2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Learning-based Model Predictive Control for an Autonomous Formula Student Racing Car

David R. Gomes, Miguel Ayala Botto, and Pedro U. Lima

***Abstract*— Advancements in Automated Driving Systems (ADSs) have enabled the achievement of a certain level of autonomy while commuting in a car. However, emergency and high-speed maneuvers still arise as significant challenges for ADSs due to the intrinsic nonlinearity and fast-paced behavior of such events. These maneuvers are a distinctive feature within the recently established motorsport discipline of Autonomous Racing (AR). In this work, we explore the use of Learning-based Model Predictive Control (LMPC) to address possible model mismatches of the first principles model in high-speed racing. To this end, a Model Predictive Contouring Control (MPCC) (a specific formulation of the standard Model Predictive Control, MPC) is formulated, and a Neural Network (NN) that leverages the use of Feedforward and Recurrent layers is employed to learn the errors of the first principles model. By combining the NN with the first principles model, the LMPC is born, capable of accurately predicting the future with a computational effort compatible with real-time feasibility, effectively handling the vehicle at its limits. Furthermore, the controller can adapt to changing environments by training the NN during the race. The MPCC (formulation without the NN) is deployed on a real autonomous formula student racing car showing an improvement of 16 % in mean lap times across the same track between a common geometric controller. The LMPC is analyzed in a high-fidelity simulator, achieving an improvement of 8.9 % in mean lap times when compared to the MPCC.**

***Video Playlist* - `www.youtube.com/playlist?list=PLxbUnBTSF_WBt9GTMm6vppqskhR-RlOCe`**

I. Introduction

Recently, a new type of motorsport has emerged entitled autonomous vehicle racing, hereafter, called only autonomous racing (AR). As the name suggests, AR can be defined as the evolving sport of racing ground-based wheeled vehicles, controlled by a computer. In order to promote the research of such complex topics within an academic environment, the Formula Student Driverless (FSD) Cup competition [1] was created. Formula Student is Europe's most established educational engineering competition with more than 100 university teams taking part every year. It is within this environment that the proposed algorithm is tested (refer to Figure 1).

More than entertainment, AR is a competitive way to study open research problems in Automated Driving Systems (ADSs). For a computer to fully take a racing car to the limit, there are several complex problems that need to be tackled. Firstly, it is necessary to detect the vehicle limits in order for the software to know what is it dealing with. This can be related to tire wear, different road/weather conditions, or even the different vehicle setups available. Secondly, the software must make decisions either strategy-wise, e.g. overtaking, or energy-wise, all this at high speeds. Lastly, the software needs to know how to handle the vehicle at its limits, i.e. how to actuate the vehicle to extract the maximum performance. As such, control-wise an autonomous race car far exceeds the software requirements of a normal passenger car, providing helpful insights into new algorithms and relevant research questions [3], hence, being of extreme importance for the development of efficient but also safe ADSs.

Fig. 1: FST12, FST Lisboa's prototype of the year 2023 [2]

Current approaches to the AR problem use Model Predictive Control (MPC) as its base controller [4] [5]. With the advances in non-linear optimization solvers and better computational resources, nonlinear MPC controllers are becoming increasingly popular due to their capacity to optimally control the racing car and add complex constraints to the problem. However, there still needs to be a reasonable trade-off between precision and computational effort, forcing the system's model to be simpler, leading to suboptimal solutions.

This study explores the potential of an MPC formulation that leverages a first principles model combined with a Neural Network (NN) capable of learning model mismatches. The core problem lies in finding the best MPC formulation as well as the best NN to learn such errors, all while keeping real-time feasibility. The broader implications extend beyond the realm of AR, positioning this control paradigm as a versatile solution for problems where abundant plant data

D. Gomes and P. Lima are with the Laboratory of Robotics and Engineering Systems, ISR/IST, Universidade de Lisboa, Portugal. M. Ayala Botto is with IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Portugal. This work was partially supported by the Portuguese Science and Technology Foundation (FCT), through LARSyS - FCT Project UID/50009/2020 (DOI: 10.54499/UIDB/50009/2020 and 10.54499/UIDP/50009/2020) and LARSyS Associate Lab funding LA/P/0083/2020 (DOI:10.54499/LA/P/0083/2020).

is at one's disposal. Hence, the main contributions are:

- **Refined MPC Formulation:** A more concise and explicit cost function is developed, tailoring the control objectives to suit the specific demands of AR.
- **Neural Network Exploration:** Different Neural Network architectures and configurations are systematically explored. This exploration aims to identify the most promising configurations that yield optimal results while managing computational complexity.
- **Computational Analysis:** The computational cost of diverse Learning-based Model Predictive Control (LMPC) formulations are rigorously analyzed. This examination enables a comprehensive understanding of the trade-offs between performance and computational overhead.

