POLITECNICO DI TORINO
Repository ISTITUZIONALE

Nonlinear model predictive control via behavioral cloning: Framework, design and simulation assessment

Original
Nonlinear model predictive control via behavioral cloning: Framework, design and simulation assessment / Sun, H., Caponio, C., Costantini, M.M., Mihalkov, M., Tavernini, D., Gruber, P., Vigliani, A., Sorniotti, A., Yin, G., Montanaro, U.. - In: CHINESE JOURNAL OF MECHANICAL ENGINEERING. - ISSN 1000-9345. - 39:(2026). [10.1016/j.cjme.2026.100421]

Availability:
This version is available at: 11583/3018150 since: 2026-11-19T09:45:07Z

Publisher:
Elsevier

Published
DOI:10.1016/j.cjme.2026.100421

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

27 November 2026

Chinese Journal of Mechanical Engineering 39 (2026) 100421

Contents lists available at ScienceDirect

Chinese Journal of Mechanical Engineering

journal homepage: www.keaipublishing.com/en/journals/cjme/



Nonlinear model predictive control via behavioral cloning: Framework, design and simulation assessment

Hao Sun[a], Carmine Caponio[a], Marino Massimo Costantini[b], Mario Mihalkov[a], Davide Tavernini[a], Patrick Gruber[a], Alessandro Vigliani[b], Aldo Sorniotti[b], Guodong Yin[c], Umberto Montanaro[a,*]

[a] School of Engineering, University of Surrey, Guildford GU2 7XH, UK
[b] Department of Mechanical and Aerospace Engineering, Polytechnic University of Turin, Turin 10129, Italy
[c] School of Mechanical Engineering, Southeast University, Nanjing 211189, China

ARTICLE INFO

Keywords:
Imitation learning
Nonlinear model predictive control
Vehicle control

ABSTRACT

Over the past decades, there has been significant interest from industries and academics in model predictive control (MPC) due to its capability to systematically handle constraints and nonlinearities during the control process. However, the computational complexity limits real-time deployment, particularly in embedded systems. This paper presents a framework for imitation learning via behavioral cloning to approximate nonlinear MPC (NMPC) policies with significantly reduced computational cost while maintaining expert-level performance. The proposed framework trains a multi-layer perceptron neural network (MLP) to replicate NMPC-based control behavior, effectively reducing approximation errors introduced during training. The resulting framework is then applied to two applications: i) An NMPC-based path following controller (PF) for a scaled robotic vehicle; and ii) An NMPC-based traction controller (TC) for a rear-wheel-drive electric vehicle driving on an icy surface. Simulation results demonstrate that the trained MLP successfully replicates NMPC performance, offering a computationally efficient alternative for real-time applications.

1. Introduction

Model predictive control (MPC) is an optimization-based approach for controlling dynamical systems. Thanks to its capability of handling control and state constraints, MPC strategies have attracted considerable research interest in various applications [1–5], including connected and autonomous vehicles, mobile and swarm robots, and unmanned aerial drones. At each sampling time, MPC-based controllers compute the control action by solving an optimal control problem (OCP) which considers the performance objectives, system dynamics and system constraints [6–8]. While MPC strategies exhibit widespread applicability across different scenarios, the computational complexity of real-time optimization can be a major limitation of nonlinear cases, particularly when a high control sampling frequency is required or when operating on systems with restricted computational resources [9–12]. Linearization of the dynamical system is a straightforward strategy. Some early works [13,14] employ simplified second-order linear models to represent vehicle dynamics. However, owing to the oversimplified form of such models, the resulting control solutions may lack sufficient accuracy. A feedback linearization study [15] proposes an approach that compensates for system nonlinearities to align the control performance with the characteristics of the original nonlinear system, thereby enabling its application to linear DMPC frameworks. In order to address the uncertainties caused by system linearization, Refs. [16,17] propose a lossless convexification strategy that reformulates the traditional nonlinear optimal control problem from the spatial domain into accurate convex forms. However, such methods are effective only for specific problem structures and therefore lack general applicability.

A promising approach to tackle this issue is imitation learning (IL), in which a learning-based strategy is trained to replicate the behavior of the nonlinear MPC (NMPC) policies [18–20]. Imitation learning leverages domain knowledge through expert demonstrations, treating the expert as a black-box oracle capable of providing the desired actions upon request. In Ref. [18], the NMPC is employed to achieve maximum collision avoidance ability by integrated steering and braking, then

Peer review under the responsibility of editorial board of Chinese Journal of Mechanical Engineering.
* Corresponding author.
E-mail address: u.montanaro@surrey.ac.uk (U. Montanaro).

https://doi.org/10.1016/j.cjme.2026.100421
Received 17 October 2025; Received in revised form 26 October 2026; Accepted 26 October 2026
Available online 4 November 2026
1000-9345/© 2026 The Authors. Publishing services by Elsevier B.V. on behalf of KeAi Communications Co. Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

imitated by a neural network (NN) to satisfy real-time requirements. CarSim-based human-vehicle interactive simulation experiments demonstrate that the proposed IL-based control structure has no issue applying the offline-trained DNN in the simulation while showing robustness to parameter changes. In Ref. [19], motion planning algorithms for industrial manipulators in the presence of human operators based on NN are proposed, aiming to imitate the behavior of the NMPC scheme. The proposed NN solutions retain the safety features of NMPC in terms of speed and separation monitoring. Moreover, they improve the robot's performance by reducing both task completion time and the NMPC cost, with the latter evaluated a posteriori using experimental data. By leveraging the NMPC expert for the generation of labeled data, the system can learn a control policy that achieves similar closed-loop performance compared with that provided by the expert.

In this paper, we propose a framework to train a multi-layer perceptron (MLP) that mimics an expert NMPC controller while minimizing approximation errors. The trained MLP achieves control performance comparable to that of the expert NMPC controller with significantly lower computational cost, making it suitable for real-time implementation. The key objective of this process is to minimize the approximation error between the trained MLP and the expert NMPC policy, which leads to an approximated control policy and sub-optimal performance with respect to the expert controller. The proposed strategy is applied to:

i) A path following controller (PF);
ii) A traction controller (TC).

First of all, the proposed framework is calibrated and implemented in an NMPC-based path-following controller for a scaled robotic vehicle. In automated vehicle applications, path-following controllers [21–24] are fundamental components of the control architecture. They generate actuator commands, typically in the form of steering angles and braking torques, to track trajectories provided by higher-level modules, such as path planners [25,26]. These controllers also adapt to dynamic and uncertain real-world environments. Among the various strategies, NMPCs are well-suited for autonomous driving tasks in scaled robotic platforms [27]. The simulation analysis is carried out considering a vehicle model experimentally validated along a variety of both longitudinal and lateral maneuvers [28], and implemented in MATLAB/Simulink [29] to demonstrate its potential applicability. After confirming the feasible operation of the proposed method in Case Study 1, it is further applied to a traction controller for the on-road vehicle to validate its practical effectiveness. Traction controllers modulate the driver's traction torque request to mitigate wheel slip, thereby improving vehicle stability by maximizing tire force generation [20,30]. The functionality of TC is especially advantageous in low tire-road friction conditions, where even modest torque demands can trigger rapid wheel dynamics that exceed the driver's control, resulting in substantial wheel slip and potentially causing the vehicle to spin. Furthermore, in TC applications, due to the fast dynamics involved, a small sampling time is crucial to guarantee satisfactory closed-loop wheel slip tracking; thus, well-performing but complex NMPC solutions might not be real-time capable.

