École polytechnique fédérale de Lausanne

Semester Project

Master in Computational Science and Engineering

Neural system identification and control for Formula Student Driverless Cars

Author:
Tudor Oancea

Supervisors:
Yingzhao Lian
Prof. Colin Jones

Contents

1 Introduction

In recent years, autonomous driving has emerged as a technological pinnacle, revolutionizing the way we perceive transportation and paving the way for a future with increased safety and efficiency on the roads. While autonomous vehicles have made significant strides in everyday driving scenarios, another exciting dimension of this technology has captured the imagination of engineers and racing enthusiasts alike - autonomous racing.

Autonomous racing represents the fusion of cutting-edge autonomous driving technology and the high-performance world of motorsports. In contrast to regular driving, where the emphasis is on safety and adherence to traffic rules, racing pushes the limits of speed, agility, and precise control. It particularly drives the engineering community to develop more and more optimized control algorithms that mix complexity and real-time feasibility.

Classical control strategies, such as Model Predictive Control (MPC), have been successfully employed in the context of autonomous racing, leveraging accurate mathematical models of vehicle dynamics and racetrack conditions. However, these strategies face challenges when it comes to dealing with uncertainties and model-plant mismatch encountered in real-world racing environments.

In recent years, neural networks have gained popularity in the field of control systems, particularly through reinforcement learning algorithms like Deep Q-Networks (DQN), Deep Deterministic Policy Gradients (DDPG), and other actor-critic methods. These algorithms have shown promise in handling the continuous states involved in autonomous driving tasks. However, a common limitation of these methods is their sample inefficiency, requiring a significant amount of training data to achieve optimal performance.

To address these challenges, more specialized techniques known as 'imitation-based learning' have emerged, drawing inspiration from supervised learning. These methods utilize examples extracted from an expert agent, often a human driver, to train deep neural networks to imitate the desired control inputs. Additionally, attempts have been made to train neural networks to mimic the outputs of a model predictive control algorithm, combining the flexibility of neural networks with the well-established guarantees of system-theory-validated methods like MPC.

However, one common issue faced by these neural network-based control methods is the lack of accounting for constraints. Guaranteeing constraint satisfaction is crucial in autonomous racing to prevent unsafe or undesirable behaviors. Unlike MPC, which can ensure constraint satisfaction by design, neural network-based approaches often struggle to provide such guarantees.

Recently, a team of researchers from the Pacific Northwest National Laboratory has proposed an innovative solution to address the challenges of system identification and control using a novel approach based on differentiable programming. The system identification problem can be modelled with either Neural State-Space Models (NSSM) or Neural Ordinary Differential Equations (NODE), which are proven methods that can capture intricate nonlinear dynamics with relatively shallow networks [Rah+22]. Meanwhile, for the control task, a neural network is trained to approximate a state-feedback law by leveraging back-propagation and drawing inspiration from model predictive control (MPC). This approach, known as Differentiable Predictive Control (DPC) [DTV22],[Drg+22a],[Drg+22b], extends Explicit Model Predictive Control by incorporating data-based techniques while remaining computationally feasible for medium to large horizon sizes. Furthermore, DPC offers the flexibility to explicitly incorporate constraints during the learning process, which is crucial in domains such as race car control. It has a smaller computational and memory footprint than regular MPC, which makes it suited for deployment on embedded platforms.

In this project, we will investigate the application of NODEs and DPC to a particular autonomous racing application: Formula Student Driverless competitions, student-led and

industry-supported events designed for budding engineers to develop their practical skills and accumulate experience in the automotive industry. This work was realized in the context of the EPFL Racing Team, an EPFL MAKE project competing in these competitions.

There are two main motivations for this project:

1. produce a system model that is more precise than the first-principles models currently used by the EPFL RT Driverless team to create more lightweight and realistic Model in the Loop (MiL) tests.

2. devise a control scheme that approximated well the existing MPC scheme while having a lighter computational and memory footprint.

We will first give in section 2 some context on the Driverless autonomous competitions, as well as the typical software architecture used by competing teams. We then introduce our work on system identification and control in sections 3 and 4 respectively, and detail our implementation, the dataset creation and the experiments we carried out in simulation.

2 Context and previous work

2.1 Formula Student competitions and EPFL Racing team

Formula Student competitions, known for their exhilarating blend of engineering prowess and motorsport excitement, serve as a platform where aspiring engineers can showcase their skills and innovation. These competitions are more than just thrilling racing events; they have a primary goal of providing educational opportunities for future engineers, allowing them to apply theoretical knowledge in a practical and competitive environment. The significance of these competitions extends beyond academia, with industrial partners actively participating and often recruiting talent from Formula Student teams. The influence of the industry is evident in the evolution of the events themselves, as they have transitioned from combustion-based competitions to focusing on electrical and autonomous events. This shift highlights the industry's recognition of the importance of developing expertise in cutting-edge technologies, particularly in the autonomous driving domain. It signals a future where autonomous events may dominate, aligning the competitions with the evolving needs and advancements in the automotive industry.