The paper is organized as follows: Section II summarizes the research conducted on control algorithms for the AR scenario; Section III presents the proposed controller, focusing on the vehicle model, the MPC formulation, and the NN architecture; Section IV highlights the achieved main results of such an approach and compares the influence of the NN. Finally, Section V concludes with the main point of the proposed approach as well as future work.

II. Related Work

The abstraction of what the control module of an autonomous race car is doing can be pinpointed to the search of a steering angle, δ, and a throttle/brake request, d, which are subsequently sent to the vehicle's low-level controllers to actuate on the car accordingly. Moreover, there is a reference path, x^{ref}, that may or may not be associated with time. The ultimate goal is to design a controller such that for the current vehicle states, $x(t)$, and the reference path, x^{ref}, a control input, $u(t)$, is computed.

According to [6], the control algorithms can be divided into 4 different categories: Classic Control, Model Predictive Control, Learning-based, and End-to-end. Classical Control encapsulates more traditional techniques like PIDs, LQRs, and geometry-based controllers [7] [8]. The MPC category is self-explanatory and the most popular one [9] [10]. Learning-based approaches leverage *Iterative Learning Control* (ILC) based methodologies which work well in AR due to the repeated multiple laps. It is in this category that the proposed algorithm is categorized. Finally, End-to-end controllers are also starting to get more attention with Reinforcement Learning and Imitation Learning as the most common approaches [11] [12].

Regarding the Learning-based approaches, in [13] an optimization-based, data-driven framework is proposed where a terminal cost and a sampled safe set are learned from data to guarantee recursive feasibility and non-decreasing performance cost at each iteration. However, the terminal sets are computed with some safety margins, not allowing the vehicle to fully reach its potential. This approach is tested on a 1:10 scale RC car.

The Machine Learning (ML) algorithm Gaussian Processes is used in [14] in order to come up with a *learning-based cautious* Nonlinear MPC. The ML algorithm is used to identify uncertainties and mismatches in the vehicle dynamics model based on the collected data. The downsides of this approach are the fixed kernels and the low number of inducing points between iterations due to the high control frequency, which consequently, does not effectively approximate the model mismatch. This idea is also successfully applied to an FSD car in [15].

In [16], a two-layer Multilayer Perceptron (MLP) is introduced learning the car's model into a *Deep State Space*. It is then used inside the optimization problem in order to better represent the dynamics of the car. By only relying on a NN to learn the dynamics of the car prior knowledge regarding the vehicle is discarded and not generalizable. This approach is tested on a real sports car.

Moreover, in [17] a *Real-time Neural*-MPC is introduced that scales the previous idea of having some sort of neural network learn the model mismatches, but by linearizing both the cost function as well as the learned dynamics, more complex NNs can be used while maintaining real-time feasibility and have more accuracy. However, short-comings also come from the linearization, not always surpassing a non-linearized version in terms of accuracy.

In [18], a similar concept to the proposed work is persuaded in the sense that prior knowledge about the vehicle's model is integrated into the NN. However, it is done by purely using a model trained with a loss regarding the data fit as well as how far the current fit is apart from the first principles model. This approach is known as PINNs [19] and in [18] was implemented in TUM's Indy Autonomous Racing Car.

Finally, in [20] a similar idea to the proposed work is suggested. An MPCC formulation that leverages an NN to predict the physical model error is presented. However, because of extended computational times, and simple NN structures, the achieved control frequencies remain significantly lower than the desired ones in the context of AR, and the capacity of the NN to learn the model mismatch is not ideal.

The approach presented in this work is inspired by the good preliminary results obtained in [20]. The MPCC formulation as well as the used error correction NN are further investigated and refined to achieve better results. Additionally, the whole NN integration with the first principles model is reformulated in order to become agnostic to the change of coordinates (that induces errors). Moreover, a real-time high-fidelity simulation is used to study the algorithm's performance. Finally, the computational burden is significantly decreased, leading to 52 % faster computational times and 300 % more consistency in solve times (standard deviation of the solve times).

III. Proposed Method

The proposed Learning-based Model Predictive Control (LMPC) formulation leverages the already known Model Predictive Contouring Control (MPCC) with successful use cases in the racing scenario (both in cars [21] and drones

[22]) and some modifications that make it specifically tailored for the FSD scenario (tight curves, small straights, and tight track width). The other key point of the proposed approach is the use of an NN to enhance the model's prediction capability by modelling the dynamical equations:

$$f_{dynamics} = f_{model} + f_{NN}. \tag{1}$$

Where f_{model} corresponds to the first principles model and f_{NN} the NN output. While f_{model} describes the main dynamics in a vehicle, it still suffers from unmodelled phenomena (e.g. actuator delays) and parameter uncertainty (e.g. tire and aerodynamics parametrization), which begins to exert a greater influence on the vehicle's behavior at higher speeds. Therefore, a simple NN (f_{NN}) is introduced that captures the influence of such terms. The final model of the vehicle used consists of a grey-box model that mainly relies on the physical model but has the flexibility to correct itself with the outputs from the NN. This approach has several benefits:

- Since the error of the physical model is what is being learned by the NN (hence, the NN's output), it is centered at zero by nature (if the model was perfect, the error would be zero). Centered data has faster convergence, can prevent bias in the model, and has better numerical stability [23].
- Learning residuals has also proven to be a more effective way to train an NN and can lead to better results, e.g. the CNN ResNet is based on this principle [24].
- By training the NN around the physical model, rather than discarding it entirely, the prior knowledge is effectively incorporated into the NN. Becoming a more generalizable approach than a Deep State Space.
- By using a Neural Network to discover the unmodelled and unknown dynamics, complex systems like the powertrain and suspension do not need to be exactly parametrized, saving time and money.

