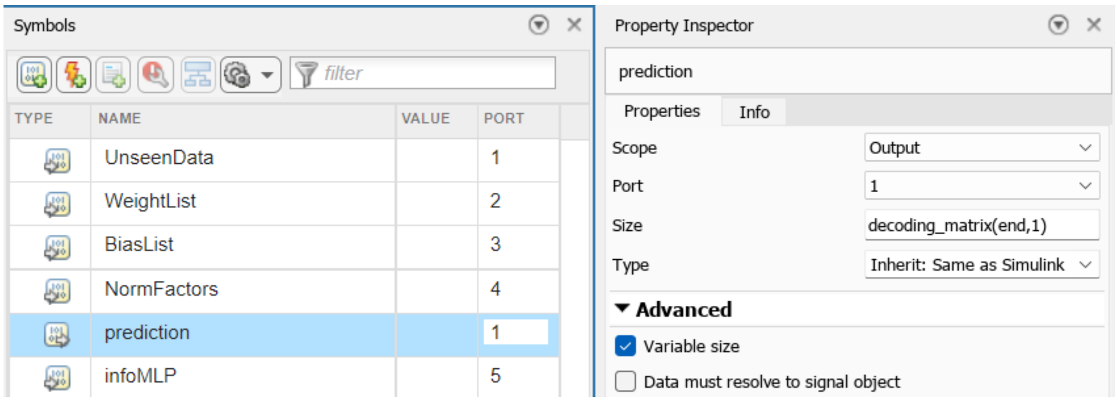
MATLAB/Simulink deployment

```matlab
function prediction = DeployMLP(UnseenData, ...
                                WeightList, ...
                                BiasList, ...
                                NormFactors, ...
                                infoMLP)
% Persistent variable in order to optimise the process
persistent W_decode b_decode
persistent InNormFactors OutNormFactors DepthMLP
%% One-time calculation branch
if isempty(W_decode)||isempty(b_decode)
    tic;
    disp('One-time calculations in progress...')
    DepthMLP        =  length(infoMLP(:,1))-1;
    W_decode        =  zeros(DepthMLP+1,1);
    b_decode        =  zeros(DepthMLP+1,1);
    InNormFactors   =  NormFactors(1:infoMLP(1,1));
    OutNormFactors  =  NormFactors(length(NormFactors)-
   infoMLP(end,1)+1:end);
    CurrentLayer    =  zeros(max(infoMLP(:,1)),1);
    sizes           =  [infoMLP(1:end-1,1).*infoMLP(2:end
   ,1) infoMLP(2:end,1)];
    for i=1:DepthMLP
        W_decode(i+1) = W_decode(i)+sizes(i,1);
        b_decode(i+1) = b_decode(i)+sizes(i,2);
    end
    Input_consistency_check(UnseenData);
    disp('One-time calculations ended...')
    toc;
    disp('------------------------------------------')
end
```

MATLAB/Simulink deployment

```
%% Every-time calculation branch
CurrentLayer = UnseenData./InNormFactors;
for i=1:DepthMLP
    Biases = BiasList(b_decode(i)+1:b_decode(i+1));
    Weights = reshape(WeightList(W_decode(i)+1:W_decode(
  i+1)),infoMLP(i,1),infoMLP(i+1,1));
    CurrentLayer = activation_function(infoMLP(i+1,2),
  Weights'*CurrentLayer+Biases);
end
prediction   =   CurrentLayer.*OutNormFactors;
```

It was important to properly set each input and output in the data editor of the function. This is because of the limitation in propagating signals in Simulink due to code generation. In fact, the output of the block must have been set to variable size, where the maximum size is the width of the output layer.



Symbols

TYPE	NAME	VALUE	PORT
	UnseenData		1
	WeightList		2
	BiasList		3
	NormFactors		4
	prediction		1
	infoMLP		5

Property Inspector

prediction

Properties | Info

Scope	Output
Port	1
Size	decoding_matrix(end,1)
Type	Inherit: Same as Simulink

Advanced

[✓] Variable size

[ ] Data must resolve to signal object

Figure 4.9: Setting in data editor for prediction output.

MATLAB/Simulink deployment

```matlab
%% Nested functions
    function result = activation_function(type, z)
        % This function decode which activation function it is necessary
        % to use in according to the code provided by the decoding matrix.
        switch type
            case 1 % sigmoid
                result = 1 ./ (1 + exp(-z));
            case 2 % tanh
                result = tanh(z);
            case 3 % relu
                result = max(0, z);
            case 4 % lin
                result = z;
            case 5 % softmax
                exp_z = exp(z);
                result = exp_z / sum(exp_z);
            case 6 % step
                result = double(z > 0);
            case 7 % sine
                result = sin(z);
            otherwise; error('Invalid activation function type');
        end
end
    function Input_consistency_check(input_layer)
        if ~isvector(input_layer)
            error('DeployMLP can process no more than one input at the same time in Simulink');
        elseif length(input_layer)~=infoMLP(1,1)
            error('Input size is wrong: detected [%d] instead of [%d]', length(input_layer), infoMLP(1,1));
        end
    end
end
```

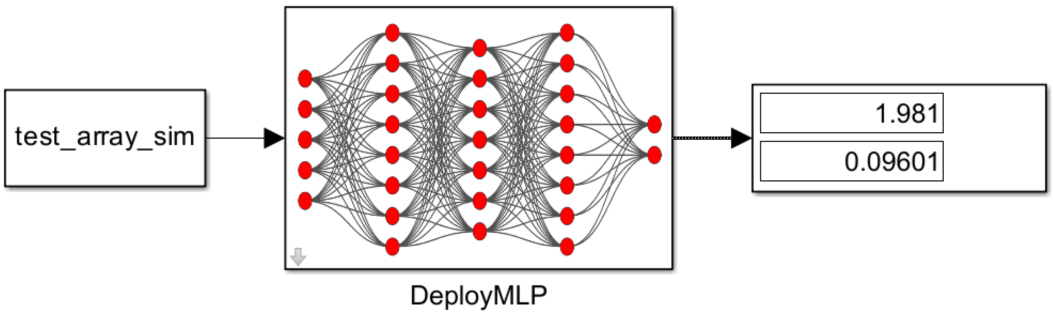
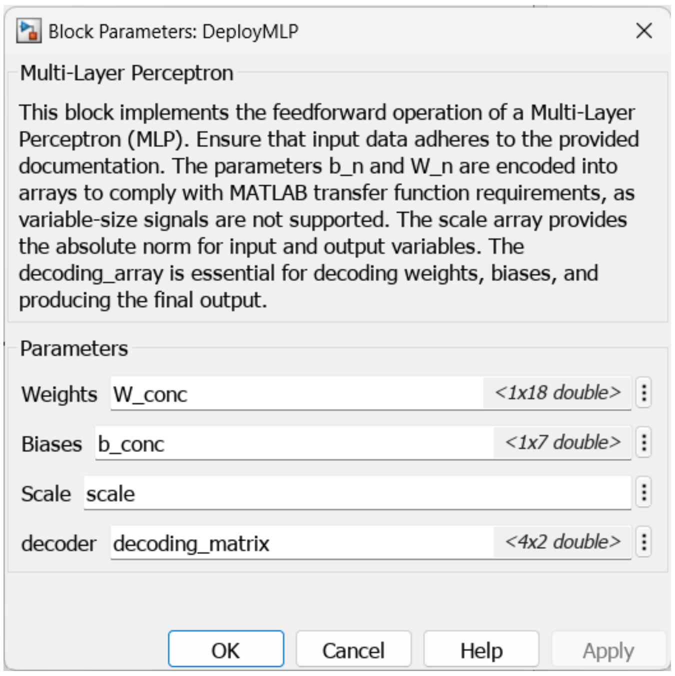


Figure 4.10: Simulink block for MLP deployment. This block is masked in order to be ready-to-use after loading of the MATLAB variables that describe the MLP structure. The workspace should look like in this figure 4.7.


Block Parameters: DeployMLP

Multi-Layer Perceptron

This block implements the feedforward operation of a Multi-Layer Perceptron (MLP). Ensure that input data adheres to the provided documentation. The parameters b_n and W_n are encoded into arrays to comply with MATLAB transfer function requirements, as variable-size signals are not supported. The scale array provides the absolute norm for input and output variables. The decoding_array is essential for decoding weights, biases, and producing the final output.

Parameters

Weights	W_conc	<1x18 double>
Biases	b_conc	<1x7 double>
Scale	scale	
decoder	decoding_matrix	<4x2 double>

Figure 4.11: Properties of the Simulink block for MLP deployment. These are the 4 variables that contains all the information about the MLP.

4.1.6 Training material & conclusion

Finally a tool-chain that is able to let the user customise the MLP in Matlab while training it on Python was created. Moreover, after the training is completed, this versatile framework allows for the potential deployment of the neural network in MATLAB, Simulink, or Python in according to the user's needs. The tool-chain is provided together with some files in order to train the user to use the tool-chain itself. Among these files there is a `readme.txt` and a video registration `.mp4` together with its `.pptm` source file.

4.2 Toolkit

The toolchain just explained in the previous section 4.1 is then used into another toolkit wholly dedicated to the behaviour cloning of MPCs, in this case a Path Tracking (PT) application. This toolkit, as done for the toolchain since was developed from scratch as well. It is specifically designed for the automated process from creating MPC systems through ACADO to deploy the final MLP in the experimental. of course This involves not only generating MPCs but also creating a dataset based on simulations of the MPCs following the methodologies shown in subsection 3.6.2.

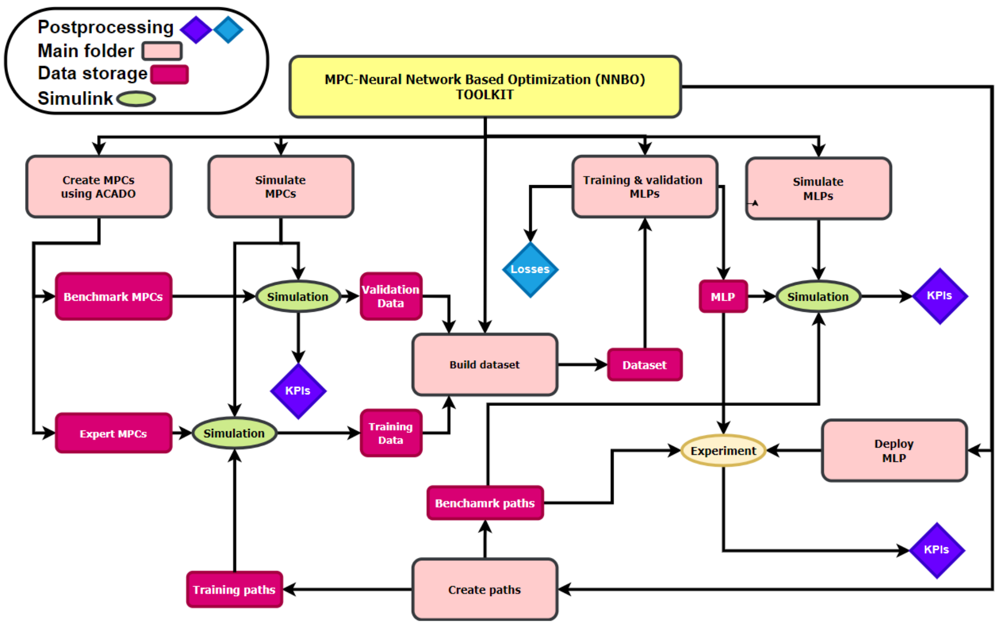
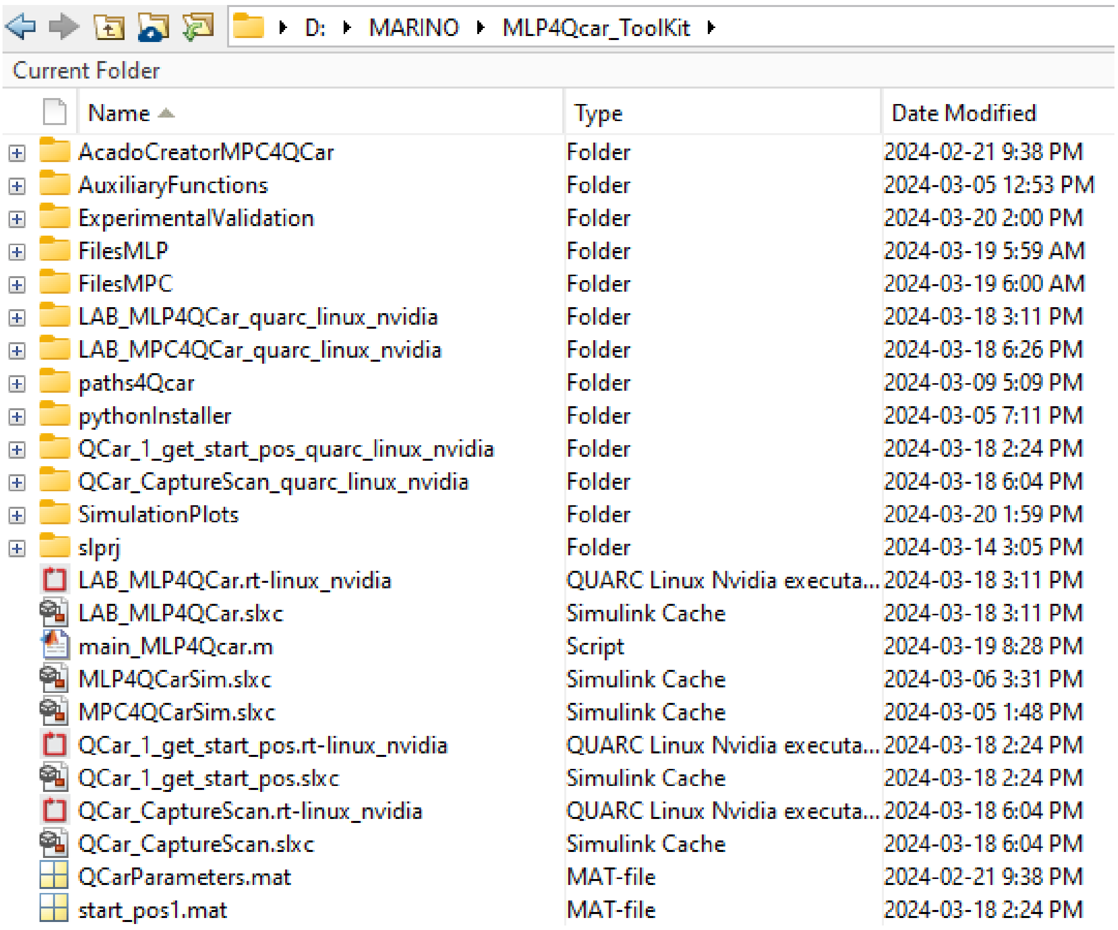


Figure 4.12: Developed toolkit structure scheme.