These methods are still hindered by the high computational cost of solving constrained optimization problems online, particularly when complex vehicle dynamics, such as those of full-size vehicles, are considered. In this paper, it is demonstrated that the NMPC traction controller presented in Ref. [30], which is not suitable for real-time implementation, can be replicated using the proposed framework. The resulting NN is computationally efficient and achieves closed-loop vehicle responses comparable to those of the original NMPC controller. The simulation analysis is conducted using a vehicle model experimentally validated on the icy surface, whose model is implemented in the AVL vehicle simulation model (VSM), a high-fidelity vehicle simulation platform. Overall, the contribution of this paper can be summarized as:

- Aiming at different cases, high-fidelity and experimentally validated nonlinear dynamic models are constructed for the corresponding multi-objective optimization problems. The generated data is systematically labeled and assembled into an expert dataset, which serves as the foundation for subsequent training. By construction, the optimal solutions derived for specific objectives generally demonstrate greater effectiveness.
- Following the last point, an MLP-based training framework is developed to approximate expert NMPC policies with significantly improved computational efficiency, while enabling real-time capability. The generated data are added to the expert dataset during the iterative process, helping to preserve comparable control performance. Refs. [31,32] demonstrate that, with large amounts of accurate training data, even MLP-based architectures with simple structures can achieve strong performance and show potential for deployment and adaptation to different tasks.
- To validate the practical value of the proposed method, two representative vehicle control applications with real-time requirements are presented. Following the outlined methodology, a path-following controller for 1/10-scale Quanser autonomous cars (Qcars) and a traction controller for on-road vehicles are systematically constructed, thereby demonstrating the adaptability of the strategy across different platforms.

The rest of this paper is organized as follows. Section 2 provides a background on NMPC and MLP networks. In Section 3, a comprehensive overview of the functionality of the proposed framework is provided. In Section 4, a sensitivity analysis is conducted to determine the most suitable MLP network topology during the initialization stage. Next, the proposed strategy is applied to scaled QCars to achieve accurate path tracking performance and is subsequently implemented on an on-road vehicle. After calibration, the MLP-based traction controller achieves reliable autonomous driving performance on icy surfaces. Finally, Section 5 draws the main conclusions.

2. Background

This section provides an overview of NMPC and MLP networks that are utilized in the proposed behavioral cloning framework.

2.1. Nonlinear model predictive control

The NMPC is a control strategy for discrete-time systems where at each sampling time, the control action is computed by solving an optimal control problem that minimises a cost function J subject to the system dynamics over a predefined horizon [33]. The NMPC employs a plant model, also referred to as the prediction model. This model is represented as a system of first-order differential equations formulated in state-space form as:

$$\dot{\boldsymbol{X}}_{\mathrm{st}} = f(\boldsymbol{X}_{\mathrm{st}}, \boldsymbol{P}, \boldsymbol{U}), \tag{1}$$

where $\boldsymbol{X}_{\mathrm{st}}$ is the state vector, $\boldsymbol{P}$ is the parameter vector, and $\boldsymbol{U}$ is the control input of the NMPC. The corresponding OCP is defined as:

$$\begin{cases} \min_U J(\boldsymbol{X}_{\mathrm{st}}(0), \boldsymbol{U}(\cdot)) \\ = l_N(\boldsymbol{X}_{\mathrm{st}}(N)) + \sum_{k=0}^{N-1} l(\boldsymbol{X}_{\mathrm{st}}(k), \boldsymbol{U}(k)), \\ \mathbf{s.\ t.} \quad \text{for } k = 0, 1, 2, \ldots, N-1, \\ \boldsymbol{X}_{\mathrm{st}}(k+1) = f_d(\boldsymbol{X}_{\mathrm{st}}(k), \boldsymbol{P}, \boldsymbol{U}(k)), \\ \underline{\boldsymbol{X}}_{\mathrm{st}} \le \boldsymbol{X}_{\mathrm{st}}(k) \le \bar{\boldsymbol{X}}_{\mathrm{st}}, \\ \underline{\boldsymbol{U}} \le \boldsymbol{U}(k) \le \bar{\mathbf{U}}, \\ \boldsymbol{X}_{\mathrm{st}}(0) = \boldsymbol{X}_{\mathrm{in}}, \end{cases} \tag{2}$$

where $\boldsymbol{X}_{in}$ is the initial value of the state vector; $\boldsymbol{U}(\cdot)$ is the control sequence over the discrete horizon $k = 0, 1, \ldots, N-1$; N is the number of steps of the prediction horizon $H_p = NT_s$, with T_s being the

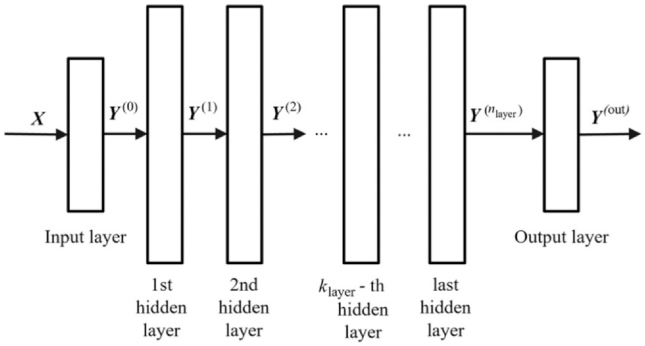


Fig. 1. MLP schematic.

discretization time; k indicates the discretization step along the prediction horizon; $\boldsymbol{X}_{\text{st}}$ and $\bar{\boldsymbol{X}}_{\text{st}}$ are the lower and upper limits for $\boldsymbol{X}_{\text{st}}$; $\underline{\boldsymbol{U}}$ and $\bar{\boldsymbol{U}}$ are the lower and upper limits for $\boldsymbol{U}$; $\boldsymbol{X}_{\text{st}}(k+1) = f_d(\boldsymbol{X}_{\text{st}}(k), \boldsymbol{P}, \boldsymbol{U}(k))$ is the discretized version of the model in Eq. (1); $l(\boldsymbol{X}_{\text{st}}(k), \boldsymbol{U}(k))$ is the stage cost function associated with each time step, defined as a least-squares function; and $l_N(\boldsymbol{X}_{\text{st}}(N))$ is the terminal cost, also called the Mayer term, which is used to make the closed-loop system converge towards the desired dynamics and enhance stability. For $k = 0$, the initial state and parameters are measured/estimated, while for $k > 0$ the evolution of the state variable is predicted through the prediction model.

2.2. Multi-layer perceptron network

Multi-layer perceptron neural networks are a class of feedforward neural network (FFNN), characterized by the unidirectional propagation of information [34]. As shown in Fig. 1, data flow sequentially from the input layer through one or more hidden layers, and reach the output layer. No feedback between layers and nodes is considered.

The algorithm adopted for the training process of MLP weights and biases is the backpropagation algorithm, which targets the minimization of a loss function E. Hence, the algorithm [35] implemented in the proposed framework is summarized as follows. By denoting $n_n \in \mathbb{N}$ the number of inputs, and $n_{\text{layer}} \in \mathbb{N}$ and $n_{\text{out}} \in \mathbb{N}$ the number of hidden layers and outputs, respectively, an MLP network can be expressed as

$$\begin{cases} \boldsymbol{Y}^{(0)} = \boldsymbol{X}, \\ \boldsymbol{Z}^{(k_{\text{layer}})} = \boldsymbol{W}^{(k_{\text{layer}})} \cdot \boldsymbol{Y}^{(k_{\text{layer}}-1)} + \boldsymbol{B}^{(k_{\text{layer}})}, \\ \boldsymbol{Y}^{(k_{\text{layer}})} = \varepsilon^{(k_{\text{layer}})}(\boldsymbol{Z}^{(k_{\text{layer}})}), \\ \boldsymbol{Z}^{(\text{out})} = \boldsymbol{W}^{(\text{out})} \cdot \boldsymbol{Y}^{(n_{\text{layer}})} + \boldsymbol{B}^{(\text{out})}, \\ \boldsymbol{Y}^{(\text{out})} = \varepsilon^{(\text{out})}(\boldsymbol{Z}^{(\text{out})}), \end{cases} \tag{3}$$