A. Learning-based Model Predictive Control Formulation

Model Predictive Control (MPC) [25] is a popular optimization-based controller that can handle problem constraints. It uses a mathematical representation of the plant to predict the future, and together with a cost function, it outputs the optimal control inputs. However, if the model is not accurate enough the plant's future behavior will not be properly predicted, accumulating error between the future steps of the MPC.

Physical Model: For the physical model, the planar model [26] was chosen due to its moderate complexity and precision. It is described mathematically in (2).

The inputs of the system are the time derivatives of the throttle, d_d $[\mathrm{s}^{-1}]$, and steering, d_δ $[\mathrm{rad}\cdot\mathrm{s}^{-1}]$. This was chosen since it was identified that using the derivatives rather than the variables themselves produced a much more numerical stable convergence. Additionally, the weights and constraints for these variables in the MPC formulation become independent of the sampling time. Without this change, the convergence times almost doubled.

The states are the car's pose (cartesian coordinates X [m],Y [m], and orientation Ψ [rad]), longitudinal and lateral velocity, v_x $[\mathrm{m}\cdot\mathrm{s}^{-1}]$ and v_y $[\mathrm{m}\cdot\mathrm{s}^{-1}]$, the car's yaw rate, r $[\mathrm{rad}\cdot\mathrm{s}^{-1}]$, and finally, the throttle of the car, d, that can range from 1 (full throttle) to -1 (full braking), and the wheel steering angle, δ [rad]. The states that are on the car frame (v_x, v_y, and r) are assumed to be computed for the center of gravity (CoG). Naturally, m is the mass of the vehicle, I_z is the yaw inertia, l_f and l_r the distance from CoG to the front and rear wheels, respectively, and t_w is the vehicle's track width.

$$\begin{cases} \dot{d} = & d_d \\ \dot{\delta} = & d_\delta \\ \dot{X} = & v_x\cos(\Psi) - v_y\sin(\Psi) \\ \dot{Y} = & v_x\sin(\Psi) + v_y\cos(\Psi) \\ \dot{\Psi} = & r \\ \dot{v_x} = & ((F_{x_{fl}} + F_{x_{fr}})\cos(\delta) - (F_{y_{fl}} + F_{y_{fr}})\cdot \\ & \cdot\sin(\delta) + (F_{x_{rl}} + F_{x_{rr}}) - F_{loss})/m + v_y r \\ \dot{v_y} = & ((F_{x_{fl}} + F_{x_{fr}})\sin(\delta) + (F_{y_{fl}} + F_{y_{fr}})\cdot \\ & \cdot\cos(\delta) + (F_{y_{rl}} + F_{y_{rr}}))/m - v_x r \\ \dot{r} = & ((F_{x_{fl}} + F_{x_{fr}})\sin(\delta)l_f + (F_{y_{fl}} + F_{y_{fr}})\cdot \\ & \cdot\cos(\delta)l_f - (F_{y_{rl}} + F_{y_{rr}})l_r + \tfrac{t_w}{2}\cdot \\ & \cdot(F_{x_{fr}} - F_{x_{fl}})\cos(\delta) + \tfrac{t_w}{2}(F_{y_{fl}} - F_{y_{fr}})\cdot \\ & \cdot\sin(\delta) + \tfrac{t_w}{2}(F_{x_{rr}} - F_{x_{rl}}))/I_z \end{cases} \tag{2}$$

The term F_{loss} is the sum of the forces F_{drag} and F_{roll}, associated with the aerodynamic drag and rolling resistance of the car and can be expressed as

$$F_{\text{roll}} = C_r \cdot m \cdot g, \qquad F_{\text{drag}} = \frac{1}{2}\rho C_d A_F v_x^2,$$

with C_r the rolling coefficient, g the gravitational acceleration, p the air's density, and A_F the vehicle's frontal area.