The toolkit is designed to be user-friendly, requiring minimal setup. Users can simply run the main file named `main_MLP4QCar.m` after ensuring that the MATLAB working folder is correctly set. Once executed, all subsequent actions can be performed directly from the MATLAB command window. This approach was chosen as the optimal solution, considering the complexity of the workflow described at the beginning of Chapter 4.

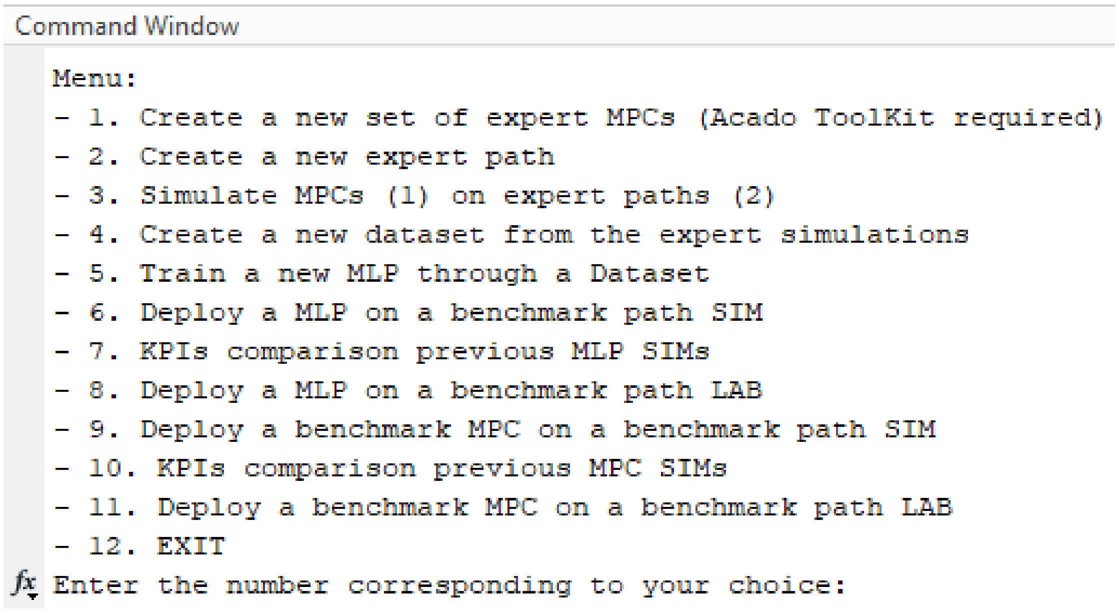


Name	Type	Date Modified
AcadoCreatorMPC4QCar	Folder	2024-02-21 9:38 PM
AuxiliaryFunctions	Folder	2024-03-05 12:53 PM
ExperimentalValidation	Folder	2024-03-20 2:00 PM
FilesMLP	Folder	2024-03-19 5:59 AM
FilesMPC	Folder	2024-03-19 6:00 AM
LAB_MLP4QCar_quarc_linux_nvidia	Folder	2024-03-18 3:11 PM
LAB_MPC4QCar_quarc_linux_nvidia	Folder	2024-03-18 6:26 PM
paths4Qcar	Folder	2024-03-09 5:09 PM
pythonInstaller	Folder	2024-03-05 7:11 PM
QCar_1_get_start_pos_quarc_linux_nvidia	Folder	2024-03-18 2:24 PM
QCar_CaptureScan_quarc_linux_nvidia	Folder	2024-03-18 6:04 PM
SimulationPlots	Folder	2024-03-20 1:59 PM
slprj	Folder	2024-03-14 3:05 PM
LAB_MLP4QCar.rt-linux_nvidia	QUARC Linux Nvidia executa...	2024-03-18 3:11 PM
LAB_MLP4QCar.slxc	Simulink Cache	2024-03-18 3:11 PM
main_MLP4Qcar.m	Script	2024-03-19 8:28 PM
MLP4QCarSim.slxc	Simulink Cache	2024-03-06 3:31 PM
MPC4QCarSim.slxc	Simulink Cache	2024-03-05 1:48 PM
QCar_1_get_start_pos.rt-linux_nvidia	QUARC Linux Nvidia executa...	2024-03-18 2:24 PM
QCar_1_get_start_pos.slxc	Simulink Cache	2024-03-18 2:24 PM
QCar_CaptureScan.rt-linux_nvidia	QUARC Linux Nvidia executa...	2024-03-18 6:04 PM
QCar_CaptureScan.slxc	Simulink Cache	2024-03-18 6:04 PM
QCarParameters.mat	MAT-file	2024-02-21 9:38 PM
start_pos1.mat	MAT-file	2024-03-18 2:24 PM

Figure 4.13: How the MATLAB folder should look like while using the toolkit.

4.2.1 GUI Functionalities

The complexity of the problem necessitates extensive trial and error, demanding automation of the process. For instance, the creation of the set of LMPCs alone would be time-consuming if not automated. Similarly, simulating datasets of varying shapes and training MLPs followed by simulation would be a daunting task without automation. Essentially, this toolkit is tailored for the QCar controller NNBO problem but can be adapted for any system and any MPC controller by adjusting the settings in ACADO. Below is the main menu of the toolkit when the script is executed in MATLAB.

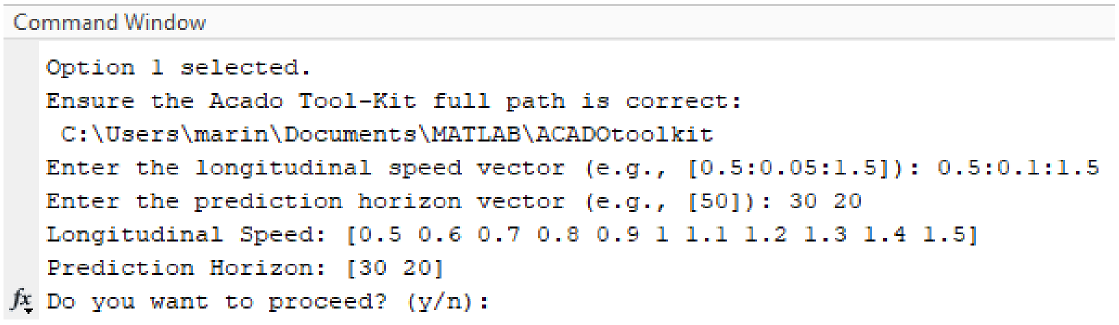


```
Menu:
- 1. Create a new set of expert MPCs (Acado ToolKit required)
- 2. Create a new expert path
- 3. Simulate MPCs (1) on expert paths (2)
- 4. Create a new dataset from the expert simulations
- 5. Train a new MLP through a Dataset
- 6. Deploy a MLP on a benchmark path SIM
- 7. KPIs comparison previous MLP SIMs
- 8. Deploy a MLP on a benchmark path LAB
- 9. Deploy a benchmark MPC on a benchmark path SIM
- 10. KPIs comparison previous MPC SIMs
- 11. Deploy a benchmark MPC on a benchmark path LAB
- 12. EXIT
Enter the number corresponding to your choice:
```

Figure 4.14: Toolkit main GUI.

At this point the user just ensured the toolkit to be set in a proper way.

Below is the menu that appears when the user presses the number corresponding to the submenu. When creating the LMPC, the program first checks for the presence of the Acado Toolkit in the path. If Acado is not found, the program prompts the user to provide the correct path. In case Acado needs to be downloaded, the toolkit itself provides a link leading to the Acado Toolkit download page.



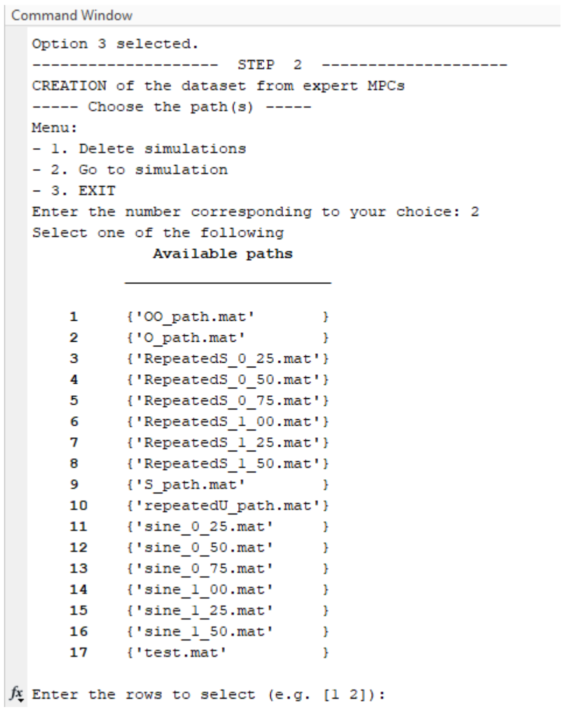
```
Option 1 selected.
Ensure the Acado Tool-Kit full path is correct:
 C:\Users\marin\Documents\MATLAB\ACADOtoolkit
Enter the longitudinal speed vector (e.g., [0.5:0.05:1.5]): 0.5:0.1:1.5
Enter the prediction horizon vector (e.g., [50]): 30 20
Longitudinal Speed: [0.5 0.6 0.7 0.8 0.9 1 1.1 1.2 1.3 1.4 1.5]
Prediction Horizon: [30 20]
Do you want to proceed? (y/n):
```

Figure 4.15: MPC creation dedicated GUI.

The provided values consist of two vectors: the first vector represents the list of linearised points, while the second vector represents the prediction horizon. Upon pressing enter, the created LMPCs will be a combination of these two vectors. Therefore, if the first vector has length N and the second vector has length M, the total number of MPCs created will be $N \times M$.

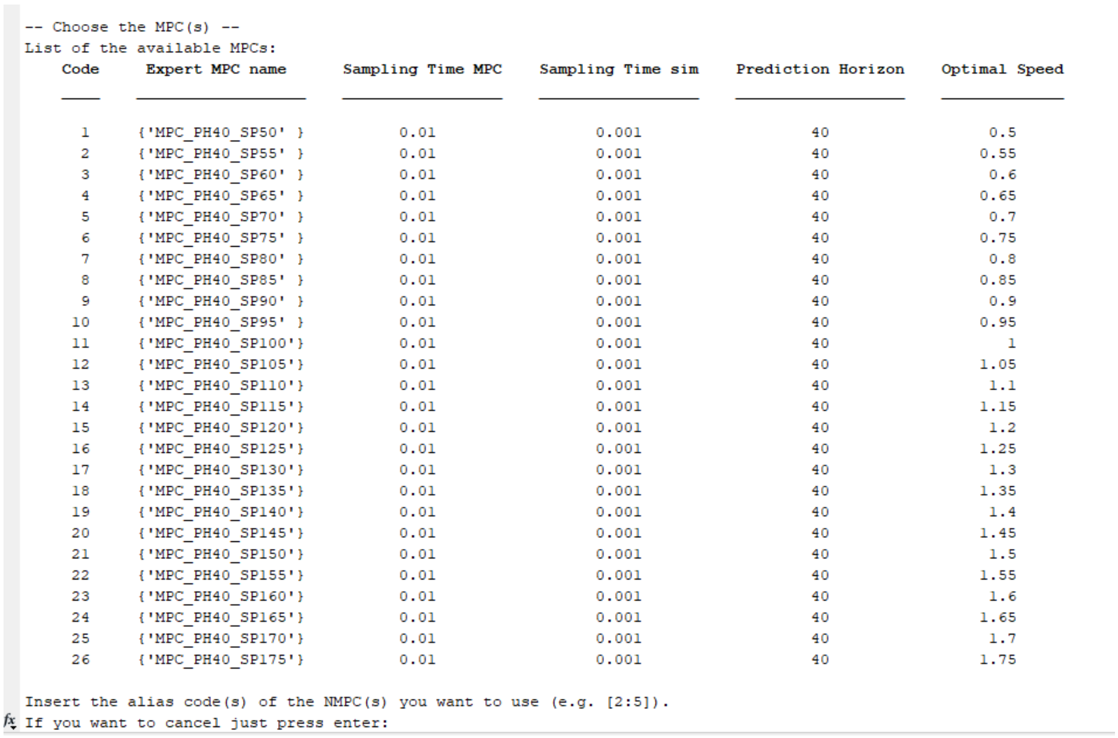
Option"2" instead leads the user to another script where they can construct the desired XY path. This process is not detailed here due to space limitations and because it may vary depending on the application. It's important to note that this toolkit is designed to be general-purpose, and the example provided is just one of the many possible applications.