where $\boldsymbol{X} = [x_1, x_2, \ldots, x_{n_{\text{in}}}]$ denotes the input vector of the neural network; $\boldsymbol{Y}^{(0)}$ is the input layer, which generally processes the input vector without modifying the numerical values [34]; $\boldsymbol{Y}^{(k_{\text{layer}})}$ represents the output of the k_{layer}-th hidden layer, where $k_{\text{layer}} \in [1, n_{\text{layer}}]$. $\boldsymbol{W}^{(k_{\text{layer}})}$ and $\boldsymbol{B}^{(k_{\text{layer}})}$ denote the weight matrices and bias vectors of the k_{layer}-th hidden layer. $\varepsilon^{(k_{\text{layer}})}$ is the activation function of the k_{layer}-th hidden layer, which processes the weighted sum of the outputs of the previous layer $\boldsymbol{Z}^{(k_{\text{layer}})}$. The output of the entire MLP ($\boldsymbol{Y}^{(\text{out})}$) is determined by the corresponding weight matrix $\boldsymbol{W}^{(\text{out})}$, bias vector $\boldsymbol{B}^{(\text{out})}$ and activation function $\varepsilon^{(\text{out})}$, which are used to process the weighted sum of the final hidden layer, denoted as $\boldsymbol{Z}^{(\text{out})}$. The weight parameters and biases are learned through the training process. Moreover, the training is based on the dataset where each data point includes an observation-action pair that requires approximation. In the context of imitation learning, training involves identifying the optimal set of weights and biases minimizing the error (or loss) between the action computed by the network itself ($\boldsymbol{Y}^{(\text{out})}$) and that generated by the expert NMPC ($\boldsymbol{Y}_{\text{train}}$).

$$\begin{cases} \dfrac{\partial E}{\partial \boldsymbol{W}^{(\text{out})}} = \left[\boldsymbol{Y}^{(n_{\text{layer}})T}\right] \cdot \dfrac{\partial E}{\partial \boldsymbol{Y}} \times [\varepsilon^{(\text{out})'}(\boldsymbol{Z}^{(\text{out})})], \\ \dfrac{\partial E}{\partial \boldsymbol{B}^{(\text{out})}} = \text{rowsum}\left(\dfrac{\partial E}{\partial \boldsymbol{Y}} \times [\varepsilon^{(\text{out})'}(\boldsymbol{Z}^{(\text{out})})]\right), \end{cases} \tag{4}$$

$$\begin{cases} \dfrac{\partial E}{\partial \boldsymbol{W}^{(k_{\text{layer}})}} = [\boldsymbol{Z}^{(k_{\text{layer}}-1)T}] \cdot \dfrac{\partial E}{\partial \boldsymbol{Y}^{(k_{\text{layer}}+1)}} \\ \boldsymbol{W}^{(k_{\text{layer}}+1)T} \times [\varepsilon^{(k_{\text{layer}})'}(\boldsymbol{Z}^{(k_{\text{layer}})})], \\ \dfrac{\partial E}{\partial \boldsymbol{B}^{(k_{\text{layer}})}} = \text{rowsum} \\ \left(\dfrac{\partial E}{\partial \boldsymbol{Y}^{(k_{\text{layer}}+1)}} \cdot \boldsymbol{W}^{(k_{\text{layer}}+1)T} \times [\varepsilon^{(k_{\text{layer}})'}(\boldsymbol{Z}^{(k_{\text{layer}})})]\right), \end{cases} \tag{5}$$

where the rowsum(v)-function provides the sum of the vector v, $\varepsilon^{(\cdot)'}$ is the first derivative of the activation function $\varepsilon^{(\cdot)}$, while $[\frac{\partial E}{\partial \boldsymbol{W}^{(\text{out})}}, \frac{\partial E}{\partial \boldsymbol{B}^{(\text{out})}}, \frac{\partial E}{\partial \boldsymbol{W}^{(k_{\text{layer}})}}, \frac{\partial E}{\partial \boldsymbol{B}^{(k_{\text{layer}})}}]$ are the gradients used to update the weights and biases of the network as:

$$\begin{cases} \boldsymbol{W}_{t+1}^{(\text{out})} \leftarrow \boldsymbol{W}_t^{(\text{out})} - \alpha \dfrac{\partial E}{\partial \boldsymbol{W}^{(\text{out})}}, \\ \boldsymbol{B}_{t+1}^{(\text{out})} \leftarrow \boldsymbol{B}_t^{(\text{out})} - \alpha \dfrac{\partial E}{\partial \boldsymbol{B}^{(\text{out})}}, \\ \boldsymbol{W}_{t+1}^{(k_{\text{layer}})} \leftarrow \boldsymbol{W}_t^{(k_{\text{layer}})} - \alpha \dfrac{\partial E}{\partial \boldsymbol{W}^{(k_{\text{layer}})}}, \\ \boldsymbol{B}_{t+1}^{(k_{\text{layer}})} \leftarrow \boldsymbol{B}_t^{(k_{\text{layer}})} - \alpha \dfrac{\partial E}{\partial \boldsymbol{B}^{(k_{\text{layer}})}}, \end{cases} \tag{6}$$

where $0 < \alpha \leq 1$ represents the learning rate, which controls the step size of the weight and bias updates along the negative gradient of the loss function. The subscript 't' indicates the epoch at which the parameter is updated. Fundamentally, gradients quantify the impact of weight and bias adjustments on the overall loss function. When the error reaches zero, no further parameter updates occur. Conversely, larger errors necessitate updates to the model parameters. In this context, the learning rate plays a crucial role as it determines the rate at which the model adapts to the data. A higher learning rate facilitates faster convergence but may lead to instability or overshooting of the optimal solution. In contrast, an excessively low learning rate may slow convergence or hinder the optimization process, thereby reducing training efficiency [36].

3. Training framework

The proposed framework for creating NMPC clones consists of a set of functions developed in MATLAB and Python, along with the required interfaces among these programming environments. The related information flow is shown in Fig. 2.

For the end user, the proposed training tool is a MATLAB function, 'MLP_builder' ('train', 'MLP_struct'), where 'MLP_struct' is a data structure including the input and the training data, i.e., the input/output of the expert NMPC (see also Stage 1 in Fig. 2), along with MLP hyper-parameters, e.g., data shuffling preferences, layer sizes, hidden layer architecture, and activation functions to use for each layer (see also the type of data in Table 1 for some of the fields of this data structure). The output of this function is the clone of the expert NMPC as an MLP, i.e., its weights and biases $\boldsymbol{W}^{(k_{\text{layer}})}$, $\boldsymbol{B}^{(k_{\text{layer}})}$, $\boldsymbol{W}^{(\text{out})}$, $\boldsymbol{B}^{(\text{out})}$ (see also stage 3 of Fig. 2), which are added to the data input structure. The training data are provided as a.CSV file where each row is a set of labeled data with the first n_{in} columns being the NMPC inputs while the remaining are the NMPC outputs. The end-user function interfaces with a Python script through the 'pyrunfile' command. The Python script is responsible for constructing and training the MLP according to the user settings (see also Stage 2 of Fig. 2) through the use of a custom-made set of routines implementing the backpropagation algorithm in Eqs. (4)–(6). Once the training process in Python is completed, the weights

and biases, scaling factors used for data normalization, and training loss records are returned to the function. Some of the fields of the data structure 'MLP_struct' and functions included in the training framework are also reported in Table 1.

A schematic of the information flow from the definition of the training problem via the MLP_struct structure to the output NN and the programming environment used at each stage is shown in Fig. 3. Once the training problem is defined by the user, the communication between MATLAB and Python is established, and the training process is initiated. During training, loss values are displayed in the Windows command prompt, offering real-time feedback on the optimization progress and convergence of training errors. Upon completion of the training, the output parameters are saved and prepared for subsequent use in MATLAB, Simulink, or Python, thus allowing the integration of the trained MLP in these programming environments, which are usually utilized for control purposes. Overall, the proposed Behavioral Cloning approach is summarized in Algorithm 1.