Furthermore, the other forces are all related to each tire, hence the subscripts $_{fl}$ (front left), $_{fr}$ (front right), $_{rl}$ (rear left), and $_{rr}$ (rear right). The subscript $_x$ is related to the longitudinal force and the subscript $_y$ to the lateral force both applied in the tire and can be obtained as

$$F_x = \left(\frac{\eta \cdot G_R}{r_{\text{effective}}}\right) \cdot T(d, \delta, v_x, a_x),$$
$$F_y = D\sin[C\arctan\{B\alpha - E(B\alpha - \arctan(B\alpha))\}].$$

Regarding the longitudinal force, it depends on the motor's and transmission efficiency, η, the gear ratio, G_R, the effective wheel radius of the tire, $r_{effective}$, and the torque, T. It is important to mention that Torque Vectoring (TV) is being applied in the car as a low-lever controller, depending on the throttle, steering, longitudinal velocity and acceleration (a_x). This behavior is also directly modeled in the physical model.

Concerning the lateral force, it follows the Magic Formula structure [27], with coefficients (D,C,B, and E) obtained by fitting several tire data into the formula. The generated downforce, longitudinal, and lateral load transfers can be modeled to some extent in the D coefficient. This formula solely depends on each wheel slip angle defined as

$$\alpha_{fl/fr} = \delta - \arctan\left(\frac{v_y + l_f r}{v_x \mp r\frac{t_w}{2}}\right), \qquad \alpha_{rl/rr} = -\arctan\left(\frac{v_y - l_r r}{v_x \mp r\frac{t_w}{2}}\right)$$

Note that there are some simplifications regarding this approach, e.g., the longitudinal and lateral forces are decoupled and longitudinal slippage is not being considered.

Formulation: The proposed LMPC formulation is an optimization problem defined as

$$\min_{\substack{u_t,\cdots,u_{t+N-1}\\ v_t,\cdots,v_{t+N}}} \sum_{k=t}^{t+N} q_c\,(\hat{e}_k^c)^2 + q_l(\hat{e}_k^l)^2 - \lambda\, v_k + \beta\,(u_k)^2 + \alpha\,\epsilon^2 \tag{3a}$$

subject to

$$x_{k+1} = f_{model+NN}(x_k, u_k) \tag{3b}$$
$$s_{k+1} = s_k + v_k\, T_s \tag{3c}$$
$$e_k = [X\ Y]^T - g(s_k) \tag{3d}$$
$$\hat{e}_k^c = \left| g_\perp^{'}(s_k)^T \cdot e_k \right| \tag{3e}$$
$$\hat{e}_k^l = \left| g^{'}(s_k)^T \cdot e_k \right| \tag{3f}$$
$$x_0 = x(t);\ u_0 = u(t);\ s_0 = s(t) \tag{3g}$$
$$x_{\min} \leq x_k \leq x_{\max} \tag{3h}$$
$$u_{\min} \leq u_k \leq u_{\max} \tag{3i}$$
$$\left(\frac{a_{x_k}}{\mu_x}\right)^2 + \left(\frac{a_{y_k}}{\mu_y}\right)^2 \leq (g + a_{aero_k})^2 + \epsilon_k. \tag{3j}$$

Where q_c, q_l, λ, β, and α are parameters/weights in the cost function, and T_s is the time between MPC steps. Additionally, μ_x and μ_y are the grips corresponding to the tire-road interaction.

A concise explanation and derivation of the MPCC concept can be seen in [28]. The main idea of this formulation consists on penalizing $\hat{e}_k^c$ (3e) (the approximated distance of the car to the path) and maximizing v_k (the velocity at which the path is being traveled). This allows the controller to come up with its own racing line online (e.g. cutting the chicanes from the inside).

The whole path is turned into a third-order arc-length spline (function g parameterized by the traveled distance with regard to the path, designated as progress, s) in order to have a differentiable function inside the MPC to follow. The functions that give the tangent ($g^{'}$) and orthogonal ($g_\perp^{'}$) vectors with regards to the path are necessary for the formulation as seen in (3e) and (3f). The current and the next step progress can be approximated by v_k (the progress's velocity) (3c). The term $\hat{e}_k^l$ (3f) is related to how well is v_k approximation. Finally, a friction ellipse is inserted into the formulation as a soft constraint (3j) (hence a slack variable, ϵ), which mimics the existing coupling between the longitudinal and lateral forces and asserts the vehicle's limits. The downforce generated in the vehicle is also taken into account in this constraint (a_{aero}). Finally, (3g) corresponds to the initial conditions, (3h) to the state constraints, and (3i) to the input constraints. Note that within the state progression (3b) it depends on both the first principles model but also the NN.

The two proposed formulations are going to be used and compared, the MPCC and the LMPC, with the only difference being that the latter one leverages an NN (seen in (3b)). The proposed MPCC formulation was taken into competition and the **video**[1] shows one of its runs in Formula Student Spain (11th of August of 2023, 1st Place Overall Driverless Category) [29], showcasing the performance of such a controller and how good of a base it is to build upon.