The next step involves simulating the MPCs over the paths created in the previous step. This simulation marks the initial stage of dataset creation. Here, a list of the MPCs and training paths generated in the preceding steps is displayed as in figure 4.16 and 4.17. At this point any combination of them is simulated. The data generated from these simulations are then systematically stored in a dedicated folder within the toolkit.



```
Command Window
Option 3 selected.
--------------------  STEP  2  --------------------
CREATION of the dataset from expert MPCs
----- Choose the path(s) -----
Menu:
- 1. Delete simulations
- 2. Go to simulation
- 3. EXIT
Enter the number corresponding to your choice: 2
Select one of the following
             Available paths
          ______________________

    1     {'OO_path.mat'       }
    2     {'O_path.mat'        }
    3     {'RepeatedS_0_25.mat'}
    4     {'RepeatedS_0_50.mat'}
    5     {'RepeatedS_0_75.mat'}
    6     {'RepeatedS_1_00.mat'}
    7     {'RepeatedS_1_25.mat'}
    8     {'RepeatedS_1_50.mat'}
    9     {'S_path.mat'        }
    10    {'repeatedU_path.mat'}
    11    {'sine_0_25.mat'     }
    12    {'sine_0_50.mat'     }
    13    {'sine_0_75.mat'     }
    14    {'sine_1_00.mat'     }
    15    {'sine_1_25.mat'     }
    16    {'sine_1_50.mat'     }
    17    {'test.mat'          }

fx Enter the rows to select (e.g. [1 2]):
```

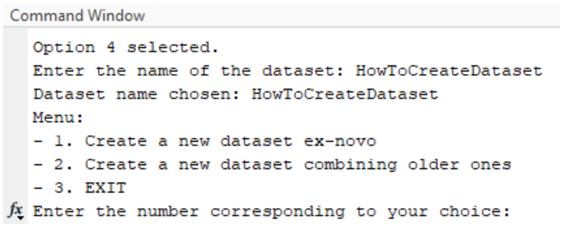
Figure 4.16: MPC simulation, list of available paths GUI.



```
-- Choose the MPC(s) --
List of the available MPCs:
    Code      Expert MPC name       Sampling Time MPC    Sampling Time sim    Prediction Horizon    Optimal Speed
    ____     __________________     _________________    _________________    __________________    _____________

     1       {'MPC_PH40_SP50' }           0.01                 0.001                  40                 0.5
     2       {'MPC_PH40_SP55' }           0.01                 0.001                  40                0.55
     3       {'MPC_PH40_SP60' }           0.01                 0.001                  40                 0.6
     4       {'MPC_PH40_SP65' }           0.01                 0.001                  40                0.65
     5       {'MPC_PH40_SP70' }           0.01                 0.001                  40                 0.7
     6       {'MPC_PH40_SP75' }           0.01                 0.001                  40                0.75
     7       {'MPC_PH40_SP80' }           0.01                 0.001                  40                 0.8
     8       {'MPC_PH40_SP85' }           0.01                 0.001                  40                0.85
     9       {'MPC_PH40_SP90' }           0.01                 0.001                  40                 0.9
    10       {'MPC_PH40_SP95' }           0.01                 0.001                  40                0.95
    11       {'MPC_PH40_SP100'}           0.01                 0.001                  40                   1
    12       {'MPC_PH40_SP105'}           0.01                 0.001                  40                1.05
    13       {'MPC_PH40_SP110'}           0.01                 0.001                  40                 1.1
    14       {'MPC_PH40_SP115'}           0.01                 0.001                  40                1.15
    15       {'MPC_PH40_SP120'}           0.01                 0.001                  40                 1.2
    16       {'MPC_PH40_SP125'}           0.01                 0.001                  40                1.25
    17       {'MPC_PH40_SP130'}           0.01                 0.001                  40                 1.3
    18       {'MPC_PH40_SP135'}           0.01                 0.001                  40                1.35
    19       {'MPC_PH40_SP140'}           0.01                 0.001                  40                 1.4
    20       {'MPC_PH40_SP145'}           0.01                 0.001                  40                1.45
    21       {'MPC_PH40_SP150'}           0.01                 0.001                  40                 1.5
    22       {'MPC_PH40_SP155'}           0.01                 0.001                  40                1.55
    23       {'MPC_PH40_SP160'}           0.01                 0.001                  40                 1.6
    24       {'MPC_PH40_SP165'}           0.01                 0.001                  40                1.65
    25       {'MPC_PH40_SP170'}           0.01                 0.001                  40                 1.7
    26       {'MPC_PH40_SP175'}           0.01                 0.001                  40                1.75

Insert the alias code(s) of the NMPC(s) you want to use (e.g. [2:5]).
fx If you want to cancel just press enter:
```

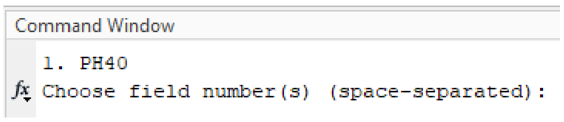
Figure 4.17: MPC simulation, list of available MPCs GUI.

With the simulations now finally stored, the next option in the toolkit is to create a dataset. Users have the option to either mix preexisting datasets or create a new dataset from scratch.



```
Command Window
Option 4 selected.
Enter the name of the dataset: HowToCreateDataset
Dataset name chosen: HowToCreateDataset
Menu:
- 1. Create a new dataset ex-novo
- 2. Create a new dataset combining older ones
- 3. EXIT
fx Enter the number corresponding to your choice:
```

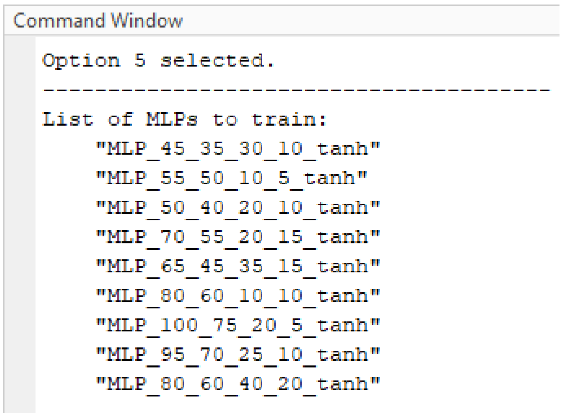
Figure 4.18: Dataset creation menu.



```
Command Window
1. PH40
fx Choose field number(s) (space-separated):
```

Figure 4.19: In dataset creation, the choice of prediction horizon is crucial. This selection determines the number of columns in the dataset, as illustrated in Equation 3.8.

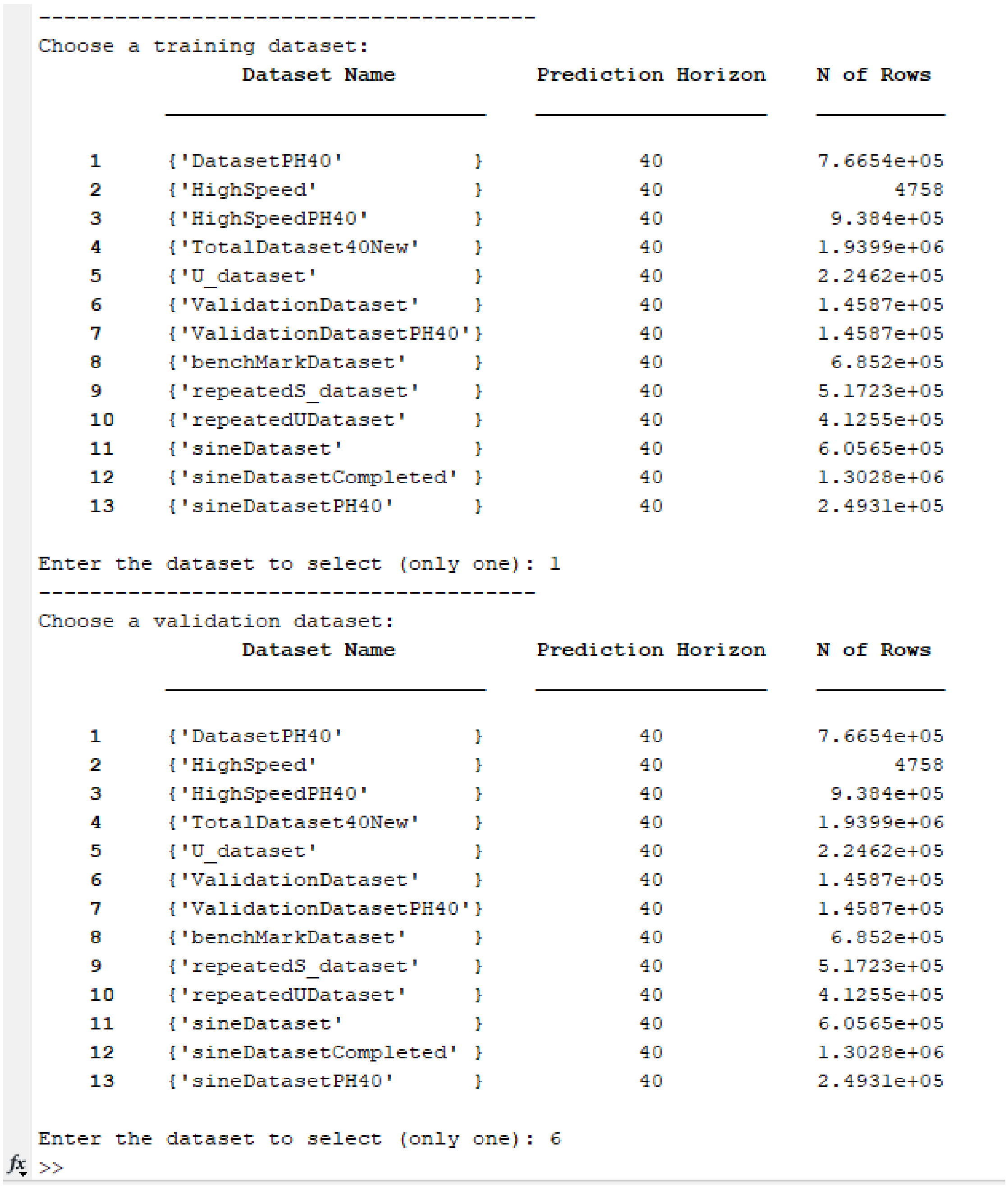
The subsequent step involves utilising the created dataset to train the MLPs. It's essential to specify the setup, including the depth, layer widths, and activation functions of the MLP, in the main script `main_MLP4QCars.m`. The hyperparameters instead can be modified going to the function that manage the entire process Once the training process is initiated, a list of MLPs that the software will train is displayed, as shown in Figure 4.20.



```
Command Window
Option 5 selected.
---------------------------------------
List of MLPs to train:
    "MLP_45_35_30_10_tanh"
    "MLP_55_50_10_5_tanh"
    "MLP_50_40_20_10_tanh"
    "MLP_70_55_20_15_tanh"
    "MLP_65_45_35_15_tanh"
    "MLP_80_60_10_10_tanh"
    "MLP_100_75_20_5_tanh"
    "MLP_95_70_25_10_tanh"
    "MLP_80_60_40_20_tanh"
```

Figure 4.20: MLP training, list of MLPs.

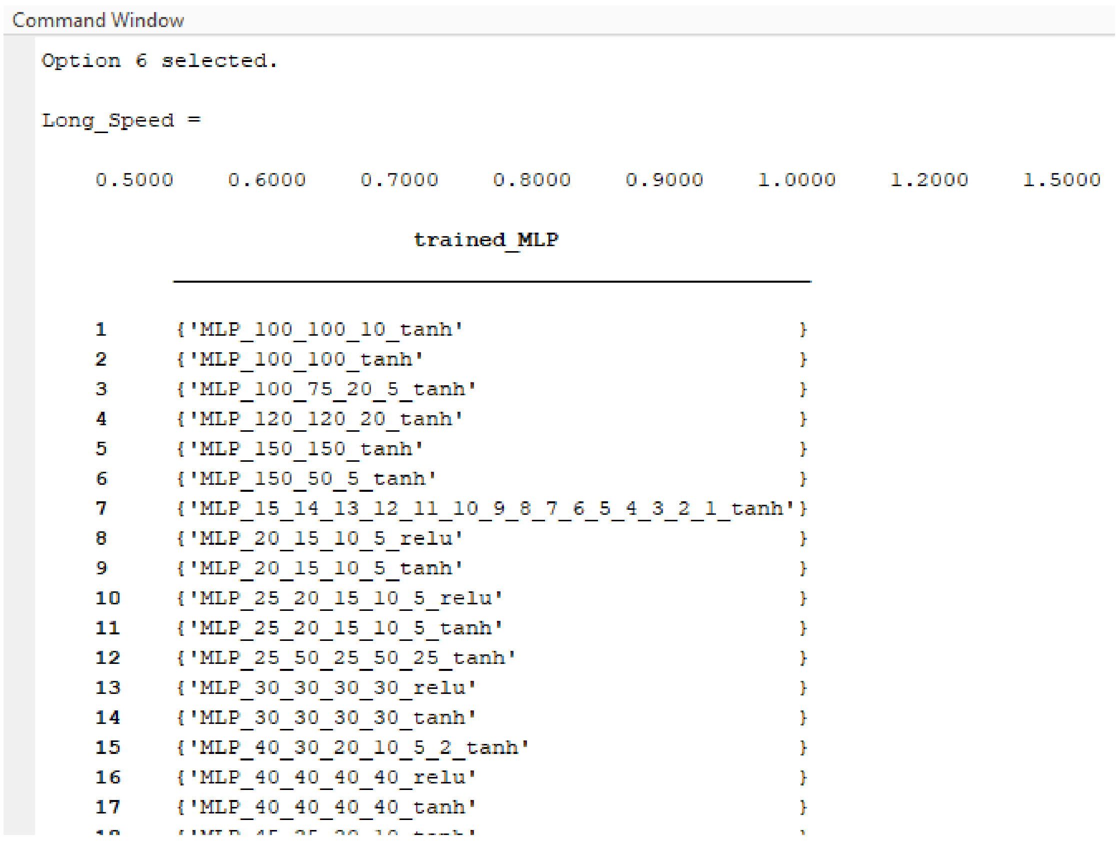
Subsequently, the user will select the datasets to be used, followed by choosing the training and validation datasets. A screenshot of the graphical user interface (GUI) that appears is provided on the next page. It is possible to request the training of multiple MLPs by adding them to the main script. In such cases, the list of MLPs to be trained will include more than one, as shown in Figure 4.20. The process is fully automated, but the settings for each MLP and dataset remain consistent. If a previously created MLP is detected among those stored in the dedicated folder, the toolkit will prompt the user to decide whether to continue or not.



```
--------------------------------------
Choose a training dataset:
                Dataset Name            Prediction Horizon    N of Rows

          _________________________    __________________    __________


    1     {'DatasetPH40'            }          40            7.6654e+05
    2     {'HighSpeed'              }          40                  4758
    3     {'HighSpeedPH40'          }          40             9.384e+05
    4     {'TotalDataset40New'      }          40            1.9399e+06
    5     {'U_dataset'              }          40            2.2462e+05
    6     {'ValidationDataset'      }          40            1.4587e+05
    7     {'ValidationDatasetPH40'}            40            1.4587e+05
    8     {'benchMarkDataset'       }          40             6.852e+05
    9     {'repeatedS_dataset'      }          40            5.1723e+05
    10    {'repeatedUDataset'       }          40            4.1255e+05
    11    {'sineDataset'            }          40            6.0565e+05
    12    {'sineDatasetCompleted'   }          40            1.3028e+06
    13    {'sineDatasetPH40'        }          40            2.4931e+05

Enter the dataset to select (only one): 1
--------------------------------------
Choose a validation dataset:
                Dataset Name            Prediction Horizon    N of Rows

          _________________________    __________________    __________


    1     {'DatasetPH40'            }          40            7.6654e+05
    2     {'HighSpeed'              }          40                  4758
    3     {'HighSpeedPH40'          }          40             9.384e+05
    4     {'TotalDataset40New'      }          40            1.9399e+06
    5     {'U_dataset'              }          40            2.2462e+05
    6     {'ValidationDataset'      }          40            1.4587e+05
    7     {'ValidationDatasetPH40'}            40            1.4587e+05
    8     {'benchMarkDataset'       }          40             6.852e+05
    9     {'repeatedS_dataset'      }          40            5.1723e+05
    10    {'repeatedUDataset'       }          40            4.1255e+05
    11    {'sineDataset'            }          40            6.0565e+05
    12    {'sineDatasetCompleted'   }          40            1.3028e+06
    13    {'sineDatasetPH40'        }          40            2.4931e+05

Enter the dataset to select (only one): 6
fx >>
```

Figure 4.21: MLP training, choice of the datasets.