The EPFL Racing Team is a Formula Student team, gathering more than 100 students. This young team has been created in 2019 and has built 4 high-performance electrical race cars: Orion (2019), Mercury (2021), Artemis (2022) and Ariane (2023) that you can find in figure 2. This last iteration has the biggest number of technological improvements in the team's history: a full-carbon monocoque, a 4-wheel-drive powertrain, the most complete aerodynamic package ever, a redesigned and more durable battery pack, and autonomous driving capabilities. It includes a comprehensive sensor suite (Camera, LiDAR, Inertial Navigation Systems, wheel encoders and high precision RTK positioning system) as well as powerful computing platforms (a National Instruments sbRIO 9627 operating the low-level control, estimation and safety software, and an Nvidia Jetson Orin AGX unit supporting the Driverless system) enabling the team to deploy complex real-time algorithms that will be described in the following section.

2.2 Driverless pipeline

Most Formula Student Driverless teams follow the same global software architecture comprising three main tasks: *Perception*, *Estimation* and *Control*. We provide below a brief description of this Autonomous Racing System (ARS) and a schematic summary of the global concept in figure 3

Figure 2: Ariane, EPFL Racing Team's latest single-seater autonomous electrical race car

2.2.1 Perception

The first task is to use the monocular RGB camera and the LiDAR to detect the positions of the cones delimiting the racetrack (as shown in figure 4) relative to the car. To this end, a complex algorithm merging the information from both sensors is employed to first detect the cones' bounding boxes on the received image (using object detection state-of-the-art algorithm YOLO v7 [WBL22]), then project the LiDAR's point cloud on those boxes to isolate the points corresponding to the cones, and finally cluster them to retrieve the cones' range and bearing.

2.2.2 Estimation

The second task is two-fold:

1. A *velocity estimation* procedure determines the current longitudinal velocity, lateral velocity and yaw rate of the car by fusing the data coming from the wheel encoders and the INS with an Extended Kalman Filter (EKF).

2. Another EKF is used to combine the velocity estimation and the cones observations to perform landmark-based *Simultaneous Localization And Mapping (SLAM)*.

2.2.3 Control

The third task is again two-fold:

1. Plan an appropriate trajectory and velocity profile that minimizes the lap time while still respecting the track boundaries.

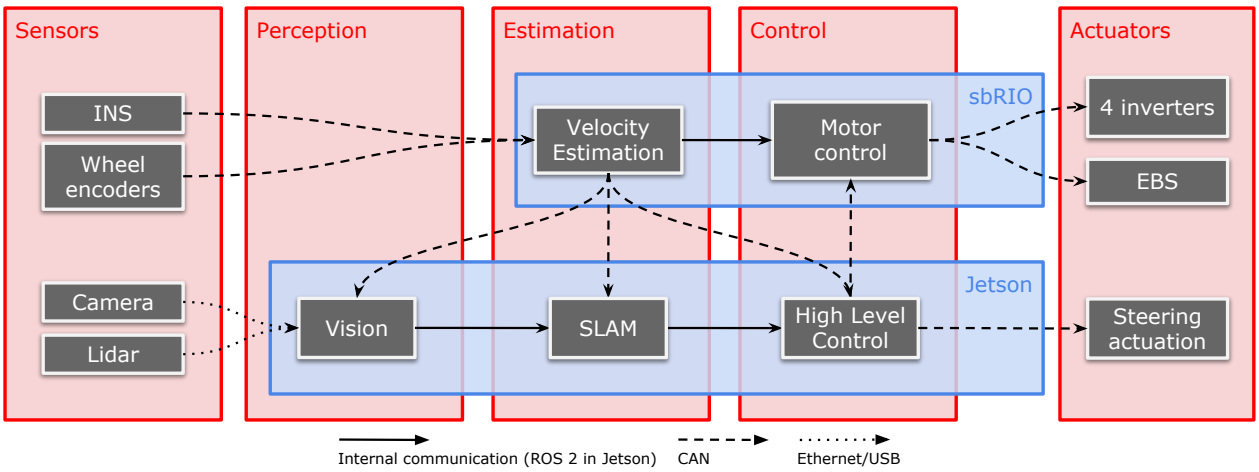


Figure 3: High-level organization of our driverless concept, where each independent module is represented by a gray box. The blue boxes indicate the device the modules run on, and the line styles correspond to the communication form.