Algorithm 1. NMPC behavioral cloning based on MLP

I. Training and Validation Dataset Creation

II. MLP Model Setup (in MATLAB)

a) Define MLP structure (n_{in}, n_{out}, n_{layers}, activation functions);

b) Number of epochs n_{epochs} (training iterations);

c) Optimizer settings (learning rate);

d) Initialize MLP (define an initial set of weights and biases), set the **best accuracy** $L_{best} = 0$;

III. MATLAB → Python Interface

IV. Training Start

1: **for** $1 \le n \le n_{epochs}$ **do**

2: Feed the MLP with the training inputs and calculate the MLP output;

3: Compare MLP output with reference output from dataset, calculate prediction error e_{train};

4: Update the MLP network with e_{train};

5: Feed the new MLP with the validation input and calculate the output;

6: Compare MLP output with reference output from dataset, calculate validation accuracy L_{val};

7: **if** $L_{val} > L_{best}$ **then**

8: Save MLP hyperparameters (weights and biases), $L_{best} = L_{val}$;

9: **end if**

10:Display training and validation accuracies;

11:**end for**

V. Python → MATLAB **Interface**

VI. Training is finished

4. Simulation

4.1. Simulation setup

In this section, the proposed MLP-based NMPC behavioral cloning framework is applied to two representative case studies: i) a path-following controller for scaled Qcars, and ii) a traction controller for on-road vehicles. First, the model and the controller parameters are presented in Table 2.

Then, the topology of the network within the MLP framework is determined through a sensitivity analysis on the number of neurons and hidden layers. In detail, Fig. 4 reports the root mean square error (RMSE) achieved by six different network topologies and the respective

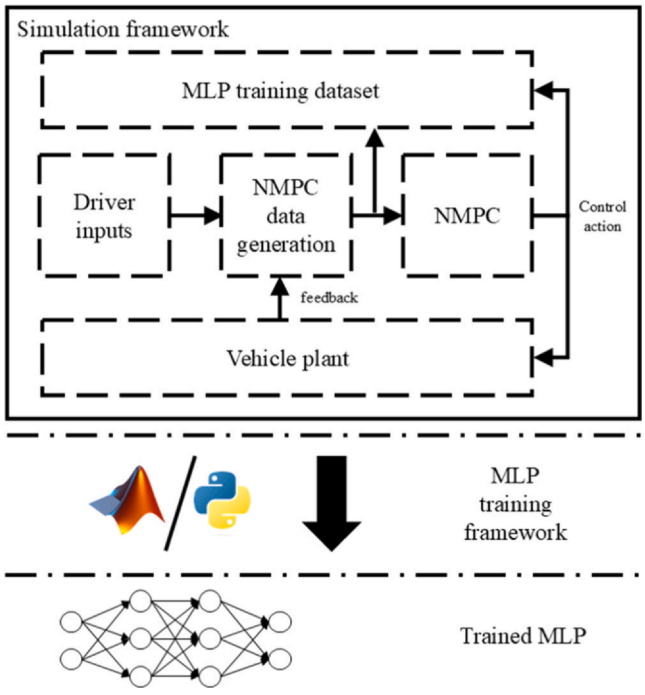


Fig. 2. Imitation learning framework: from training data generation to the trained MLP.

training time. For both cases, the training is conducted on a HP commercial laptop equipped with an AMD Ryzen 5 PRO 230 processor (3.50 GHz), integrated AMD Radeon 760M Graphics, 16 GB RAM, and SSD storage, running Windows 11. The implementation was developed using Python 3.11, with all algorithms implemented through custom code without relying on external third-party libraries. The network architecture is selected based on the sensitivity analysis presented in Fig. 4, where different numbers of hidden layers and neurons are systematically evaluated in terms of both approximation accuracy (RMSE) and training time. As can be observed, the network with a single hidden layer and 10 neurons is characterized by the shortest training time but exhibits the poorest path-following performance. Conversely, increasing the number of hidden layers and/or neurons improves the approximation accuracy and enables the MLP to more closely reproduce the behavior of the expert NMPC, although at the expense of increased training time. These results demonstrate a clear trade-off between model complexity and performance: simpler architectures exhibit shorter training times but larger approximation errors, whereas deeper and wider networks achieve improved accuracy at the cost of higher computational effort during training. Based on this analysis, a network with two hidden layers and 50 neurons per layer was selected, as it provides a suitable balance between approximation accuracy and computational efficiency. In brief, for the PF controller case, the NMPC achieves an average solving time of 5.80 ms, while the corresponding MLP inference time is 1.73 ms, yielding a speedup ratio of 3.35. The RMSE of the steering angle is 11 deg, and the RMSE of the lateral error is 0.017m. For the TC controller, the average NMPC solving time is 16.73 ms and the average MLP inference time is 4.61 ms, corresponding to a speedup ratio of 3.63. In terms of accuracy, at accelerator pedal positions of 60% and 90%, the RMSEs of $_{Tcorr}$ are 19.12 Nm and 36.47 Nm, respectively, while the corresponding RMSEs of $_{kL}$ are 0.35 and 0.48, respectively.

4.2. The path-following controller for scaled robotic QCars

4.2.1. Expert demonstrator formulation

Case Study 1 focuses on a robotic scaled car (Fig. 5), the self-driving QCar platform. This open-architecture system is configured as an all-

Table 1
Functions and the data structure of the training framework.

Name	Type	Description
MLP str = MLP builder	Function	Create a MATLAB structure named MLP struct with the fields required for the training process.
MLP struct.File-csv-name	Data	Specify the path and name of the.csv file containing the dataset used for training.
MLP struct.Epochs	Data	Maximum number of epochs for the training.
MLP struct.HiddenSizes	Data	Number and the size of the hidden layers. This input is a vector of n layer elements, the *i*-th entry is the size of the *i*-th hidden layer
MLP struct.LearningRate	Data	Learning rate.
MLP struct.BatchSize	Data	Batch size.
MLP builder('train', S);	Function	Execute the training problem defined by the variable S, which has the data type MLP struct.
consistency check(S)	Function	Check if the training problem defined by the variable S with the data type MLP struct is consistent, e.g., check the existence of the.cvs file. If the check fails, MLP builder ('train', S) exits with the error code.
MLP training (·)	Function	Execute the training of the NN using the backpropagation algorithm.
Online_Losses	Function	Enable the real-time display of the training losses through the '.cmd' of Windows.
Pyrunfile('.py', ·, settings)	Function	Enable the connection between MATLAB and Python, calling the '.py' script that executes the training. This function outputs the weight and bias matrices of the trained MLP, as well as the scaling factors of the inputs and outputs.
Losses plot	Function	Plot the training and validation losses obtained for each episode.

wheel-drive (AWD) vehicle, powered by a direct current (DC) electric motor, with its rotational speed tracked by an encoder, and operates with a single-transmission-ratio gearbox. Steering actuation is applied by a servo motor, which ensures a maximum steering angle of approximately $\pm 30^{\circ}$. Additionally, it is equipped with an onboard 9-axis inertial measurement unit (IMU) and a light detection and ranging (LiDAR) for comprehensive environmental sensing. The onboard computing system is powered by an NVIDIA Jetson TX2 processor, incorporating a Quad-Core ARM Cortex-A57 microcontroller. For this case study, an experimentally validated simulation model in MATLAB-Simulink is used, encompassing: (i) the longitudinal vehicle dynamics and (ii) the lateral vehicle dynamics, which is based on a dynamic single-track model. The validation results for longitudinal and lateral dynamics are reported in Ref. [28], showing an accurate match between simulations and experiments.