Now the simulation deployment comes. As in the previous cases the GUI still consists in the list of controllers. In this case the list of controllers instead of being composed by MCP like in the first step 4.17 it is composed now but trained MLP. Then the user will choice one or more MLP to deploy.

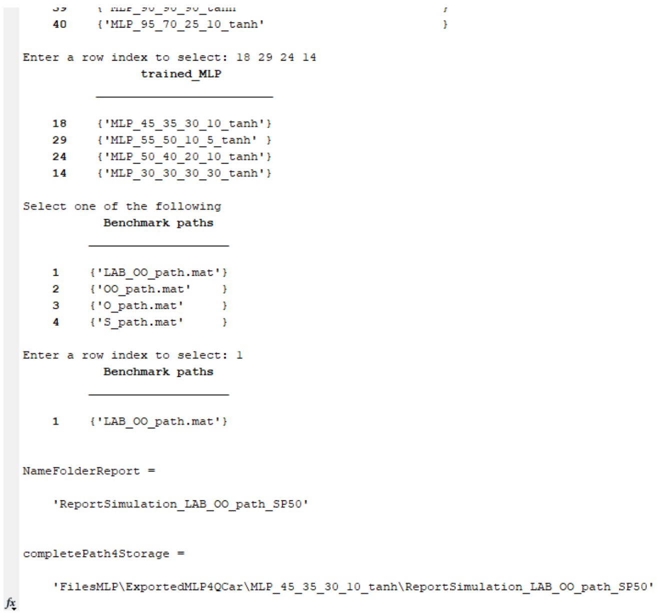


Command Window

```
Option 6 selected.

Long_Speed =

    0.5000    0.6000    0.7000    0.8000    0.9000    1.0000    1.2000    1.5000

                            trained_MLP
          ________________________________________________

    1     {'MLP_100_100_10_tanh'                        }
    2     {'MLP_100_100_tanh'                           }
    3     {'MLP_100_75_20_5_tanh'                       }
    4     {'MLP_120_120_20_tanh'                        }
    5     {'MLP_150_150_tanh'                           }
    6     {'MLP_150_50_5_tanh'                          }
    7     {'MLP_15_14_13_12_11_10_9_8_7_6_5_4_3_2_1_tanh'}
    8     {'MLP_20_15_10_5_relu'                        }
    9     {'MLP_20_15_10_5_tanh'                        }
    10    {'MLP_25_20_15_10_5_relu'                     }
    11    {'MLP_25_20_15_10_5_tanh'                     }
    12    {'MLP_25_50_25_50_25_tanh'                    }
    13    {'MLP_30_30_30_30_relu'                       }
    14    {'MLP_30_30_30_30_tanh'                       }
    15    {'MLP_40_30_20_10_5_2_tanh'                   }
    16    {'MLP_40_40_40_40_relu'                       }
    17    {'MLP_40_40_40_40_tanh'                       }
```

Figure 4.22: Developed toolkit structure scheme.

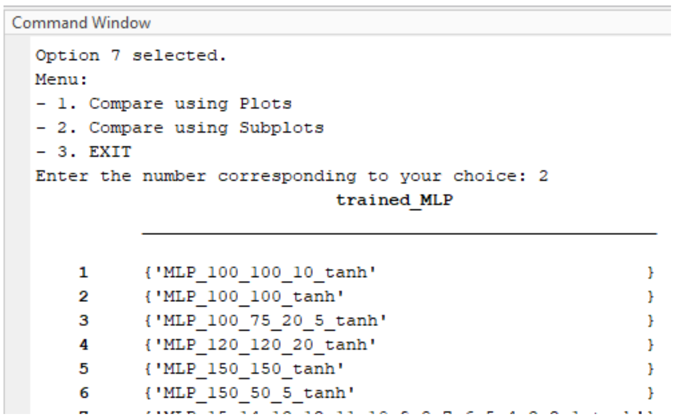
The deployment of a single MLP will involve testing it at multiple velocities, as the state now includes velocity as well. Consequently, the controller is no longer dedicated to a single longitudinal speed, as seen in the case of LMPCs. The velocities employed will be displayed in a vector within the command window. Subsequently, simulations are performed, and various reports, plots, and KPIs are stored in a dedicated folder within the toolkit.



```
    40    {'MLP_95_70_25_10_tanh'                          }

Enter a row index to select: 18 29 24 14
                trained_MLP
          ________________________

    18    {'MLP_45_35_30_10_tanh'}
    29    {'MLP_55_50_10_5_tanh' }
    24    {'MLP_50_40_20_10_tanh'}
    14    {'MLP_30_30_30_30_tanh'}

Select one of the following
             Benchmark paths
          ___________________

    1    {'LAB_OO_path.mat'}
    2    {'OO_path.mat'    }
    3    {'O_path.mat'     }
    4    {'S_path.mat'     }

Enter a row index to select: 1
             Benchmark paths
          ___________________

    1    {'LAB_OO_path.mat'}


NameFolderReport =

    'ReportSimulation_LAB_OO_path_SP50'


completePath4Storage =

    'FilesMLP\ExportedMLP4QCar\MLP_45_35_30_10_tanh\ReportSimulation_LAB_OO_path_SP50'
```

Figure 4.23: MLP deployment, example of use. Notice the full path of the dedicated folder where the result are stored is provided on the MATLAB command window.

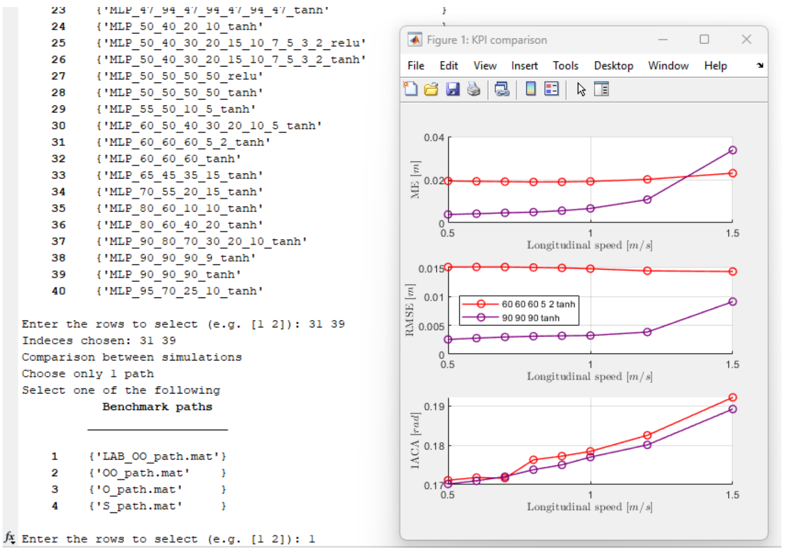
The comparison of KPIs is a crucial phase, warranting the creation of a dedicated section in the main GUI. Within this section, users can compare the KPIs of different MLPs, which can be selected using the familiar GUI. Similarly, the same functionality is available for comparing the KPIs of MPCs in their respective section of the GUI.



```
Command Window
Option 7 selected.
Menu:
- 1. Compare using Plots
- 2. Compare using Subplots
- 3. EXIT
Enter the number corresponding to your choice: 2
                             trained_MLP
          ________________________________________________

    1     {'MLP_100_100_10_tanh'                        }
    2     {'MLP_100_100_tanh'                           }
    3     {'MLP_100_75_20_5_tanh'                       }
    4     {'MLP_120_120_20_tanh'                        }
    5     {'MLP_150_150_tanh'                           }
    6     {'MLP_150_50_5_tanh'                          }
```

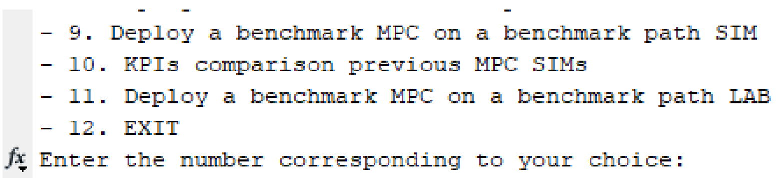
Figure 4.24: KPIs comparison.

Below is proposed an instance of the KPIs comparison.



```
23    {'MLP_47_94_47_94_47_94_47_tanh'
24    {'MLP_50_40_20_10_tanh'
25    {'MLP_50_40_30_20_15_10_7_5_3_2_relu'
26    {'MLP_50_40_30_20_15_10_7_5_3_2_tanh'
27    {'MLP_50_50_50_50_relu'
28    {'MLP_50_50_50_50_tanh'
29    {'MLP_55_50_10_5_tanh'
30    {'MLP_60_50_40_30_20_10_5_tanh'
31    {'MLP_60_60_60_5_2_tanh'
32    {'MLP_60_60_60_tanh'
33    {'MLP_65_45_35_15_tanh'
34    {'MLP_70_55_20_15_tanh'
35    {'MLP_80_60_10_10_tanh'
36    {'MLP_80_60_40_20_tanh'
37    {'MLP_90_80_70_30_20_10_tanh'
38    {'MLP_90_90_90_9_tanh'
39    {'MLP_90_90_90_tanh'
40    {'MLP_95_70_25_10_tanh'

Enter the rows to select (e.g. [1 2]): 31 39
Indeces chosen: 31 39
Comparison between simulations
Choose only 1 path
Select one of the following
           Benchmark paths

         _____________________


   1    {'LAB_OO_path.mat'}
   2    {'OO_path.mat'    }
   3    {'O_path.mat'     }
   4    {'S_path.mat'     }

fx Enter the rows to select (e.g. [1 2]): 1
```

Figure 4.25: KPIs comparison, subplots.



```
- 9. Deploy a benchmark MPC on a benchmark path SIM
- 10. KPIs comparison previous MPC SIMs
- 11. Deploy a benchmark MPC on a benchmark path LAB
- 12. EXIT
fx Enter the number corresponding to your choice:
```

Figure 4.26: The remainder of the toolkit's main GUI includes additional options. Option 9 offers the same functionalities as option 7, but it deploys the MLP in the real environment, performing experimental tests. Similarly, options 10, 11, and 12 mirror options 7, 8, and 9, respectively, but are tailored for MPCs benchmark controllers.

4.2.2 Training Material & Conclusion

The training material for the University of Surrey has not been created yet due to time constraints. Nevertheless, this toolkit represents a novel workflow programming approach, enabling the entire process from creating an MPC to comparing the KPIs of the experimental tests of the trained MLPs to be completed in very few steps.

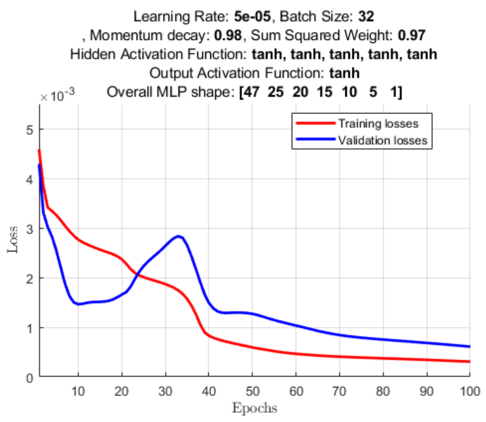
Chapter 5

Training & Validation

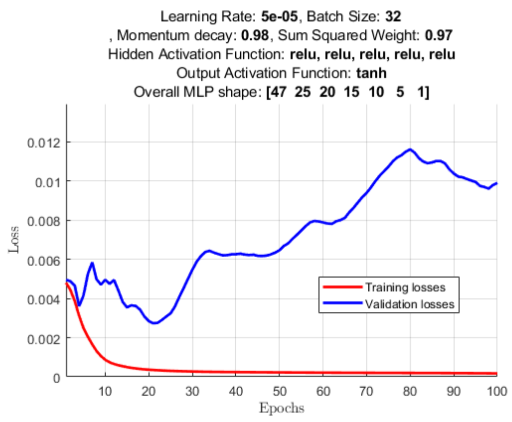
In this chapter, only data regarding the training and validation phases are reported. The methodologies used are those reported in sections 3.7.1 and 3.7.2. The philosophy employed involves performing a quick sensitivity analysis over the shape of the MLP, the number of layers and neurons, and the best activation functions. The hyperparameters are chosen according to the methodologies outlined in subsection 3.7.2, particularly in table 3.3.

5.1 Losses: ReLu vs tanh

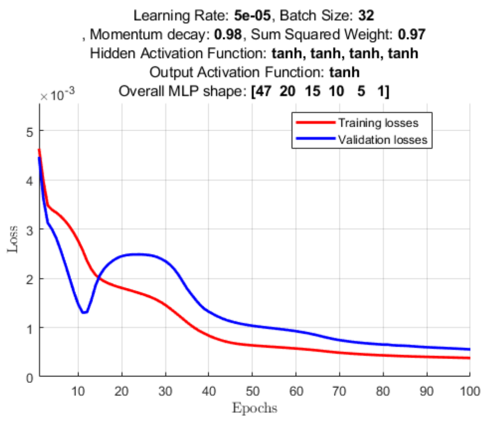
In this section, the plots compare the losses of two identical trainings, differing only in the activation function used in the hidden layers.