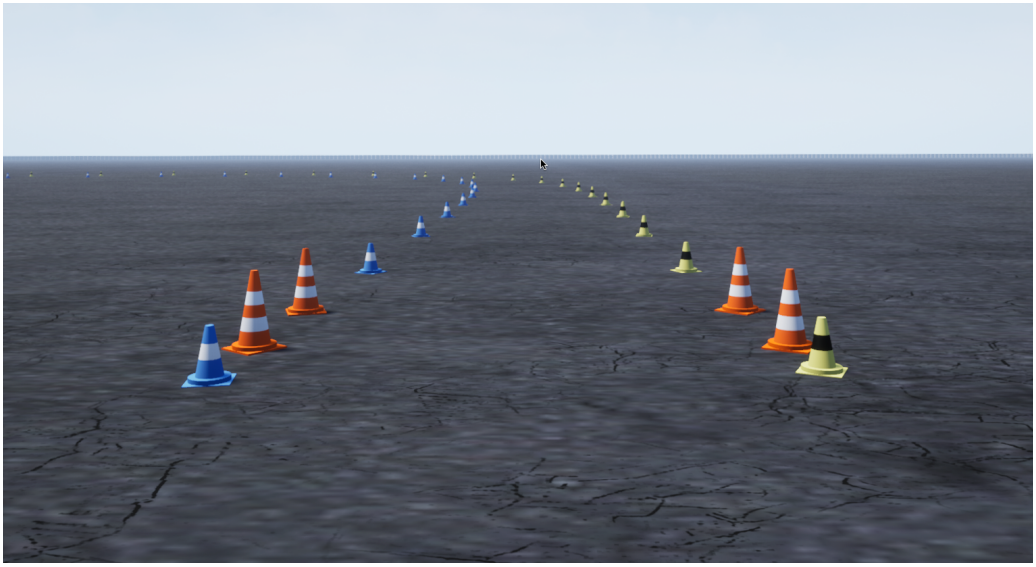


Figure 4: Example of cones delimiting a typical FS racetrack

2. Compute the steering and motor commands to follow this motion plan as closely as possible.

While several approaches combine the motion planning and control tasks [LDM14], we have made the choice to treat them separately to reduce the nonlinearity of the optimal control problems and simplify the tuning procedure of the controller. Another important simplification we made is to separate the motor control into a high-level controller that only computes a global torque command, thus allowing us to use simple bicycle models in the high-level controller, and a low-level controller that finally computes the torque distribution between the four wheels using torque-vectoring algorithm.

Our motion-planning algorithm runs offline and is only used on tracks that are known beforehand (which is the case for most of the FS events). It is solely based on the formula

$$a_y = \frac{v_x^2}{R} = \kappa v_x^2, \tag{1}$$

where a_y is the lateral acceleration, v_x is the longitudinal velocity, and R, κ are respectively the corner radius and curvature. It takes as input a maximal velocity $v_{x,\max}$, a maximal lateral

acceleration $a_{y,\max}$ and the tracks' center line and right/left widths. It first computes a reference trajectory that minimizes the total curvature and therefore maximizes the longitudinal velocity attainable for a given lateral acceleration. Then, using equation (1), it computes the reference velocity profile as

$$v_x^{\text{ref}} = \min(v_{x,\max}, \sqrt{\frac{a_{y,\max}}{\kappa}}) .$$

The final outputs are the reference trajectory, heading and longitudinal velocity as four interpolating splines $X^{\text{ref}}(s), Y^{\text{ref}}(s), \varphi^{\text{ref}}(s), v_x^{\text{ref}}(s)$ of a continuous parameter s denoting the curvilinear position of a point on the reference, as well as temporal reformulations $X^{\text{ref}}(t)$, $Y^{\text{ref}}(t)$, $\varphi^{\text{ref}}(t)$, $v_x^{\text{ref}}(t)$ based on a passage time profile $t(s)$ obtained by integration of the velocity profile.

At each sampling time, the controller will project the current position on the reference trajectory to find the current curvilinear position $\hat{s}$ and the corresponding passage time $\hat{t}$, and then extract a reference horizon of size N_f by evaluating the reference splines at times $\hat{t}, \hat{t}+dt, \hat{t}+2dt, \ldots, \hat{t}+N_f dt$ where dt is the sampling period. This reference horizon $\left\{(X_k^{\text{ref}}, Y_k^{\text{ref}}, \varphi_k^{\text{ref}}, v_{x,k}^{\text{ref}})\right\}_{k=0\ldots,N_f}$ is then tracked by a Nonlinear Model Predictive Controller (NMPC) using a kinematic bicycle model, running at 20Hz with a prediction horizon of 2s discretized into 40 control intervals. The exact NMPC formulation is described in more detail in section 4.