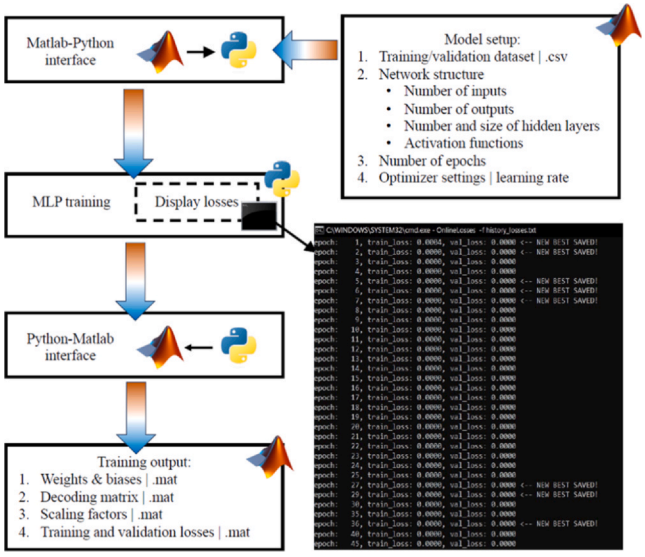


Fig. 3. MATLAB/Python information exchange schematic

Table 2
Parameters for both cases.

Parameter	Path tracking	Traction control
Time step T_s	10 ms	10 ms
Prediction horizon H_p	200 ms	500 ms
Integrator	Runge-Kutta	Runge-Kutta
Optimizer	qpOASES	qpOASES

For the proposed NMPC-PF formulation, the internal model state vector consists of the heading angle error $\Delta\psi$, the lateral error Δy, and their time derivatives (i.e., $\Delta \boldsymbol{e} = [\Delta y, \Delta\dot{y}, \Delta\psi, \Delta\dot{\psi}]$) augmented with the steering angle, thus resulting in $\boldsymbol{x}_{\mathrm{PF}} = [\Delta \boldsymbol{e}, \delta]^{\top}$. The parameter P_{PF} is selected as $P_{\mathrm{PF}} = v_x$, while the control input is the steering angle derivative, i.e., $u_{\mathrm{PF}} = \dot{\delta}$. Selecting $\dot{\delta}$ as the control input allows for weighing the rate of change of the steering angle in the NMPC optimization problem, which limits steering angle oscillations and chattering. The internal model is the lateral dynamics model of the vehicle reported in Ref. [37], written with respect to the desired lane errors:

$$\Delta\dot{\boldsymbol{e}} = \boldsymbol{A}\Delta\boldsymbol{e} + \boldsymbol{B}_1\delta + \boldsymbol{B}_2\rho, \tag{7}$$

where ρ is the reference curvature, while the matrices $\boldsymbol{A}$, $\boldsymbol{B}_1$, $\boldsymbol{B}_2$ are:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{C_{\alpha f}+C_{\alpha r}}{mv_x} & \frac{C_{\alpha f}+C_{\alpha r}}{m} & -\frac{C_{\alpha f}l_f - C_{\alpha r}l_r}{mv_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{C_{\alpha f}l_f - C_{\alpha r}l_r}{I_z v_x} & \frac{C_{\alpha f}l_f - C_{\alpha r}l_r}{I_z} & -\frac{C_{\alpha f}l_f^2 + C_{\alpha r}l_r^2}{I_z v_x} \end{bmatrix}, \quad \boldsymbol{B}_1 = \begin{bmatrix} 0 \\ \frac{C_{\alpha f}}{m} \\ 0 \\ \frac{C_{\alpha f}l_f}{I_z} \end{bmatrix}, \quad \boldsymbol{B}_2 = v_x \begin{bmatrix} 0 \\ -\frac{C_{\alpha f}l_f - C_{\alpha r}l_r}{mv_x} - v_x \\ 0 \\ -\frac{C_{\alpha f}l_f^2 + C_{\alpha r}l_r^2}{I_z v_x} \end{bmatrix}, \tag{8}$$

Eq. (8) is based on the system parameters, defined as the yaw mass moment of inertia $I_z = 0.034$ kg·m^2, the vehicle mass $m = 2.7$ kg, the front and rear semi-wheelbases l_f and l_r, and the front and rear axle cornering stiffness $C_{\alpha f} = 9.60$ N/rad and $C_{\alpha r} = 10.92$ N/rad. While $l_f = 0.128$ m and $l_r = 0.128$ m are known and provided by the robotic vehicle manufacturer, $C_{\alpha f}$ and $C_{\alpha r}$ are identified through the two-stage least-squares method in Ref. [28].

Consequently, the prediction model is:

$$\dot{\boldsymbol{x}}_{\mathrm{PF}} = \boldsymbol{A}_a\boldsymbol{x}_{\mathrm{PF}} + \boldsymbol{B}_{1,a}u_{\mathrm{PF}} + \boldsymbol{B}_{2,a}\rho, \tag{9}$$

where augmented matrices $\boldsymbol{A}_a$, $\boldsymbol{B}_{1,a}$ and $\boldsymbol{B}_{2,a}$ are:

$$A_a = \begin{bmatrix} A & B_1 \\ O_{1,4} & 0 \end{bmatrix}$$
$$B_{1,a} = \begin{bmatrix} O_{4,1} \\ 1 \end{bmatrix}, \quad B_{2,a} = \begin{bmatrix} B_2 \\ 0 \end{bmatrix}, \tag{10}$$

with $\boldsymbol{O}_{n,m}$ being the null element of the vector space $\mathbb{R}^{n,m}$. Similar to the TC, the cost function l_{PF} is defined as the sum of two contributions:

$$l_{\mathrm{PF}} = l_{1,\mathrm{PF}} + l_{2,\mathrm{PF}}. \tag{11}$$

For the controllers using the baseline models, the first Lagrange term $l_{1,\mathrm{PF}}$ minimizes the lateral displacement of the vehicle, as:

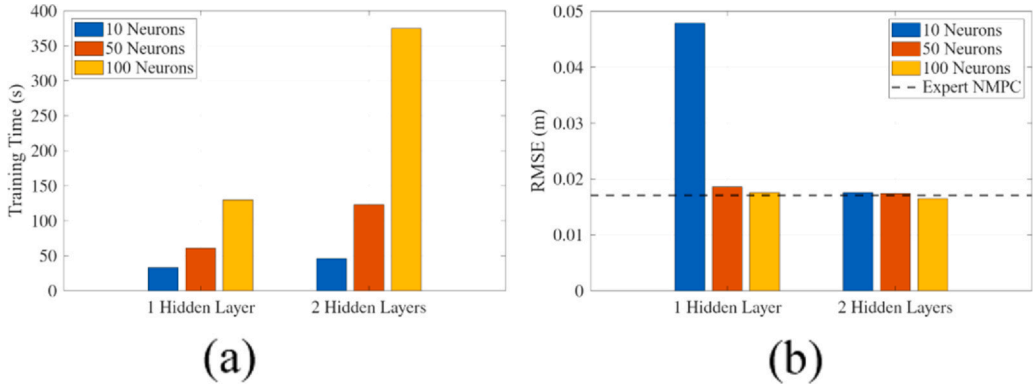


Fig. 4. The sensitivity analysis with respect to the number of neurons: (a) Six network configurations are considered, comprising networks with either one or two hidden layers, where each hidden layer contains 10, 50, or 100 neurons, respectively. (b) The RMSE of the lateral error under different topologies.

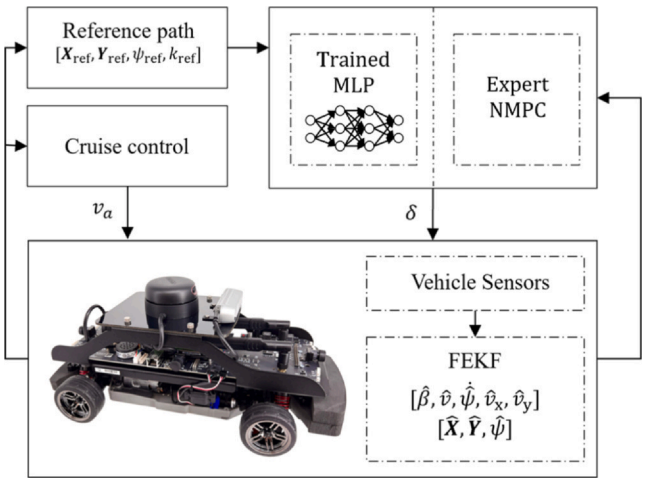


Fig. 5. QCar platform of the University of Surrey.