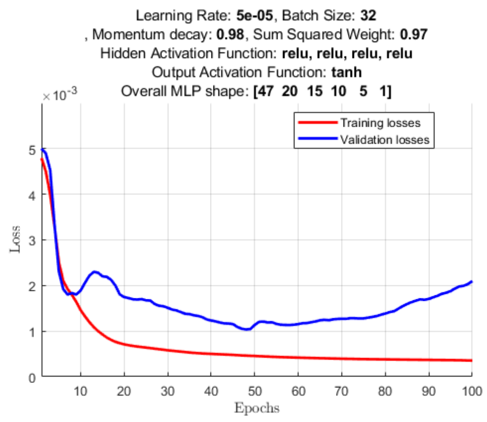
(a) Losses loss with ReLu function



(b) Losses loss with tanh function

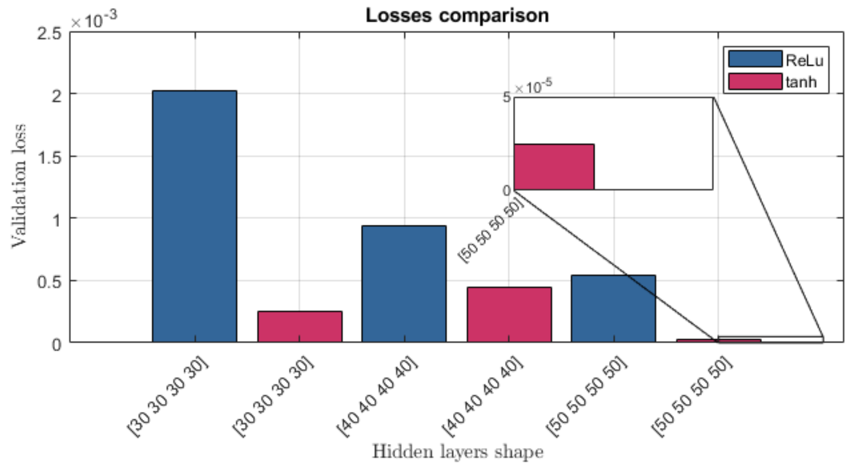


(c) Losses with ReLu function

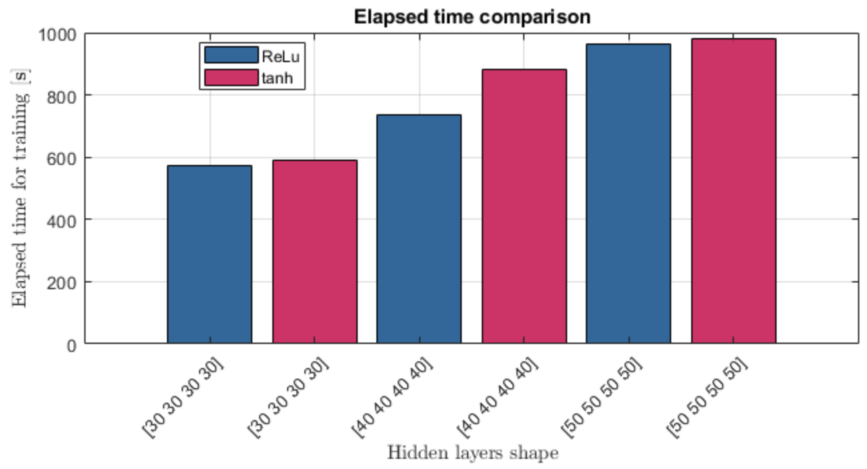


(d) Losses loss with tanh function

Figure 5.1: When comparing the losses from ReLU hidden layers with those from tanh hidden layers over epochs, a notable pattern emerges. Specifically, the tanh activation functions exhibit a somewhat analogous behaviour, characterised by a slight spike at the beginning of training. In contrast, ReLU training tends to follow a more linear trajectory, culminating eventually in overfitting. This discrepancy underscores the different dynamics between the two activation functions and their impact on the training process.

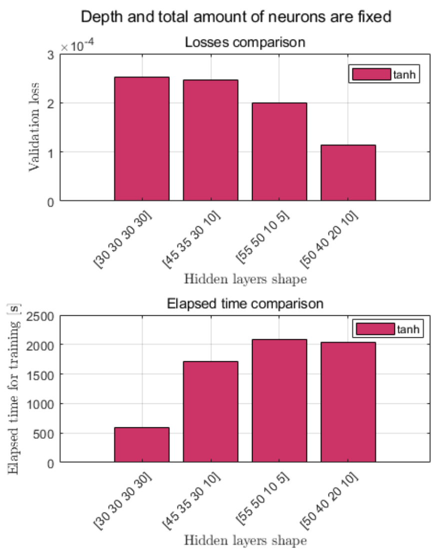


(a) The validation loss is calculated based on the performance of the model on the validation dataset, which was generated by running Model Predictive Control (MPC) on the eight path.

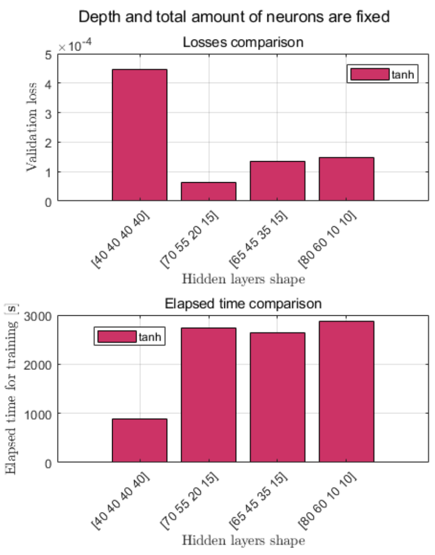


(b) The elapsed time, expressed in seconds, encompasses the entire process, including both training and data preprocessing steps such as normalisation.

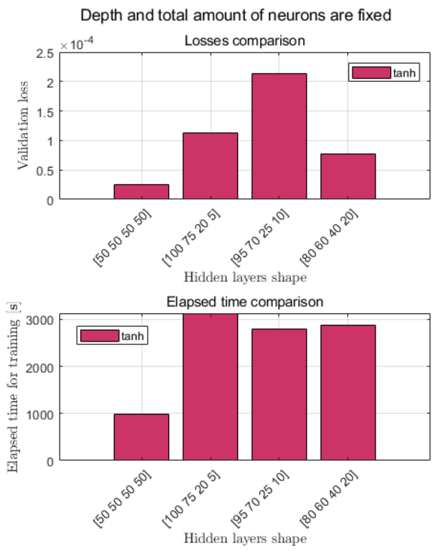
Figure 5.2: The plot shows validation losses and elapsed time for training for six types of MLPs, with three different structures all featuring the same depth (four hidden layers). Initially, ReLU activation function was employed, followed by tanh activation. Despite slightly longer elapsed times, the MLP using tanh activation exhibited superior validation losses. Consequently, the marginal increase in time seems justified given the significant reduction in validation losses. This observation suggests that the tanh activation function in hidden layers is more suitable for control applications, as indicated by sensitivity analysis.



(a) Depth = 4;
Neurons = 120;



(b) Depth = 4;
Neurons = 160;



(c) Depth = 4;
Neurons = 200;

Figure 5.3: Comparison between losses from ReLu hidden layers and tanh hidden layers. Note that MLPs whose hidden layers have the same width are faster to train. Moreover, sometimes higher elapsed times do not imply lower losses (e.g. MLPs [55 50 10 5] and [50 40 10 5]).

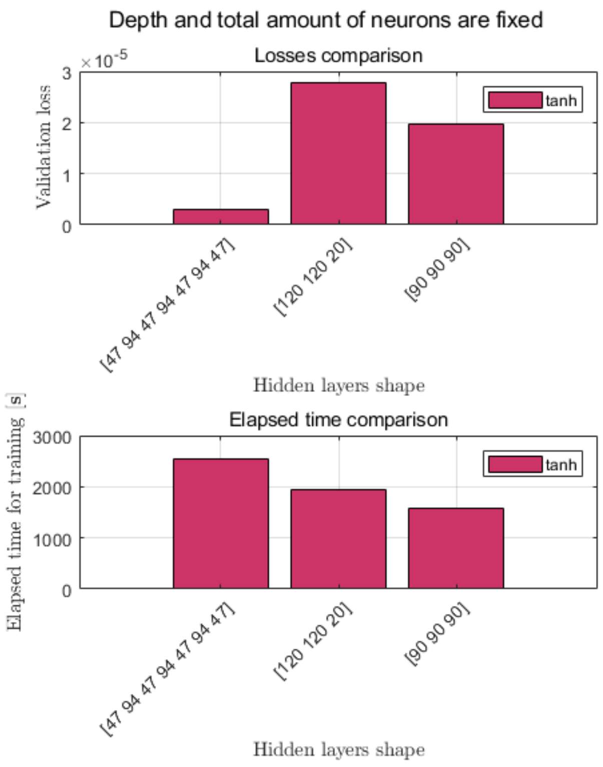


Figure 5.4: Some of other hidden structures tried. It is noticeable how the depth affects elapsed time and final validation loss.

Chapter 6

Simulation Results

The simulations are conducted within the Simulink environment. The model used for testing the MLPs is analytical and is derived from prior studies on the SSRAV (i.e. QCar). It's important to note that there exists a noticeable dissimilarity between the analytical model and the real plant, as demonstrated in the experimental findings discussed in Chapter 7. Therefore, the efficacy of the controller on the model does not guarantee its effectiveness on the actual plant. The methodology employed for data collection in the simulation is 'you-measure-only-once.' This is because simulation results remain unaffected by stochastic events, unlike real-world experiments. Consequently, in the absence of noise, the results, and consequently the KPIs, remain consistent.

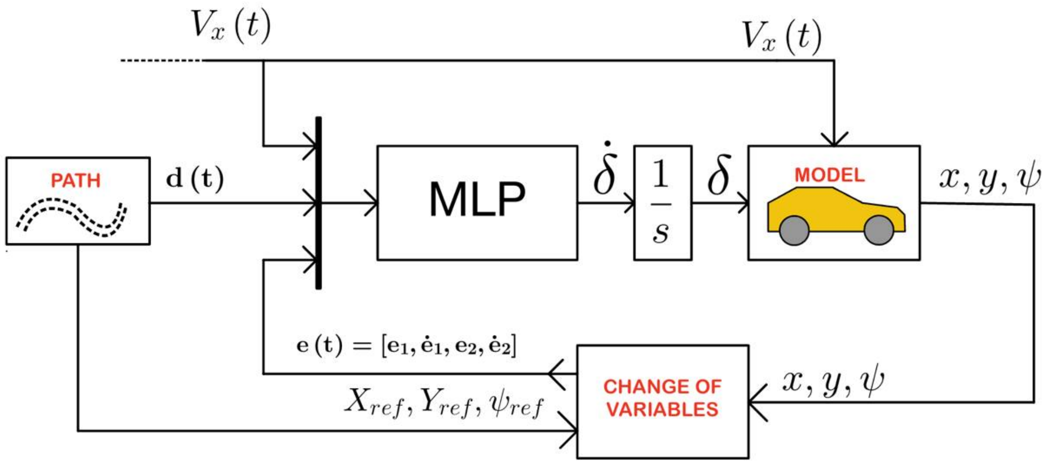
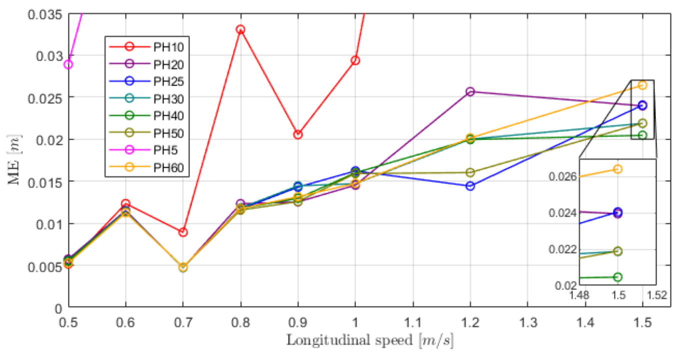


Figure 6.1: Simulink model schematic for simulations.

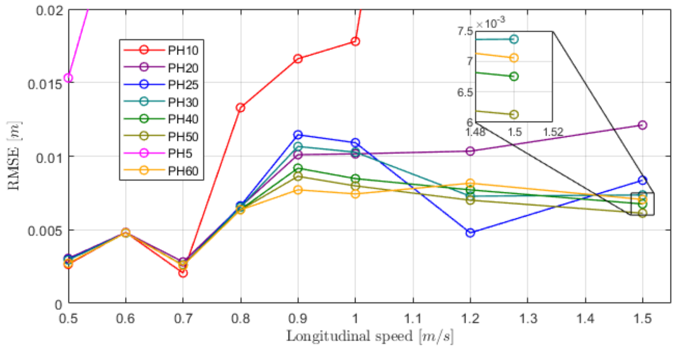
The signals utilised in the simulations, both input and output, correspond to those present in the real model. However, unlike the real scenario, these signals are not subjected to processing by a Kalman filter. This deviation is due to the absence of instrumental noise the absence of inaccuracies caused by the measurements.

From here, several plots illustrating the most representative simulations are presented. The evaluation is focused on the KPIs described in Section 3.2.
In figure 6.2 Multiple LMPCs, differing only in their prediction horizon lengths, are simulated to determine if there is an optimal horizon length greater than 20 steps. As previously discussed in Section 3.1 and summarised in Table 3.2, various prediction horizon lengths were evaluated based on recorded KPIs from simulations. The results indicate that the optimal prediction horizon length, according to the observed KPIs, is 40 steps. Of course this length depends also on the kind of path there the simulation are performed.

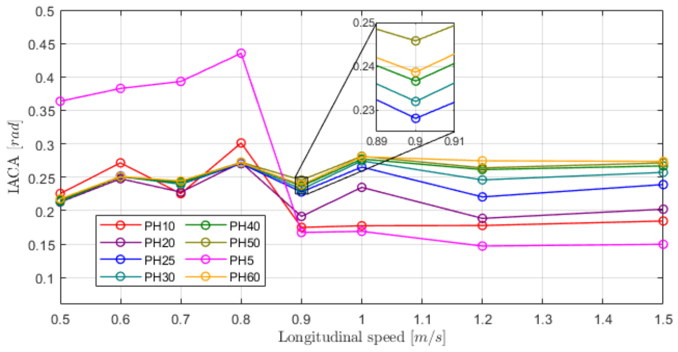
6.1 Simulation MPCs



(a) Maximum lateral error.



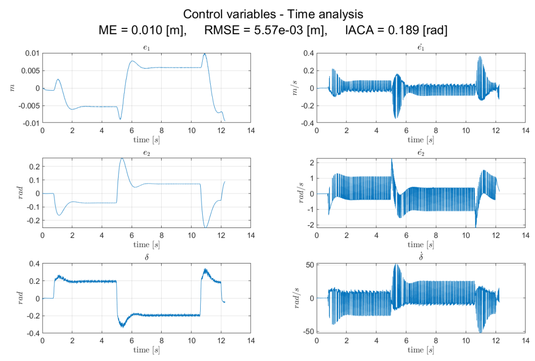
(b) Root mean square lateral error.



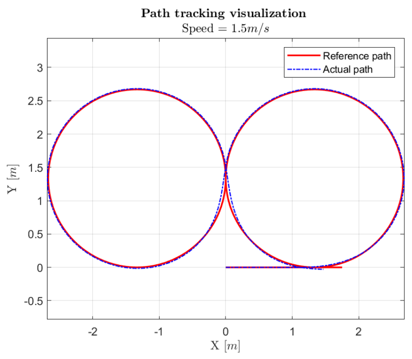
(c) Integral of absolute action control.

Figure 6.2: Sensitivity analysis of KPIs over longitudinal velocity from MPC simulations aimed at determining the optimal prediction horizon.