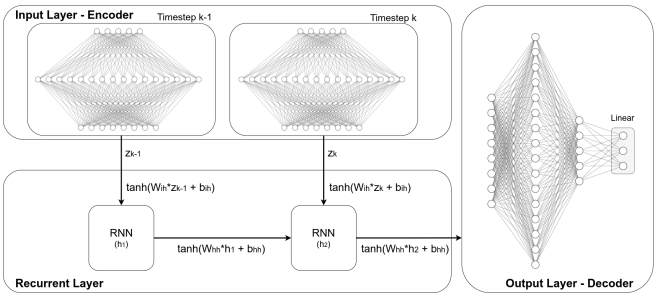

Fig. 2: Encoder-Decoder Recurrent Neural Network

B. NN Architecture

When looking for the best NN architecture to capture the first principle model errors, two main factors need to be considered: complexity and effectiveness. Since the NN will be used directly inside MPC's optimization problem, it cannot be too complex or the optimization problem will not converge fast enough. However, by selecting a too simple NN it will not be able to effectively model the errors.

The two most popular choices in Deep State Space Models for Nonlinear System Identification [30] are Multi-Layer Perceptrons (MLPs) and Recurrent NNs (RNNs). However, another possible approach is to combine them. This takes advantage of the RNN capability to extract information related to the sequential nature of the data, but also uses MLPs to map the input into a bigger latent space capable of capturing more aspects and information from the input data (and consequently, outputs). This is the motivation behind the proposed approach and a visual representation can be seen in Figure 2 where two timesteps are used and only the last hidden state is decoded.

In order to keep the NN reasonably simple, the input features are chosen to be v_x, v_y, r, d, and δ. However, this does not mean that the NN is not taking into account other variables. For example, by having access to both velocities in distinct timesteps the NN can learn its own representations of accelerations. As for the output it corresponds to the three errors related to the local states: e_{v_x}, e_{v_y}, and e_r, since the other states depend on these.

Data Acquisition: Data is collected at the same frequency as the controller. It is not acquired at higher rates in order to learn actuator delays and other unknown dynamics related to the time between iterations in the MPC. The ground truth is assumed to be the sensor/state estimator at use, in this case, an Extended Kalman Filter from [31]. The quality of this data will also determine the quality of the error being learned by the NN, hence it is extremely important to have a trustworthy state estimator. Based on the current states and

[1] https://youtu.be/I2oAC1NEV2Q

TABLE I: Summary of some analyzed NNs architectures

	Model Architecture	Nº Parameters	Test Loss (MSE)
1)	MLP 5-32-32-32-3	2403	0.4733
2)	MLP 5-64-64-64-3	8899	0.3756
3)	RNN 2-timesteps (hidden state of 32)	1347	0.3001
4)	RNN 10-timestps (hidden state of 32)	1347	0.3548
5)	LSTM 2-timesteps (hidden state of 32)	5091	0.2357
6)	LSTM 10-timesteps (hidden state of 32)	5091	0.2509
7)	GRU 2-timesteps (hidden state of 32)	3843	0.2445
8)	GRU 10-timesteps (hidden state of 32)	3843	0.2624
9)	MLP(5-32-32) - RNN(2-timesteps) - MLP(32-32-5-3)	4599	0.2292
10)	MLP(5-32-32) - LSTM(2-timesteps) - MLP(32-32-5-3)	10935	**0.2163**
11)	**MLP(5-16-8) - RNN(2-timesteps) - MLP(8-16-5-3)**	623	0.2998
12)	MLP(5-8-16) - RNN(2-timesteps) - MLP(16-8-5-3)	935	0.2893
13)	MLP(5-8) - RNN(2-timesteps) - MLP(8-5-3)	**255**	0.3844

inputs, the next timestep states can be computed using both the dynamic equations of the first principles model and a first-order forward Euler integration method. Despite simple, such integrator method is enough since the NN will learn the exact necessary contribution for this method to achieve the exact future states, not being necessary to use more complex methods such as Runge-Kutta. The current states/inputs and predicted states are stored along a run. In the post-processing an error of the dynamic equations related to the local states is extracted and used to train the NN. The same can be done online, but some parameters need to be chosen, such as the amount of data needed before training the NN. The errors can be mathematically obtained as in (4). Again, the exact contribution needed for a first-order forward Euler integration method.

$$e_{v_x/v_y/r} = \frac{x_{real} - x_{predicted}}{\Delta t}. \tag{4}$$

A sliding window approach with stride 1 was used to generate sequences from the racing data. A total of 30 minutes of racing was used to train the network. A split of 57-19-25 was adopted for the training, validation, and testing, respectively. The model is trained with ADAM [32] and a learning rate of 0.01 with a decay of 0.5 every 30 epochs. The number of epochs was chosen 150 and the batch size of 32. Each epoch takes around 1s to train in the used setup (NVIDIA RTX 2080 Mobile). Finally, the NN is trained with the Mean Squared Error (MSE), i.e. the difference between the predicted errors, $f_{NN}(v_{x_{t/t-1}}, v_{y_{t/t-1}}, r_{t/t-1}, d_{t/t-1}, \delta_{t/t-1})$, and the real ones, $e_{v_x/v_y/r}$.