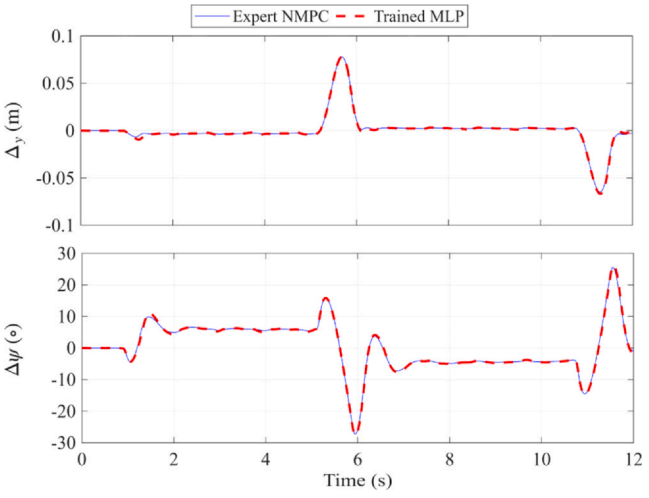


Fig. 6. Comparison of the closed-loop simulation results for the PF: expert NMPC (solid blue line) and MLP (red dashed line).

$$l_{1,\mathrm{PF}} = W_{1,\mathrm{PF}}\{[\Delta y]^2\}, \tag{12}$$

where $W_{1,\mathrm{PF}}$ is a weighting factor. The last Lagrange term, with weight $W_{2,\mathrm{PF}}$, penalizes the control effort:

$$l_{2,\mathrm{PF}} = W_{2,\mathrm{PF}}[\dot{\delta}]^2. \tag{13}$$

4.2.2. Simulations of case study 1

The expert NMPC-PF is implemented with a time step $T_s = 10$ ms and a prediction horizon $H_p = 200$ ms. The cost function weights are tuned empirically to achieve an optimal balance between control action chattering and reference path following performance. Although the proposed configuration ensures good performance in terms of lateral error, it is not real-time capable. To characterize the dynamic behavior of the plant, a constant speed $v = 1.5$ m/s test on a ∞-shape [28] is executed. In this case study, while $\boldsymbol{Y}^{(\mathrm{out})} = \dot{\delta}$, the input vector of the MLP is chosen as $\boldsymbol{X} = \boldsymbol{x}_{\mathrm{PF}}$. The collected dataset is then used to train an MLP network with two hidden layers ($n_{\mathrm{layer}} = 2$) with 50 neurons each. All hidden layers use the Rectified Linear Unit (ReLU) activation function: the output neuron uses a tanh activation, scaled by the maximum steering rate. Fig. 6 presents a comparison between the expert NMPC-PF response (blue solid line) and the corresponding MLP output (red dashed line) in the training scenario. The plot highlights *i).* the lateral error Δy; and *ii).* the heading error $\Delta\psi$. The simulation results demonstrate that the trained MLP network effectively replicates the performance of the expert NMPC-PF, ensuring comparable control behavior and dynamic response also for a path following scenario.

4.3. The traction controller for on-road vehicles

4.3.1. Expert demonstrator formulation

The NMPC-TC formulation is based on Ref. [30] and carried out within the EM-TECH project, and it is reported in this section for completeness. The internal model of the plant, also referred to as the prediction model, is introduced below. For this traction control application, the slip ratio $k_{L/R}$ is defined as:

$$k_{L/R} = \frac{\dot{\theta}_{w,L/R} R_{w,r} - v_x}{\dot{\theta}_{(w,L/R)} R_{(w,r)}} = \frac{k_{v,L/R}}{\dot{\theta}_{(w,L/R)} R_{(w,r)}} \tag{14}$$

where $k_{v,L/R}$ is the slip speed on the driven wheels, $\dot{\theta}_{w,L/R}$ is the angular velocity of the left/right wheel. $R_{w,r} = 0.279$ m is the tire rolling radius and v_x is the longitudinal speed along the centre of the wheel. In order to mitigate $k_{L/R}$, the internal model is built as follows:

$$\frac{d}{dt}k_{v,L/R} = \frac{R_w G_t T_m}{2J_w} - \left[\frac{R_w^2}{J_w} + \frac{1}{m_{eq}}\right] D\mu_{L/R} \sin\left(C \arctan\left(\frac{B}{\mu_{L/R}} \frac{k_{v,L/R}}{\hat{\theta}_{w,L/R} R_{w,r}}\right)\right) F_{z,L/R}, \tag{15}$$

$$\frac{d}{dt}\dot{\theta}_{w,L/R} = \frac{1}{J_w}\left(\frac{G_t T_m}{2} - D\mu_{(L/R)} \sin\left(C \arctan\left(\frac{B}{\mu_{(L/R)}} \frac{k_{(v,L/R)}}{\hat{\theta}_{(w,L/R)} R_{w,r}}\right)\right) F_{(z,L/R)} R_{w,r}\right), \tag{16}$$

$$\frac{d}{dt}T_m = \frac{T_{\mathrm{plant}} - T_m}{\tau_m}, \tag{17}$$



Fig. 7. The research on autonomous driving (ZEBRA) vehicle of the University of Surrey, UK.

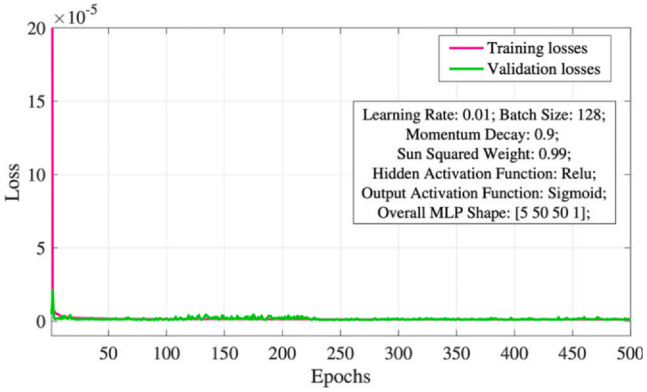


Fig. 8. MLP training and validation losses.

where $G_t = 9.23{:}\ 1$ is the drivetrain gear ratio, $J_w = 1.1$ kg·m^2 is the rear wheel inertia, $m_{eq} = 141.5$ kgis the equivalent mass, $F_{z,L/R} = 1569.2$ N is the normal load, $\mu = 0.085$ is the friction coefficient, and $B = 10$, $C = 1.65$, $D = 1$ are tire Pacejka parameters. T_m denotes the feedback torque received by the electric motor, which is affected by first-order dynamics with a time constant $\tau_m = 10$ ms. $T_{\text{plant}} = T_{\text{rec}} - T_{\text{corr}}$ is the reference torque, where T_{rec} is the torque requested by the driver, and T_{corr} is the control input applied by the NMPC. The state vector, x_{TC}, is:

$$\boldsymbol{x}_{TC} = [k_{v,L},\ k_{v,R},\ \dot{\theta}_{w,L},\ \dot{\theta}_{w,R},\ T_m]^\top. \tag{18}$$

In the specific implementation, the cost function l is defined as the sum of contributions, each representing different aspects of the system performance.

$$l_{\text{TC}} = l_{1,\text{TC}} + l_{2,\text{TC}}. \tag{19}$$

For the controllers using the baseline models, the first Lagrange term $l_{1,TC}$ tracks the reference wheel slip speeds as:

$$l_{1,\text{TC}} = W_{1,\text{TC}}\{[k_{v,L} - k_{\text{vref},L}]^2 + [k_{v,R} - k_{\text{vref},R}]^2\}, \tag{20}$$

where $W_{1,\text{TC}}$ is a weighting factor, and $k_{\text{vref},L}$ and $k_{\text{vref},R}$ represent the left and right reference wheel slip speeds, respectively. These values are computed from the reference slip ratios, $k_{\text{ref},R}$ and $k_{\text{ref},L}$, using the following formulation:

$$k_{\text{vref},L/R} = k_{\text{ref},L/R}\dot{\theta}_{w,L/R}R_{w,r}. \tag{21}$$

The last Lagrange term, with weight $W_{2,\text{TC}}$, penalizes the control effort:

$$l_{2,\text{TC}} = W_{2,\text{TC}}T_{\text{corr}}{}^2. \tag{22}$$

4.3.2. Simulations of case study 2

The simulated vehicle is the ZEBRA prototype, i.e., a modified L7e rear-wheel-drive electric quadricycle, as shown in Fig. 7. It features a centralized on-board electric powertrain, a single-speed transmission, an open differential, half-shafts, and constant velocity joints. The high-fidelity simulation model of the vehicle is implemented in the AVL VSM software and experimentally validated during a dedicated testing session conducted on an ice rink [30].