6.2 Simulation MLPs

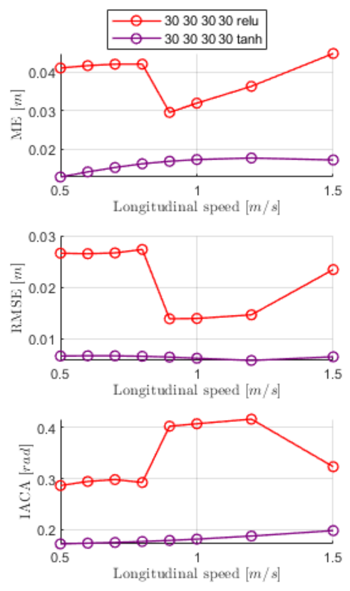


(a) Control variables over time

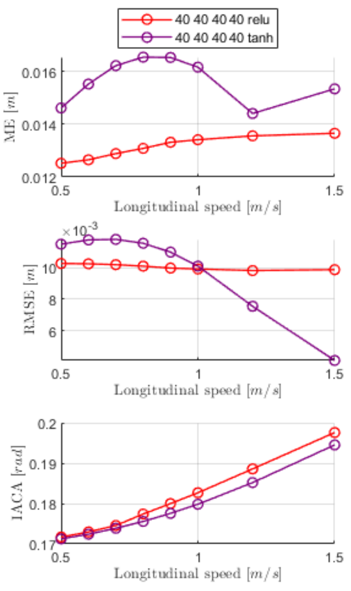


(b) xy view

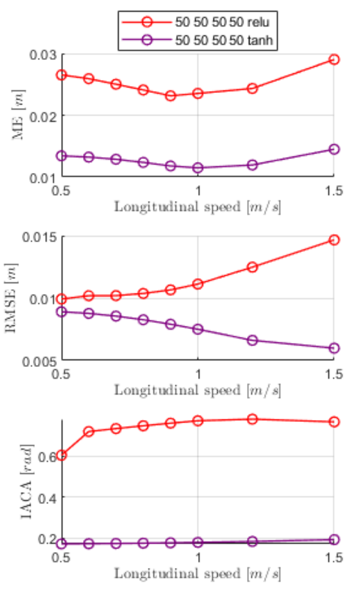
Figure 6.3: Simulation Results for MLP with architecture (120-120-20) and tanh activation function at 1.5 m/s. Even at higher speeds the errors are still acceptable. This because the simulated model is not too afar from the internal model used in the set of LMPCs. Moreover the training was well done for this MLP.



(a)

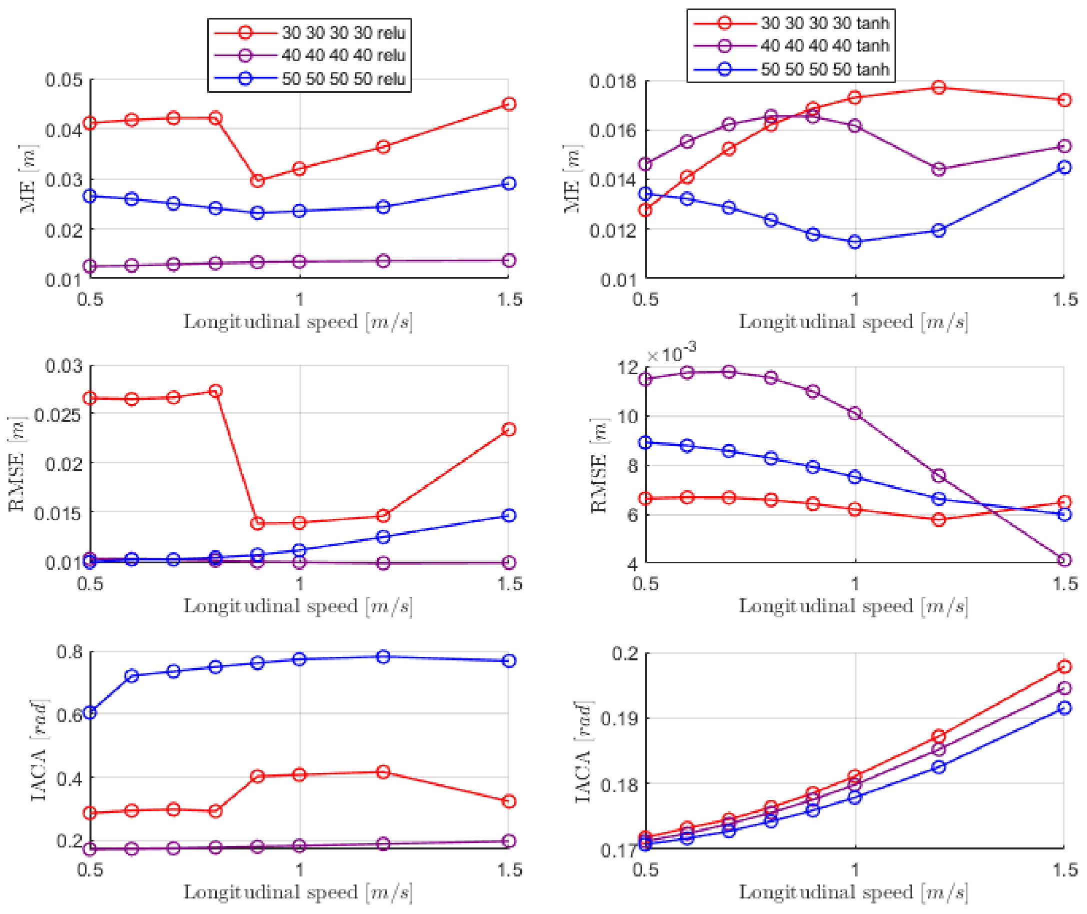


(b)



(c)

Figure 6.4: The plots above depict a sensitivity analysis conducted on activation functions. Although the number of neurons varies, the overall shape remains consistent. These plots represent MLPs with hidden layers of constant width. It's important to note that the training settings for all experiments remain uniform, as outlined in the default training table (Table 3.3).



(a) (b)

Figure 6.5: Sensitivity analysis was conducted over hidden layer sizes, revealing that increasing the width generally leads to improvements in KPIs. This trend is particularly notable for networks utilising the tanh activation function.

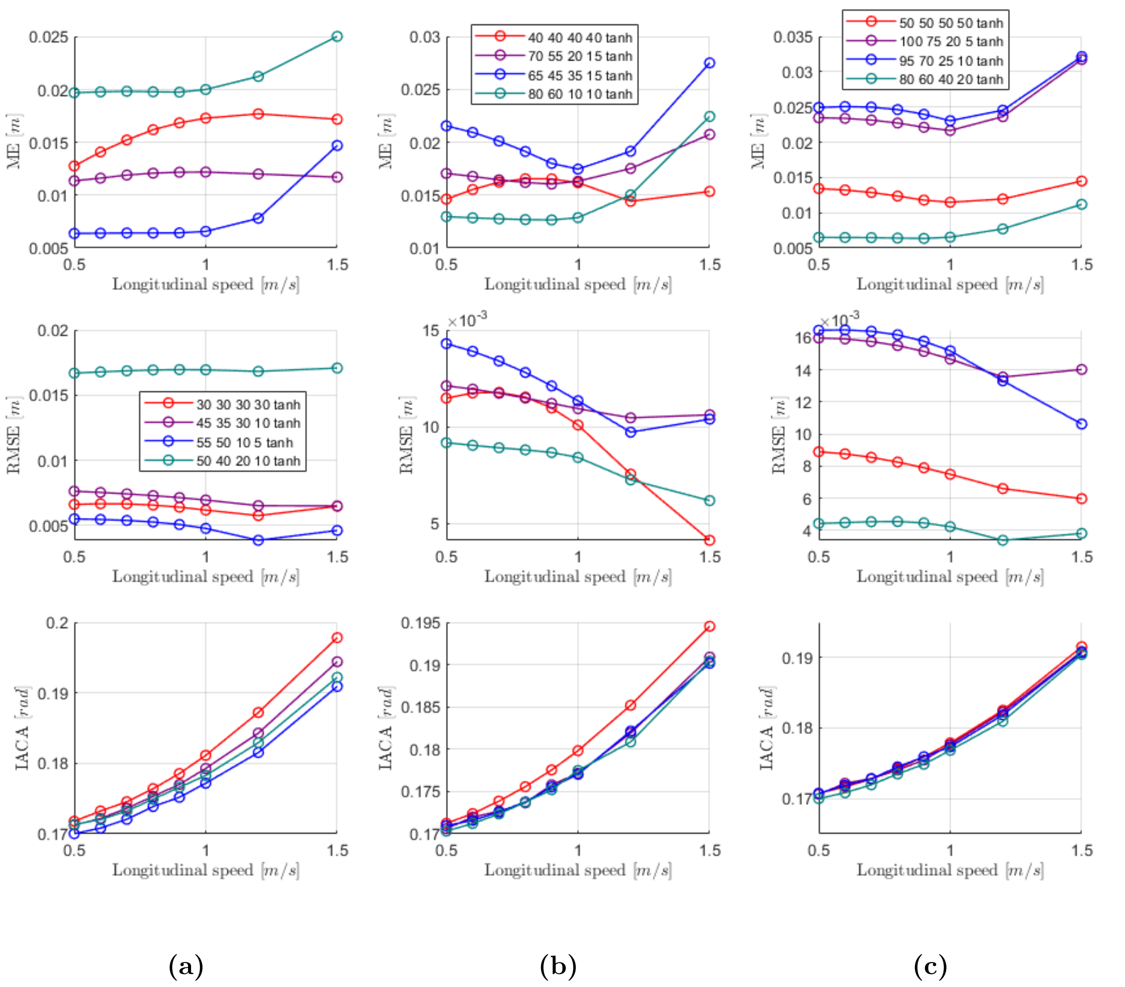
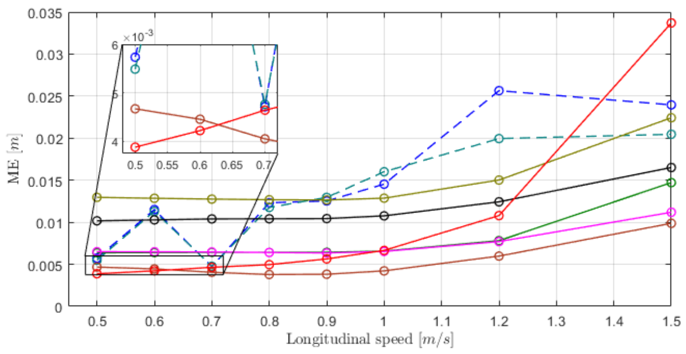
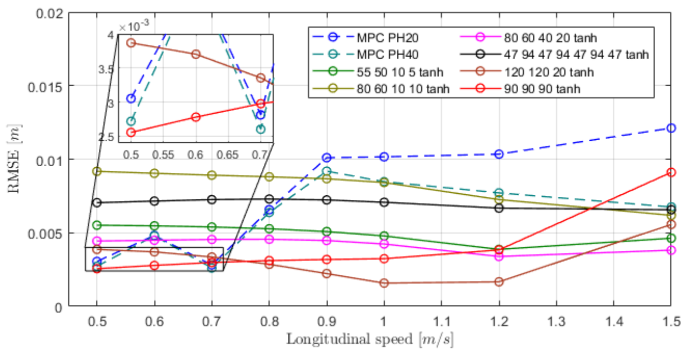


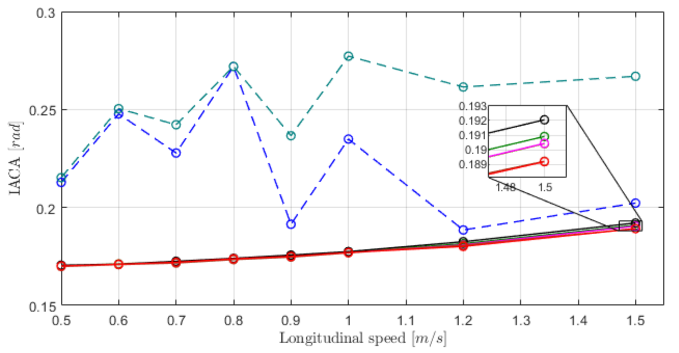
Figure 6.6: Sensitivity analysis conducted over hidden layer shapes. Similar to the approach used in analysing validation losses and elapsed time in Figure 6.5, the depth (i.e., number of layers) and overall number of neurons (i.e., sum of the widths) are kept constant. As observed in Figures 6.6c and 6.6b, pyramidal hidden layer shapes (i.e., when the first hidden layers are wider than subsequent ones) tend to yield superior KPIs when the slope is gradual. However, overall, maintaining a constant width appears to be a reliable approach, consistently delivering good results.



(a) Maximum lateral error.



(b) Root mean square lateral error.



(c) Integral of absolute action control.

Figure 6.7: A final comparison between MPCs and MLPs in simulation reveals expected variations due to the influence of stochastic events during training, such as initialisation and biases. However, it's noteworthy that some MLPs not only surpassed the KPIs of MPCs with shortened prediction horizons but also outperformed expertly tuned MPCs. This can be attributed to the higher operational frequency of the Simulink block compared to the initial MPC (i.e., 100Hz). Consequently, MLPs, despite being approximations of MPCs, can process more inputs in less time, leading to enhanced performance.

Chapter 7

Experimental Results

The simulations were conducted within a Simulink-based environment, where the QCar model was interfaced with the Simulink workstation. Detailed information regarding the communication protocol used has been omitted as it falls outside the scope of this study. Below is a schematic representation of the Simulink model utilised for experimental validations (see Figure 7.1). Notable differences include the integration of the real plant and the implementation of a Kalman filter. This filter is necessary as the signals now originate from sensors, thereby being susceptible to instrumental noise and inaccuracies.