Experiments: Several experiments regarding the architecture and hyperparameters were conducted in order to reach a compromise between complexity and accuracy. In Table I a summary of different experiments showcases possible NNs and their capacity to learn the error. Note that the hidden space in the RNN of the encoder-decoder architectures (models 9-13) matches the dimension of the output in the encoder layer.

Looking at Table I, RNNs in general, work much better than simple MLPs. Regarding the amount of timesteps used for the RNNs, two timesteps (current and last past) give the best result across all RNNs. The encoder-decoder RNN architectures do have the best results and allow the RNN to achieve similar outcomes as the LSTM [33] and GRU [34], being two layers of encoding/decoding the sweet spot between performance and complexity. All in all, the chosen architecture was Model 11, due to its trade-off between complexity (under 1000 parameters) and accuracy.

TABLE II: Summary of the activation functions studied

Activation Function	Mean Convergence Time [s]	Convergence Rate [%]	Test Loss (MSE)
No NN (baseline)	0.0075	99.63	N/A
ReLU	$\geq$1	N/A	0.4511
Softplus	0.0208	90.02	0.3954
GeLU	**0.0109**	**98.51**	**0.2998**

Another important analysis to make has to do with the activation functions. Since the NN will be placed inside the LMPC optimization problem, it needs to be a differentiable function. Using ReLUs might pose a barrier in allowing the solver to converge due to the lack of smoothness throughout the whole function. In order to analyze the impact of activation functions three distinct ones are studied: ReLU, GeLU, and softplus

$$\begin{aligned} \text{act}_{\text{ReLU}}(z) &= \max(0, z), \\ \text{act}_{\text{GeLU}}(z) &= z \cdot \frac{1}{2}[1 + \text{erf}(z/\sqrt{2})], \\ \text{act}_{\text{softplus}}(z) &= \log(1 + e^z). \end{aligned} \tag{5}$$

Table II shows the comparison for model 11 when using the three distinct activation functions in the MLPs (encoder-decoder layers). From the analysis made, GeLU outperforms the other two by a wide margin, additionally having a standard deviation of the convergence time of 7.9×10^{-4} s. Note that since the ReLU does not even converge under a second, there is no point in studying the convergence rate.

IV. Results

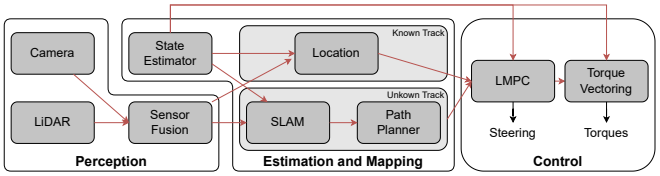

Fig. 4: Diagram of FST Lisboa's pipeline

Setup: All the data used to train the NN was taken from several tests done with FST12, FST Lisboa latest prototype (Figure 1). These tests were performed in several environments, wet, dry, racing asphalt, and asphalt with gravel. FST12 is a 4WD electric car equipped with an Intel Core i5-11600, with 2 sticks of 16 GB of DDR4. The autonomous systems pipeline can be visualized in Figure 4 where three main modules appear: Perception, Estimation & Mapping, and Control. Note that the only part being modified in this graph is the LMPC node, which outputs the steering and the throttle that is sent to the TV node. The whole pipeline is implemented in C++ within the ROS framework. In order to generate the LMPC the software FORCESPRO is used [35] [36]. Regarding the NN, Pytorch [37] is used for offline training, and its C++ API (LibTorch) is used for online training. A high-fidelity simulator that makes use of IPG CarMaker [38] is employed to evaluate the proposed approach.

When tested in the onboard PC the LMPC manages to reach frequencies up to 50 Hz. However, due to the