The expert NMPC-TC is implemented with a time steps $T_s = 10$ ms and a prediction horizon $H_p = 500$ ms. The cost function weights are tuned empirically to achieve an optimal balance between control action

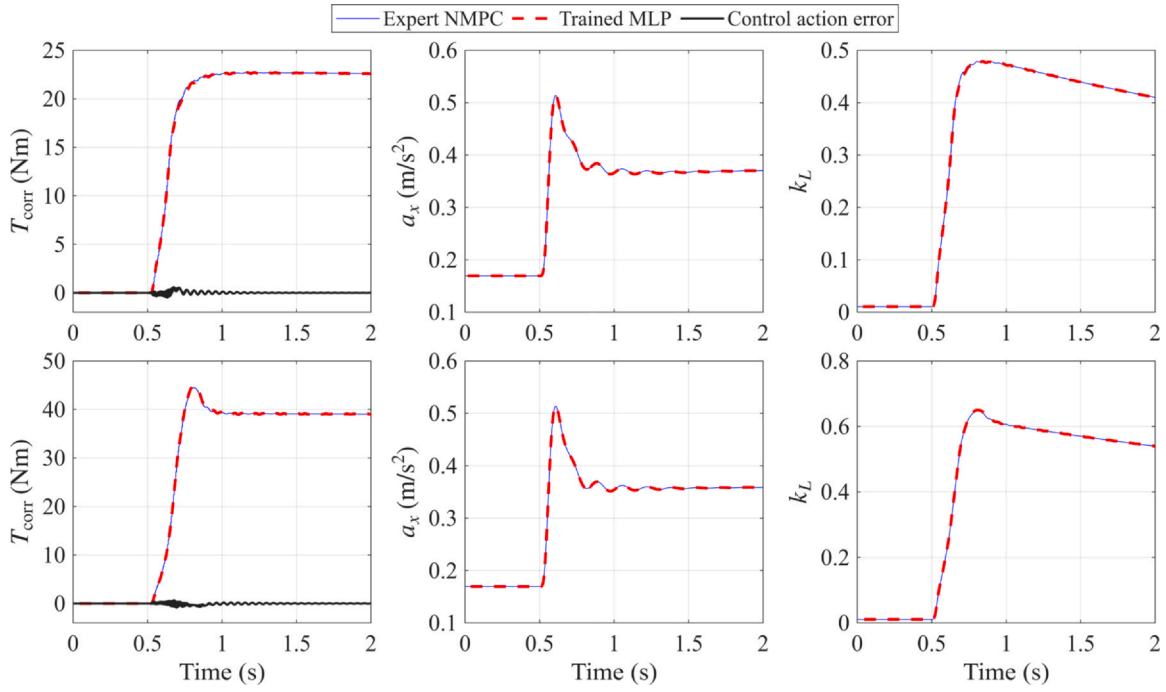


Fig. 9. Comparison of the closed-loop simulation results for the TC: expert NMPC (solid blue line) and MLP (red dashed line).

chattering and slip tracking performance. Although the proposed configuration ensures good slip tracking performance, it is not real-time capable. To analyze the plant response under different torque levels, the controlled plant is subjected to a series of torque step inputs, allowing for a detailed characterization of its dynamic behavior. In this case study, while $\boldsymbol{Y}^{(\text{out})} = T_{\text{corr}}$, the input vector of the MLP is chosen as $\boldsymbol{X} = \boldsymbol{x}_{\text{TC}}$. The collected dataset was then used to train MLP networks with two hidden layers ($n_{\text{layer}} = 2$), consisting of 50 neurons each and a ReLU activation function. The output neuron uses a sigmoid activation function, scaled by the maximum torque output of the electric machine. Fig. 8 presents the output of the framework in terms of real-time training and validation losses displayed through 'cmd' console, and overall losses for 500 epochs of training, highlighting the good fit reached by the proposed framework. Fig. 9 presents a comparison between the expert NMPC response (blue solid line), the corresponding MLP output (red dashed line), and the difference between the two control inputs (black solid line) for two different accelerator pedal positions: 60%, corresponding to the first row of plots in the figure, and 90%, being the second row. The plot highlights *i)* the torque correction T_{corr}, *ii)* the longitudinal acceleration a_x, and *iii)* the left slip ratio k_L. Simulation results demonstrate that the trained MLP network effectively replicates the performance of the expert NMPC, ensuring comparable control behavior and dynamic response under varying driving conditions. The major benefit of the proposed MLP is the computational time, as the resulting MLP network requires much less computational effort.

5. Conclusions

This paper presents a framework to approximate NMPC controllers through NN, targeting the generation of an optimal control policy with reduced computational demands. The proposed framework is validated through *i)*. a path-following algorithm for a scaled robotic car; and *ii)*. A traction control case study on a high-fidelity, experimentally validated vehicle simulation model on the icy road. In both cases, the multi-layer perceptron neural network achieves performance comparable to that of the expert model-based controller. Specifically, it mitigates wheel slip and enhances vehicle stability under low-friction conditions in traction control, while accurately tracking the reference trajectory in path following.

Future work will focus on the use of this framework for replicating vehicle dynamics to support the design of neural-network-based MPC (NN-MPC), with particular focus on multi-actuator control systems. As the proposed framework is designed to support multi-input and multi-output control devices, it will be applied to more complex control strategies, such as *i)* combined active suspension and torque vectoring; and *ii)* path following with active cruise control capability.

CRediT authorship contribution statement

Hao Sun: Writing – original draft, Visualization, Software, Methodology. **Carmine Caponio:** Writing – original draft, Methodology. **Marino Massimo Costantini:** Writing – original draft, Methodology. **Mario Mihalkov:** Visualization, Software. **Davide Tavernini:** Writing – review & editing. **Patrick Gruber:** Writing – review & editing. **Alessandro Vigliani:** Writing – review & editing. **Aldo Sorniotti:** Writing – review & editing. **Guodong Yin:** Writing – review & editing. **Umberto Montanaro:** Writing – original draft, Supervision, Conceptualization.

Funding

Partially supported by the European Union's Horizon Europe Research and Innovation Programme (Grant No. 101096083 (EMTECH)). Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or CINEA-European Climate, Infrastructure and Environment Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Declaration of Competing Interest

Guodong Yin is an editorial board member for Chinese Journal of Mechanical Engineering and was not involved in the editorial review or the decision to publish this article. All authors declare that there are no competing interests.

Acknowledgement

Not applicable.