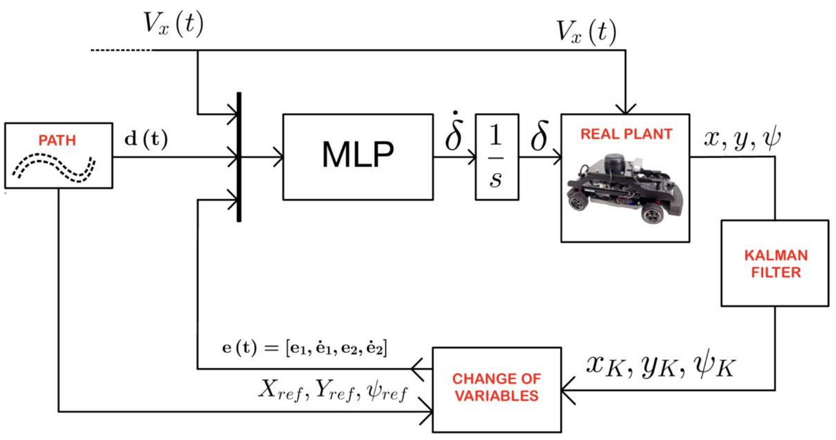
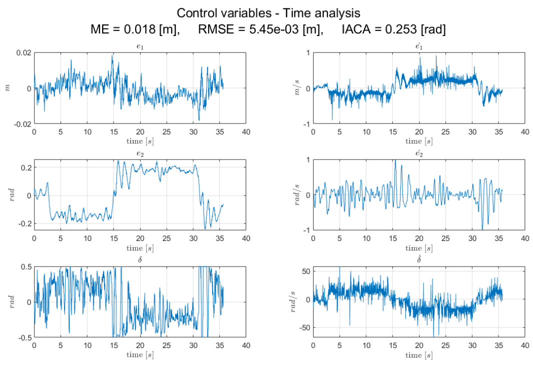
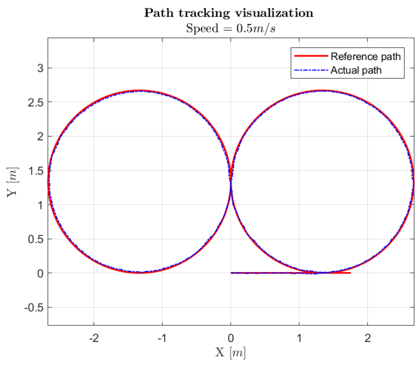


Figure 7.1: Schematic representation of the Simulink model for experimental validations. Notable differences include the inclusion of the real plant and the utilisation of a Kalman filter, necessitated by the fact that the signals are sourced from sensors, making them vulnerable to instrumental noise and inaccuracies.

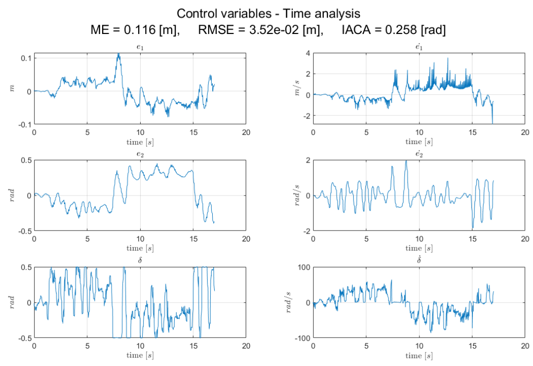


(a) Control variables over time

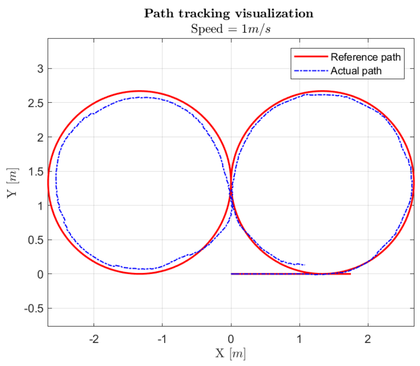


(b) xy view

Figure 7.2: Experimental Results for MLP with architecture (120-120-20) and tanh activation function at 0.5 m/s. At low speeds the errors are still acceptable.

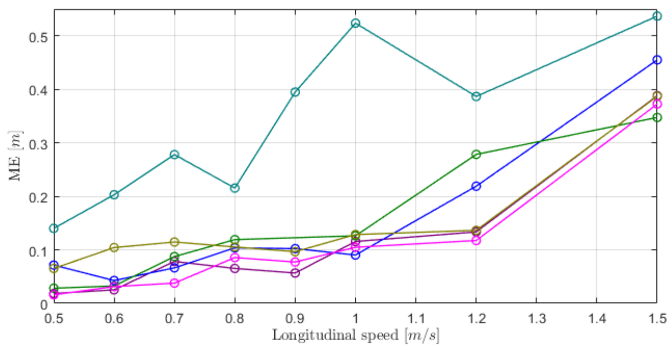


(a) Control variables over time

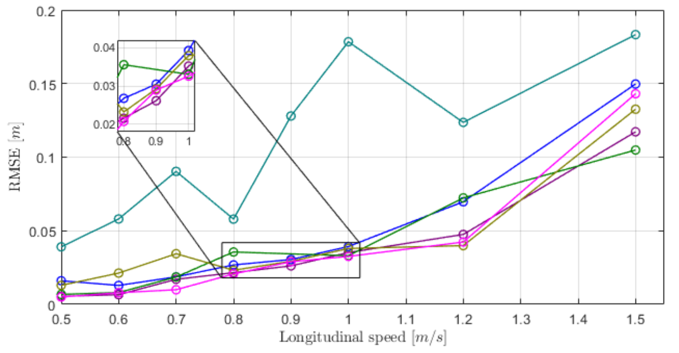


(b) xy view

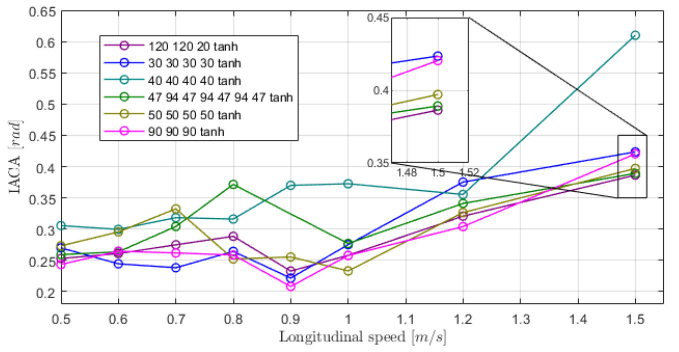
Figure 7.3: Experimental Results for MLP with architecture (120-120-20) and tanh activation function at 1.0 m/s. As the speed increases, the errors increase significantly compared to those in the simulations. Additionally, the KPIs are higher than those in the MPCs. This discrepancy may be attributed to the dataset's lack of representativeness for real-world MLP deployment, originating from simulations.



(a) Maximum lateral error.



(b) Root mean square lateral error.



(c) Integral of absolute action control.

Figure 7.4: Experimental KPIs of some MLPs. Notice how much the charge of the QCar affected the performance of [40 40 40 tanh].

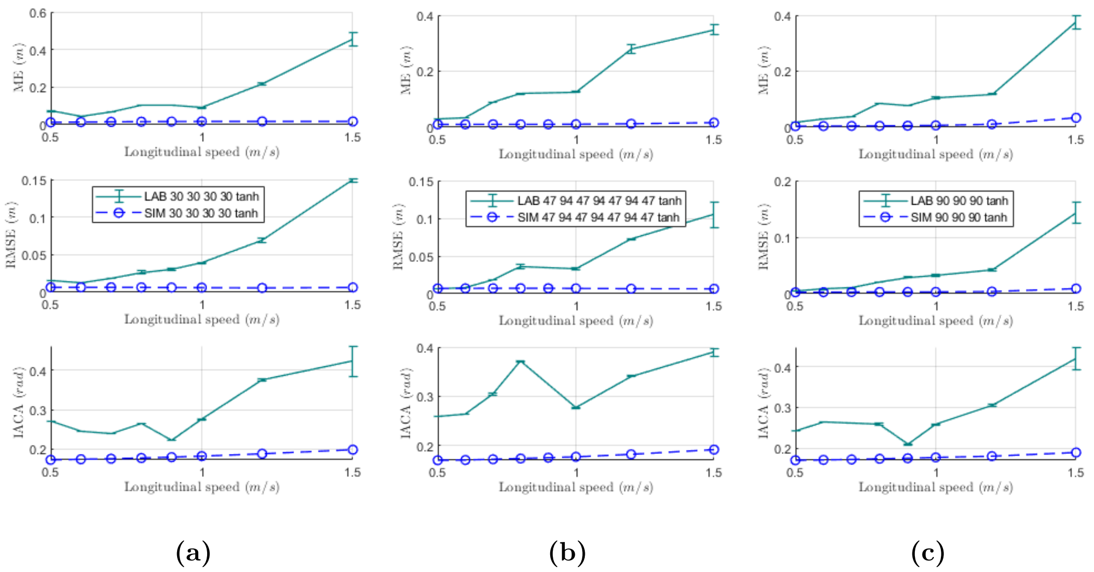


Figure 7.5: Comparison of Experimental and Simulated MLPs. Noticeably, the experiment resulted in significantly lower KPIs. However, as shown in Figure 7.6, this discrepancy can be attributed to inaccuracies in the internal model.

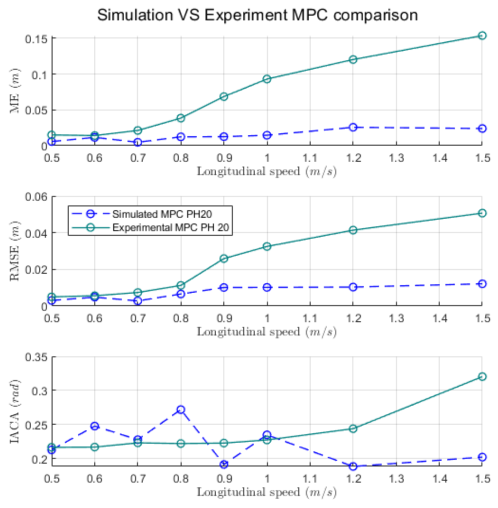


Figure 7.6: Comparison of Experimental and Simulated benchmark MPC. Substantial differences between simulation and experiment highlight the inaccuracy of the internal model in representing the real plant.

Chapter 8

Conclusions

The conclusion of this thesis can be delineated into two primary aspects:

a. **Novelty Programming Outcomes:**
   This thesis has contributed novel programming tools, including the MATLAB-Python MLP builder and the Model Predictive Control Neural Network Optimised Based (MPC NNBO) toolkit.

b. **Simulation and Experimental Results:**
   Additionally, significant insights have been gleaned from both simulation and experimental endeavours.

8.1 Innovative Programming Results

The MATLAB-Python integration for MLP and the MPC NNBO toolkit constitute significant advancements discussed in Chapter 4.

The MATLAB-Python toolchain introduces a novel method enabling users to leverage MATLAB's intuitive interface while capitalising on Python's computational efficiency for training tasks.

As for the MPC NNBO toolkit, it seamlessly integrates with any Model Predictive Controller (MPC) developed using ACADO, requiring only minimal adjustments.

Both solutions are meticulously crafted with emphasis on user-friendliness and adaptability, empowering researchers and professionals to streamline their processes and achieve heightened efficiency.

Looking ahead, these tools will undergo further refinement. The MATLAB-Python toolchain will be expanded to include additional functionalities such as enhanced dataset preprocessing and support for alternative activation functions. Meanwhile, the MPC NNBO toolkit will broaden its scope to encompass not only path tracking but also other facets of autonomous vehicle control.

Moreover, both the toolkit and toolchain have been shared with the University of Surrey, alongside comprehensive training materials, video tutorials, example applications (e.g., handwritten digit classification, function approximation), and detailed documentation. This dissemination effort constitutes an additional achievement of this project, poised to benefit future research endeavours.

8.2 Simulation and Experimental Results

The objective of this study was to train a MLP using the toolkit mentioned in the preceding section 8.1. Specifically, the goal was to surpass an already implemented Model Predictive Controller (MPC), with a non-implementable MPC serving as the expert controller. In simulation, this objective was achieved, as evidenced by the plot in Figure 7.4, where certain MLPs exhibited KPIs superior not only to the benchmark MPC but also to the expert MPC, attributed to the higher operational frequency allowed by the only-one-evaluation.

However, contrary to expectations, the experimental results showed a significant discrepancy. The controllers' performances in the experimental setup were notably lower compared to simulation, as depicted in Figure 7.5. A similar trend was observed with the benchmark MPC, as shown in Figure 7.6. This suggests that the discrepancy was inherent in the formulation of the internal model, particularly at higher velocities, where it deviated substantially from the real plant in both cases. Thus,the discrepancy observed between the simulation and experimental results was not primarily attributed to the approximation made by the trained models, but rather to the limitations of the expert controller and, subsequently, to the quality and representativeness of the dataset used for training.

In addition to the limitations and drawbacks associated with this methodology, such as lengthy training times, the need for well-chosen expert paths, and the patience required for hyper-parameter tuning, there are significant pitfalls. The primary limitation is the inability of trained models to generalise beyond a certain limit. Consequently, while easier constraints such as output saturation are generally guaranteed when using tanh as the output activation function, harder constraints like limitations over the state of the plant, which are characteristic of the MPC and can vary from one training to another, are not assured. However, it's important to note that this does not necessarily render the method ineffective; rather, it indicates that the approach may be sub-optimal in certain scenarios.

8.3 Prospective Enhancements

In addressing the underwhelming performance observed at higher velocities, three primary avenues are proposed within the simple structure of the MLP:

1. **Integration of a Realistic Internal Model:** One approach involves incorporating a more realistic internal model, possibly utilising an ANN to emulate it. This strategy aims to enhance the model's ability to adapt to varying conditions and improve performance.

2. **Implementation of Online Adaptation:** Another strategy entails developing an online adaptation mechanism to adjust weights post-pretraining. This approach enables the model to continuously learn and refine its performance in real-time, thereby potentially mitigating performance shortcomings encountered at higher velocities.

3. **Inclusion of Weights as Input during Training:** Alternatively, the MLP could be trained while also considering the weights as additional input parameters. This method allows for post-training tuning of the weights, albeit at the expense of simplicity. However, this option may offer less elegance compared to the other approaches.

Of these options, the integration of a more realistic internal model or the development of an online adaptation mechanism presents more intriguing possibilities. A combination of these approaches could also be explored for a comprehensive solution.