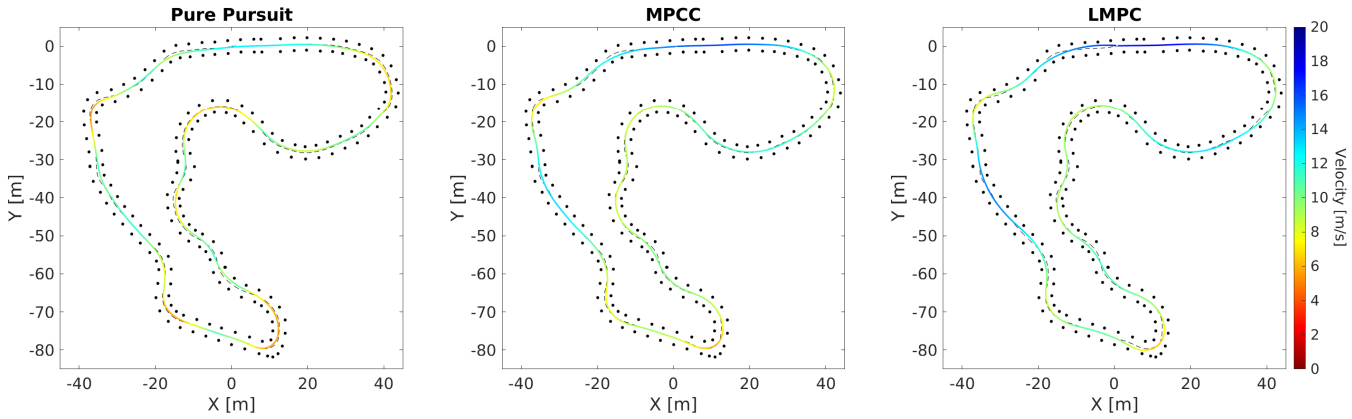

Fig. 3: Visualization of the path and mean velocities between controllers in Simulation for the FSG2023 track

pose estimation frequency (the LiDAR runs at 20 Hz) and mechanical delays in the actuators, the controller's frequency was locked at 20 Hz. When the LMPC does not converge in a specific iteration the last optimal solution is brought back and the control input related to the current timestep is used. In the extreme case (never reported in actual testing) where the LMPC does not converge and all steps of the last converged solution were already used, it falls to a simpler known geometric controller called pure pursuit [39]. Since most of the time when the LMPC does not converge it is only for one iteration, this does not have a significant effect on the controller's performance. Furthermore, the used horizon (N) is 51 steps equivalent to predicting 2.5 s into the future.

Prediction Capabilities: Table III shows the gain in performance when using the offline trained NN. The incorporation of the NN predicts better (low RMSE), generalizes well (low max error), and is more consistent (low standard deviation).

TABLE III: Model Mismatch from all the test data

	RMSE		**Max Error**		**Standard Deviation**	
	First Principles	First Principles + NN	First Principles	First Principles + NN	First Principles	First Principles + NN
v_x	0.0652	**0.0327**	**0.4031**	0.5451	0.0572	**0.0326**
v_y	0.2808	**0.0378**	1.5877	**0.2268**	0.2765	**0.0378**
r	0.1344	**0.0469**	0.4709	**0.2299**	0.1312	**0.0466**

Track Performance: When put in track, the LMPC also outperforms the version without the error correction NN (MPCC), and the pure pursuit (baseline, previous controller in FST12), as shown in Figure 3 and Table IV. Note that, the LMPC is particularly better than the MPCC in curves than straights, which aligns with the data from Table III where the difference in error between models is bigger in v_y than v_x.

Online NN: Lastly, the NN's capacity to adapt to changing conditions is also tested. In an experiment, the car's torques are decreased by 2Nm. After racing for one lap, the gathered data is used to train and update the NN online (hence the name Online NN). The associated RMSE for each state is shown in Table V. Showcasing the adaptiveness of the NN to different scenarios it has not seen before, and also its superiority (in both offline and online cases) when compared to the normal first principles model. The same experiment is done for different tire grips, producing identical results, but due to space constraints, it is not shown here.

TABLE IV: Quantitative Analysis of Figure 3

Controller	Mean Velocity [m/s]	Mean Lap Time [s]
Pure Pursuit	8.66	36.24
MPCC	9.96	32.17
LMPC	**10.89**	**29.32**

TABLE V: RMSE comparison in online learning

	First Principles	Offline NN	Online NN
v_x	0.1034	0.0100	**0.0026**
v_y	0.0757	0.0009	**0.0004**
r	0.0059	0.0025	**0.0010**

V. Conclusions and Future Work

In this work, an LMPC formulation tailored to the AR problem is designed and implemented. Furthermore, an encoder-decoder RNN learns the first principles model mismatches, and takes them into account for the final state progression model. The network is trained with data from several runs with an autonomous Formula Student racing car, and high-fidelity simulations show an improvement when compared to a common geometric-controller and when not using the error correction NN.

For future work, other similar yet different approaches can be explored, such as the newly created concept of Physically-informed Neural Networks (PINNs) [19] [18]. These NNs also try to incorporate previous knowledge of the model directly into the loss function of the NN. It would be interesting to compare both approaches in the same scenario and see which one outperforms.

REFERENCES

[1] F. S. Germany, "Formula student germany driverless official website," 2023. [Online]. Available: https://www.formulastudent.de/teams/fsd/

[2] Fst lisboa. https://pt.fstlisboa.com/. Accessed: 1st of September of 2023. [Online]. Available: https://pt.fstlisboa.com/

[3] A. Jiang, "Research on the development of autonomous race cars and impact on self-driving cars," *Journal of Physics: Conference Series*, vol. 1824, no. 1, p. 012009, Mar. 2021. [Online]. Available: https://doi.org/10.1088/1742-6596/1824/1/012009

[4] J. L. Vázquez, M. Brühlmeier, A. Liniger, A. Rupenyan, and J. Lygeros, "Optimization-based hierarchical motion planning for autonomous racing," 2020.