References

[1] Qiang Z, Dai L, Chen B, et al. Model predictive control for on-ramp vehicle merging to a platoon on main road in finite time. IEEE Trans Intell Transp Syst 2025;26(4):4731–43.
[2] Sun H, Li S, Li B, et al. Spatial–temporal transformer-based ecological car-following strategy for connected electric vehicles in dynamic environments. Energy 2025;317:134673.
[3] Cui D, Li H. Density regulation of large-scale robotic swarm using robust model predictive mean-field control. Automatica 2024;169:111832.
[4] Xu B, Dai Y, Suleman A, et al. Distributed fault-tolerant control of multi-UAV formation for dynamic leader tracking: a lyapunov-based MPC framework. Automatica 2025;175:112179.
[5] Sun H, Tavernini D, Gruber P, et al. Distributed model predictive control for vehicle platooning with coupled constraints under asynchronous communication protocols. IEEE Trans Veh Technol 2026;75(5):7298–309.
[6] Mayne DQ. Model predictive control: recent developments and future promise. Automatica 2014;50(12):2967–86.
[7] Morari M, Lee JH. Model predictive control: past, present and future. Comput & Chem Eng 1999;23(4-5):667–82.
[8] Kouvaritakis B, Cannon M. Model Predictive Control: Classical, Robust and Stochastic. Cham, Switzerland: Springer; 2016.
[9] Pan Y, Cheng CA, Saigol K, et al. Imitation learning for agile autonomous driving. Int J Robot Res 2020;39(2-3):286–302.
[10] Gómez PI, Gajardo MEL, Mijatovic N, et al. Enhanced imitation learning of model predictive control through importance weighting. IEEE Trans Ind Electron 2025;72(4):4073–83.
[11] Grüne L, Pannek J. Nonlinear Model Predictive Control: Theory and Algorithms. Cham, Switzerland: Springer; 2016.
[12] Chen H, Allgöwer F. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. Automatica 1998;34(10):1205–17.
[13] Stankovic SS, Stanojevic MJ, Siljak DD. Decentralized overlapping control of a platoon of vehicles. IEEE Trans Control Syst Technol 2000;8(5):816–32.
[14] Li S, Li K, Rajamani R, et al. Model predictive multi-objective vehicular adaptive cruise control. IEEE Trans Control Syst Technol 2011;19(3):556–66.
[15] Hu J, Bhowmick P, Arvin F, et al. Cooperative control of heterogeneous connected vehicle platoons: an adaptive leader-following approach. IEEE Robot Autom Lett 2020;5(2):977–84.
[16] Sun H, Dai L, Chen B. Tube-based distributed model predictive control for heterogeneous vehicle platoons via convex optimization. IEEE 25th International Conference on Intelligent Transportation Systems (ITSC). Macau, China: IEEE; 2022. p. 1122–7.
[17] Sun H, Li B, Zhang H, et al. Ecological electric vehicle platooning: an adaptive tube-based distributed model predictive control approach. IEEE Trans Transp Electrification 2025;11(1):1048–60.
[18] Kim S, Han K, Choi SB. Imitation learning of nonlinear model predictive control for emergency collision avoidance. IEEE Trans Intell Veh 2024;9(1):2908–22.
[19] Nurbayeva A, Shintemirov A, Rubagotti M. Deep imitation learning of nonlinear model predictive control laws for a safe physical human–robot interaction. IEEE Trans Ind Inform 2023;19(7):8384–95.
[20] Caponio C, Mihalkov M, Hankovszki Z, et al. Development of deep reinforcement learning traction controllers for front and rear wheel drive electrified vehicles. WCX SAE World Congr Exp 2025. https://doi.org/10.4271/2025-01-8803.
[21] Chen L, Qin Z, Hu M, et al. Path tracking controller design of automated parking systems via NMPC with an instructible solution. Chin J Mech Eng 2024;37(1):65.
[22] Zhang H, Zhou C, Wang C, et al. An energy efficient control strategy for electric vehicle driven by in-wheel-motors based on discrete adaptive sliding mode control. Chin J Mech Eng 2023;36(1):58.
[23] Tang B, Yang Z, Jiang H, et al. Stability-considered lane keeping control of commercial vehicles based on improved APF algorithm. Chin J Mech Eng 2024;37(1):19.
[24] Dixit S, Montanaro U, Dianati M, et al. Integral MRAC with bounded switching gain for vehicle lateral tracking. IEEE Trans Control Syst Technol 2021;29(5):1936–51.
[25] Chu W, Yang K, Li S, et al. Motion planning for autonomous driving with real traffic data validation. Chin J Mech Eng 2024;37(1):6.
[26] Sun H, Hu J, Dai L, et al. A multi-agent path planning strategy with reconfigurable topology in unknown environments. IEEE 20th International Conference on Automation Science and Engineering (CASE). Bari, Italy: IEEE; 2024. p. 2223–8.

[27] Stano P, Montanaro U, Tavernini D, et al. Model predictive path tracking control for automated road vehicles: a review. Annu Rev Control 2023;55:194–236.
[28] Caponio C, Stano P, Carli R, et al. Modeling, positioning, and deep reinforcement learning path following control of scaled robotic vehicles: design and experimental validation. IEEE Trans Autom Sci Eng 2025;22:9856–71.
[29] Martyanov AS, Solomin EV, Korobatov DV. Development of control algorithms in MATLAB/Simulink. Procedia Eng 2015;129:922–6.
[30] Mihalkov M, Caponio C, Hankovszki Z, et al. Design and analysis of traction control strategies for icy road conditions. 16th International Symposium on Advanced Vehicle Control(AVEC 2024). Cham, Switzerland: Springer; 2024. p. 413–9.
[31] Kaufmann E, Bauersfeld L, Loquercio A, et al. Champion-level drone racing using deep reinforcement learning. Nature 2023;620(7976):982–7.
[32] Tolstikhin IO, Houlsby N, Kolesnikov A, et al. MLP-mixer: an all-MLP architecture for vision. Adv Neural Inf Process Syst 2021;34:24261–72.
[33] Allgöwer F, Zheng A. Nonlinear model predictive control. Control Syst Robot Autom 2012;11:59–84.
[34] Kim P. MATLAB deep learning, with machine learning, neural networks and artificial intelligence Cham, Switzerland: Springer; 2017.
[35] Goodfellow I, Bengio Y, Courville A, et al. Deep learning. Cambridge: MIT Press; 2016.
[36] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature 2015;521(7553):436–44.
[37] Rajamani R. Vehicle dynamics and control. Boston, US: Springer; 2006.

Hao Sun is currently a Research Fellow at the *University of Surrey, UK*. He received his PhD degree from *University College London, UK*, in 2025. His research interests include control, optimization, and machine learning, with applications in intelligent vehicles and robotics.

Carmine Caponio is currently a Research Fellow at the *University of Surrey, UK*. He received his PhD degree from *University of Surrey, UK*, in 2025. His research interests include model-based and artificial intelligence-based solutions for integrated chassis control.

Marino Massimo Costantini is currently a PhD candidate at the *Polytechnic University of Turin, Italy*. He received his Bachelor's degree from the *Polytechnic University of Turin, Italy*, in 2021. His research interests include advanced vehicle state and parameter estimation for vehicle dynamics control.

Mario Mihalkov is currently a Research Fellow at the *University of Surrey, UK*. He received his PhD degree from the *University of Surrey, UK*, in 2026. His research interests include advanced model-based and machine learning algorithms for vehicle dynamics control.

Davide Tavernini is currently a Senior Lecturer at the *University of Surrey, UK*. He received his PhD degree from the *University of Padova, Italy*, in 2014. His research interests include vehicle dynamics modeling and control, applied to over-actuated vehicles.

Patrick Gruber is currently a Professor at the *University of Surrey, UK*. He received his PhD degree from the *University of Surrey, UK*, in 2009. His research interests include vehicle dynamics and control.

Alessandro Vigliani is currently a Professor at the *Polytechnic University of Turin, Italy*. He received his PhD degree from the *Polytechnic University of Turin, Italy*, in 1996. His research interests include electric and hybrid vehicles.

Aldo Sorniotti is currently a Professor at the *Polytechnic University of Turin, Italy*. He received his PhD degree from the *Polytechnic University of Turin, Italy*, in 2005. His research interests include vehicle dynamics control for electric and automated vehicles.

Guodong Yin is currently a Professor at *Southeast University, China*. He received his PhD degree from *Southeast University, China*, in 2007. His research interests include new energy vehicle design and control, vehicle dynamics and control.

Umberto Montanaro is currently a Senior Lecturer at the *University of Surrey, UK*. He received his PhD degree from *the University of Naples Federico II, Italy*, in 2009. His research interests include control theory and its applications in mechatronics and automotive systems.