Appendix A

MLP algorithms

Algorithm 2 MLP training

```
1: **Input:** Training data $(X, Y)$, learning rate $\alpha$, batch size $\mathbf{B}$, momentum decay coefficient $\beta_1$, sum square $\beta_2$
2: Normalise input features in $X$ and shuffle rows in $(X, Y)$
3: Initialise weights and biases randomly
4: **Forward Pass:**
5: **for** each training example $(x, y)$ in $(X, Y)$ **do**
6:     Compute the predicted output $\hat{y}$ using forward propagation
7:     Compute the loss $L$ using a suitable loss function (e.g., mean squared error)
8: **end for**
9: **Backward Pass:**
10: **for** each training example $(x, y)$ in $(X, Y)$ **do**
11:     Compute the gradient of the loss with respect to the output layer: $\delta_{out} = \frac{\partial L}{\partial \hat{y}} \odot f'(\text{output})$
12:     Compute the gradient of the loss with respect to the hidden layer: $\delta_{hidden} = \delta_{out} \cdot W_{out}^T \odot f'(\text{hidden})$
13:     Update output layer weights: $W_{out} = W_{out} - \alpha \cdot \text{output}^T \cdot \delta_{out}$
14:     Update output layer biases: $b_{out} = b_{out} - \alpha \cdot \text{sum}(\delta_{out})$
15:     Update hidden layer weights: $W_{hidden} = W_{hidden} - \alpha \cdot x^T \cdot \delta_{hidden}$
16:     Update hidden layer biases: $b_{hidden} = b_{hidden} - \alpha \cdot \text{sum}(\delta_{hidden})$
17: **end for**
```

Algorithm 3 Adam optimizer algorithm. All operations are element-wise, even powers. Good values for the constants are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. ϵ is needed to guarantee numerical stability.

```
1: procedure ADAM($\alpha, \beta_1, \beta_2, f, \theta_0$)
2:     ▷ $\alpha$ is the stepsize
3:     ▷ $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for the moment estimates
4:     ▷ $f(\theta)$ is the objective function to optimize
5:     ▷ $\theta_0$ is the initial vector of parameters which will be optimized
6:     ▷ Initialization
7:     $m_0 \leftarrow 0$                              ▷ First moment estimate vector set to 0
8:     $v_0 \leftarrow 0$                              ▷ Second moment estimate vector set to 0
9:     $t \leftarrow 0$                                ▷ Timestep set to 0
10:    ▷ Execution
11:    while $\theta_t$ not converged do
12:        $t \leftarrow t + 1$                        ▷ Update timestep
13:        ▷ Gradients are computed w.r.t the parameters to optimize
14:        ▷ using the value of the objective function
15:        ▷ at the previous timestep
16:        $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
17:        ▷ Update of first-moment and second-moment estimates using
18:        ▷ previous value and new gradients, biased
19:        $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
20:        $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
21:        ▷ Bias-correction of estimates
22:        $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$
23:        $\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t}$
24:        $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$        ▷ Update parameters
25:    end while
26:    return $\theta_t$                               ▷ Optimised parameters are returned
27: end procedure
```

Bibliography

[1] jemdoc+MathJax. *ACADO Toolkit*. Accessed on Feb 19th, 2024. 2014. URL: `https://acado.github.io/` (cit. on p. xv).

[2] Roger Grosse. «Lecture 5: Multilayer Perceptrons». In: *inf. téc* (2019) (cit. on p. xv).

[3] IBM. *Overfitting*. Accessed 2024. URL: `https://www.ibm.com/topics/overfitting` (cit. on p. xvi).

[4] IBM. *Underfitting*. Accessed 2024. URL: `https://www.ibm.com/topics/underfitting` (cit. on p. xvi).

[5] Shoaib Azam, Farzeen Munir, Muhammad Aasim Rafique, Ahmad Muqeem Sheri, Muhammad Ishfaq Hussain, and Moongu Jeon. «N2C: Neural Network Controller Design Using Behavioral Cloning». In: *IEEE Transactions on Intelligent Transportation Systems* 22.7 (2021), pp. 4744–4756. DOI: `10.1109/TITS.2020.3045096` (cit. on p. 3).

[6] SAE International. *SAE J3016 - Visual Chart*. Accesso al PDF. 2021. URL: `https://www.sae.org/binaries/content/assets/cm/content/blog/sae-j3016-visual-chart_5.3.21.pdf` (cit. on p. 3).

[7] Jiahao Huang, Steffen Junginger, Hui Liu, and Kerstin Thurow. «Indoor Positioning Systems of Mobile Robots: A Review». In: *Robotics* 12.2 (2023). ISSN: 2218-6581. DOI: `10.3390/robotics12020047`. URL: `https://www.mdpi.com/2218-6581/12/2/47` (cit. on p. 4).

[8] Liam Lynch, Thomas Newe, John Clifford, Joseph Coleman, Joseph Walsh, and Daniel Toal. «Automated Ground Vehicle (AGV) and Sensor Technologies-A Review». In: *2018 12th International Conference on Sensing Technology (ICST)*. 2018, pp. 347–352. DOI: `10.1109/ICSensT.2018.8603640` (cit. on p. 4).

[9] Astrid Rupp, Markus Tranninger, Raffael Wallner, Jasmina Zubača, Martin Steinberger, and Martin Horn. «Fast and Low-Cost Testing of Advanced Driver Assistance Systems using Small-Scale Vehicles». In: *IFAC-PapersOnLine* 52.5 (2019). 9th IFAC Symposium on Advances in Automotive Control AAC 2019, pp. 34–39. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2019.09.006. URL: https://www.sciencedirect.com/science/article/pii/S2405896319306251 (cit. on p. 4).

[10] Yassine Zein, Mohamad Darwiche, and Ossama Mokhiamar. «GPS tracking system for autonomous vehicles». In: *Alexandria Engineering Journal* 57.4 (2018), pp. 3127–3137. ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2017.12.002. URL: https://www.sciencedirect.com/science/article/pii/S1110016818301091 (cit. on p. 4).

[11] A.M. Ribeiro, M.F. Koyama, A. Moutinho, E.C. de Paiva, and A.R. Fioravanti. «A comprehensive experimental validation of a scaled car-like vehicle: Lateral dynamics identification, stability analysis, and control application». In: *Control Engineering Practice* 116 (2021), p. 104924. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2021.104924. URL: https://www.sciencedirect.com/science/article/pii/S096706612100201X (cit. on p. 4).

[12] Lixing Liu, Xu Wang, Xin Yang, Hongjie Liu, Jianping Li, and Pengfei Wang. «Path planning techniques for mobile robots: Review and prospect». In: *Expert Systems with Applications* 227 (2023), p. 120254. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.120254. URL: https://www.sciencedirect.com/science/article/pii/S095741742300756X (cit. on p. 4).

[13] Juqi Hu, Youmin Zhang, and Subhash Rakheja. «Adaptive Lane Change Trajectory Planning Scheme for Autonomous Vehicles Under Various Road Frictions and Vehicle Speeds». In: *IEEE Transactions on Intelligent Vehicles* 8.2 (2023), pp. 1252–1265. DOI: 10.1109/TIV.2022.3178061 (cit. on p. 4).

[14] Spyros G. Tzafestas. «Mobile Robot Control and Navigation: A Global Overview». In: *Journal of Intelligent & Robotic Systems* (2018) (cit. on p. 4).

[15] Qianwen Li, Zhiwei Chen, and Xiaopeng Li. «A Review of Connected and Automated Vehicle Platoon Merging and Splitting Operations». In: *IEEE Transactions on Intelligent Transportation Systems* 23.12 (2022), pp. 22790–22806. DOI: 10.1109/TITS.2022.3193278 (cit. on p. 4).

[16] S Dixit, Saber Fallah, Umberto Montanaro, M Dianati, A Stevens, F Mccullough, and A Mouzakitis. «Trajectory planning and tracking for autonomous overtaking: State-of-the-art and future prospects». In: *Annual Reviews in Control* 45 (2018), pp. 76–86. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2018.02.001 (cit. on p. 4).

[17] P. Stano, U. Montanaro, D. Tavernini, M. Tufo, G. Fiengo, L. Novella, and A. Sorniotti. «Model predictive path tracking control for automated road vehicles: A review». In: *Annual Reviews in Control* 55 (2023), pp. 194–236. ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2022.11.001. URL: https://www.sciencedirect.com/science/article/pii/S1367578822001377 (cit. on pp. 4, 5).

[18] Mohammad Rokonuzzaman, Navid Mohajer, Saeid Nahavandi, and Shady Mohamed. «Review and performance evaluation of path tracking controllers of autonomous vehicles». In: *IET Intelligent Transport Systems* 15.5 (2021), pp. 646–670. DOI: https://doi.org/10.1049/itr2.12051. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/itr2.12051. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/itr2.12051 (cit. on pp. 4, 5).

[19] Nguyen Hung, Francisco Rego, Joao Quintas, Joao Cruz, Marcelo Jacinto, David Souto, Andre Potes, Luis Sebastiao, and Antonio Pascoal. «A review of path following control strategies for autonomous robotic vehicles: Theory, simulations, and experiments». In: *Journal of Field Robotics* 40.3 (2023), pp. 747–779. DOI: https://doi.org/10.1002/rob.22142. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.22142. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.22142 (cit. on p. 5).

[20] Karimi Pour F, Theilliol D, Puig V, and Cembrano G. «Health-aware control design based on remaining useful life estimation for autonomous racing vehicle.» In: *ISA Trans. 2021 Jul;113:196-209. doi: 10.1016/j.isatra.2020.03.032. Epub 2020 Apr 21. PMID: 32451079.* (2021) (cit. on p. 5).

[21] Alexander Liniger and John Lygeros. «Real-Time Control for Autonomous Racing Based on Viability Theory». In: *IEEE Transactions on Control Systems Technology* 27.2 (2019), pp. 464–478. DOI: 10.1109/TCST.2017.2772903 (cit. on p. 5).

[22] Eugenio Alcalá, Vicenç Puig, Joseba Quevedo, and Ugo Rosolia. «Autonomous racing using Linear Parameter Varying-Model Predictive Control (LPV-MPC)». In: *Control Engineering Practice* 95 (2020), p. 104270. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2019.104270. URL: https://www.sciencedirect.com/science/article/pii/S0967066119302187 (cit. on p. 5).

[23] Subhan Khan, Jose Guivant, and Xuesong Li. «Design and experimental validation of a robust model predictive control for the optimal trajectory tracking of a small-scale autonomous bulldozer». In: *Robotics and Autonomous Systems* 147 (2022), p. 103903. ISSN: 0921-8890. DOI: `https://doi.org/10.1016/j.robot.2021.103903`. URL: `https://www.sciencedirect.com/science/article/pii/S0921889021001883` (cit. on p. 5).

[24] Victor Mazzilli, Stefano De Pinto, Leonardo Pascali, Michele Contrino, Francesco Bottiglione, Giacomo Mantriota, Patrick Gruber, and Aldo Sorniotti. «Integrated chassis control: Classification, analysis and future trends». In: *Annual Reviews in Control* 51 (2021), pp. 172–205. ISSN: 1367-5788. DOI: `https://doi.org/10.1016/j.arcontrol.2021.01.005`. URL: `https://www.sciencedirect.com/science/article/pii/S1367578821000055` (cit. on p. 5).

[25] B. Kosko. «Fuzzy systems as universal approximators». In: *IEEE Transactions on Computers* 43.11 (1994), pp. 1329–1333. DOI: `10.1109/12.324566` (cit. on p. 5).

[26] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. «Multilayer feedforward networks are universal approximators». In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/0893-6080(89)90020-8`. URL: `https://www.sciencedirect.com/science/article/pii/0893608089900208` (cit. on p. 5).

[27] J. Brian Froisy. «Model predictive control: Past, present and future». In: *ISA Transactions* 33.3 (1994), pp. 235–243. ISSN: 0019-0578. DOI: `https://doi.org/10.1016/0019-0578(94)90095-7`. URL: `https://www.sciencedirect.com/science/article/pii/0019057894900957` (cit. on p. 6).

[28] Cahyantari Ekaputri and Arief Syaichu-Rohman. «Model predictive control (MPC) design and implementation using algorithm-3 on board SPARTAN 6 FPGA SP605 evaluation kit». In: *2013 3rd International Conference on Instrumentation Control and Automation (ICA)*. 2013, pp. 115–120. DOI: `10.1109/ICA.2013.6734056` (cit. on p. 8).

[29] Steven Kumar, Aditya Tulsyan, Bhushan Gopaluni, and Philip Loewen. «A Deep Learning Architecture for Predictive Control». In: *IFAC-PapersOnLine* 51 (Jan. 2018), pp. 512–517. DOI: `10.1016/j.ifacol.2018.09.373` (cit. on p. 9).

[30] Angelo D. Bonzanini, Joel A. Paulson, Georgios Makrygiorgos, and Ali Mesbah. «Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks». In: *Computers & Chemical Engineering* 145 (2021), p. 107174. ISSN: 0098-1354. DOI: `https://doi.org/10.1016/j.compchemeng.2020.107174`. URL: `https:`

//www.sciencedirect.com/science/article/pii/S0098135420307110 (cit. on p. 9).

[31] Sergio Lucia, Denis Navarro, Benjamin Karg, Héctor Sarnago, and Óscar Lucía. «Deep Learning-Based Model Predictive Control for Resonant Power Converters». In: *IEEE Transactions on Industrial Informatics* 17 (2018), pp. 409–420. URL: https://api.semanticscholar.org/CorpusID:53631058 (cit. on p. 9).

[32] T. Parisini and R. Zoppoli. «A receding-horizon regulator for nonlinear systems and a neural approximation». In: *Automatica* 31.10 (1995), pp. 1443–1451. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(95)00044-W. URL: https://www.sciencedirect.com/science/article/pii/000510989500044W (cit. on p. 9).

[33] A.R. Barron. «Universal approximation bounds for superpositions of a sigmoidal function». In: *IEEE Transactions on Information Theory* 39.3 (1993), pp. 930–945. DOI: 10.1109/18.256500 (cit. on p. 9).

[34] Bernt M. Åkesson, Hannu T. Toivonen, Jonas B. Waller, and Rasmus H. Nyström. «Neural network approximation of a nonlinear model predictive controller applied to a pH neutralization process». English. In: *Computers and Chemical Engineering* 29.2 (2005), pp. 323–335. DOI: 10.1016/j.compchemeng.2004.09.023 (cit. on p. 10).

[35] M. S. Ahmed and M. A. Al-Dajani. «Neural regulator design». In: *Neural Networks* 11.9 (1998), pp. 1695–1709. DOI: 10.1016/s0893-6080(98)00097-5. URL: https://www.sciencedirect.com/science/article/pii/S0893608098000975 (cit. on p. 10).

[36] *MathWorks Automotive Advisory Board (MAAB) Guidelines.* https://es.mathworks.com/solutions/mab-guidelines.html. Accessed: March 14, 2024 (cit. on pp. 11, 21).

[37] Alex Lenail. *NN-SVG: Neural Network SVG Drawer.* https://alexlenail.me/NN-SVG/LeNet.html. Accessed on March 20, 2024. Year of Access (cit. on pp. 33, 34).

[38] M.W Gardner and S.R Dorling. «Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences». In: *Atmospheric Environment* 32.14 (1998), pp. 2627–2636. ISSN: 1352-2310. DOI: https://doi.org/10.1016/S1352-2310(97)00447-0. URL: https://www.sciencedirect.com/science/article/pii/S1352231097004470 (cit. on p. 44).