[5] A. Wischnewski, M. Euler, S. Gümüs, and B. Lohmann, "Tube model predictive control for an autonomous race car," *Vehicle System Dynamics*, vol. 60, no. 9, pp. 3151–3173, 2022. [Online]. Available: https://doi.org/10.1080/00423114.2021.1943461

[6] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022.

[7] K. L. Talvala, K. Kritayakirana, and J. C. Gerdes, "Pushing the limits: From lanekeeping to autonomous racing," *Annual Reviews in Control*, vol. 35, no. 1, pp. 137–148, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578811000101

[8] V. Sukhil and M. Behl, "Adaptive lookahead pure-pursuit for autonomous racing," 2021. [Online]. Available: https://arxiv.org/abs/2111.08873

[9] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1433–1440.

[10] J. V. Carrau, A. Liniger, X. Zhang, and J. Lygeros, "Efficient implementation of randomized mpc for miniature race cars," in *2016 European Control Conference (ECC)*, 2016, pp. 957–962.

[11] A. Tătulea-Codrean, T. Mariani, and S. Engell, "Design and simulation of a machine-learning and model predictive control approach to autonomous race driving for the f1/10 platform," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6031–6036, 2020. [Online]. Available: https://doi.org/10.1016/j.ifacol.2020.12.1669

[12] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," in *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, Jun. 2018. [Online]. Available: https://doi.org/10.15607/rss.2018.xiv.056

[13] M. Brunner, U. Rosolia, J. Gonzales, and F. Borrelli, "Repetitive learning model predictive control: An autonomous racing example," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 2545–2550.

[14] L. Hewing, A. Liniger, and M. N. Zeilinger, "Cautious nmpc with gaussian process dynamics for autonomous miniature race cars," in *2018 European Control Conference (ECC)*, 2018, pp. 1341–1348.

[15] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[16] N. A. Spielberg, M. Brown, and J. C. Gerdes, "Neural network model predictive motion control applied to automated driving with unknown friction," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 5, pp. 1934–1945, 2022.

[17] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural-mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," 2022. [Online]. Available: https://arxiv.org/abs/2203.07747

[18] P. Karle, F. Török, M. Geisslinger, and M. Lienkamp, "Mixnet: Physics constrained deep neural motion prediction for autonomous racing," *IEEE Access*, vol. 11, pp. 85 914–85 926, 2023.

[19] J. Nicodemus, J. Kneifl, J. Fehr, and B. Unger, "Physics-informed neural networks-based model predictive control for multi-link manipulators," *IFAC-PapersOnLine*, vol. 55, no. 20, pp. 331–336, 2022, 10th Vienna International Conference on Mathematical Modelling MATHMOD 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896322013118

[20] G. Costa, J. Pinho, M. A. Botto, and P. U. Lima, "Online learning of mpc for autonomous racing," *Robotics and Autonomous Systems*, vol. 167, p. 104469, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889023001082

[21] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[22] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," 2022.

[23] L. Huang, J. Qin, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "Normalization techniques in training dnns: Methodology, analysis and application," 2020.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[25] J. Rawlings, "Tutorial overview of model predictive control," *IEEE Control Systems Magazine*, vol. 20, no. 3, pp. 38–52, 2000.

[26] R. Rajamani, *Vehicle Dynamics and Control*, ser. Mechanical Engineering Series. Springer US, 2006. [Online]. Available: https://books.google.pt/books?id=sWPwXmkeHD8C

[27] H. B. Pacejka and E. Bakker, "The magic formula tyre model," *Vehicle System Dynamics*, vol. 21, no. sup001, pp. 1–18, 1992. [Online]. Available: https://doi.org/10.1080/00423119208969994

[28] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, p. 628–647, Jul. 2014. [Online]. Available: http://dx.doi.org/10.1002/oca.2123

[29] Formula student spain. https://www.formulastudent.es/. Accessed: 1st of September of 2023. [Online]. Available: https://www.formulastudent.es/

[30] D. Gedon, N. Wahlström, T. B. Schön, and L. Ljung, "Deep state space models for nonlinear system identification," *IFAC-PapersOnLine*, vol. 54, no. 7, pp. 481–486, 2021, 19th IFAC Symposium on System Identification SYSID 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896321011800

[31] N. Salgueiro, "Torque vectoring control of an electric vehicle with in-wheel motors," Master's thesis, Instituto Superior Técnico, Lisboa, 2020.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[35] A. Domahidi and J. Jerez, "Forces professional," Embotech AG, url=https://embotech.com/FORCES-Pro, 2014–2019.

[36] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "Forces nlp: an efficient implementation of interior-point... methods for multistage nonlinear nonconvex programs," *International Journal of Control*, pp. 1–17, 2017.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[38] IPG Automotive GmbH, "Carmaker," CarMaker — IPG Automotive, September 2023, [Online]. [Accessed: 1-Set-2023].

[39] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-92-01, January 1992.