2.3 Simulation environment

To be able to test our algorithms in realistic conditions without actually deploying them on a real car, we used the *Formula Student Driverless Simulator* (FSDS), a comprehensive and powerful simulation environment that was used in the Formula Student Online competition in 2020. It is based on the Microsoft AirSim [Sha+17] project and simulates FS-like cars, tracks, and a wide array of sensors that are commonly used in the FS community. We have developed our own ROS 2 bridge to interact with FSDS in an efficient and streamlined manner and have developed a Software in the Loop (SiL) testing environment: the Nvidia Jetson runs our ARS and is connected via Ethernet to a secondary computer running an FSDS instance and communicating with the Jetson via TCP.

Figure 5: Formula Student Driverless Simulator

3 System identification

In this section, we treat the first objective of our project: produce a system model that is more precise than the current first-principles models currently used by the EPFL RT Driverless team to create a more lightweight and realistic Model in the Loop (MiL) tests.

We first formally introduce in subsection 3.1 our system, notations, as well as the classical bicycle models. We then introduce the NODE learning procedure we designed in subsection 3.2, the dataset we collected in subsection 3.3 and finally our experimental results in subsection 3.4.

3.1 System description

As it is very common in Formula Student competitions, and more generally in the autonomous driving industry, we model our system using variations of the bicycle model [Raj06]. We will denote $\tilde{x} = (X, Y, \varphi, v_x, v_y, r)$ the car state variable and $\tilde{u} = (T, \delta)$ the car controls. Here (X, Y, φ) is the pose in the global fixed reference frame defined by the initial pose of the car (in such a way that this initial pose is $(X_0, Y_0, \varphi_0) = (0, 0, \frac{\pi}{2})$). The longitudinal and lateral velocities (v_x, v_y) are expressed in the car local's reference frame. The control inputs are T the traction command that is normalized between -1 and 1, with 1 corresponding to full throttle and -1 corresponding to full brake, and δ the steering angle. You can find in figure 6 a schematic representation of these variables as well as the other physical variables used in the definitions below.

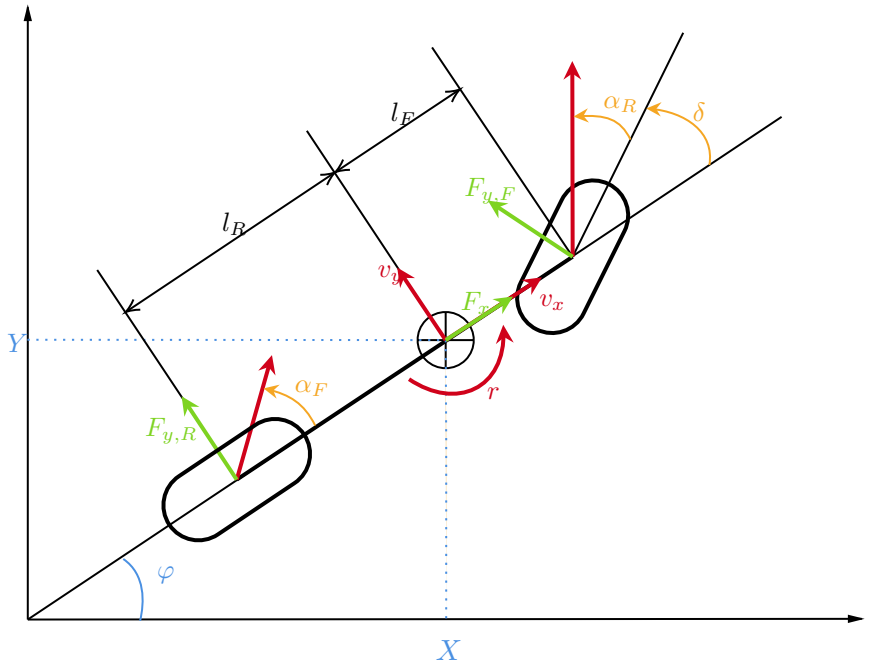


Figure 6: Bicycle model diagram

The classical 6 DoF dynamical bicycle model is expressed as such:

$$\dot{X} = v_x \cos(\varphi) - v_y \sin(\varphi) \tag{2a}$$
$$\dot{Y} = v_x \sin(\varphi) + v_y \cos(\varphi) \tag{2b}$$
$$\dot{\varphi} = r \tag{2c}$$
$$\dot{v}_x = \frac{1}{m}(F_x - F_{y,F} \sin(\delta) + m v_y r) \tag{2d}$$
$$\dot{v}_y = \frac{1}{m}(F_{y,R} + F_{y,F} \cos(\delta) - m v_x r) \tag{2e}$$
$$\dot{r} = \frac{1}{I_z}(F_{y,F} l_F \cos(\delta) \ - F_{y,R} l_R) \tag{2f}$$

where I_z is the car moment of inertia around a vertical axis, l_R and l_F are the distances between the center of mass and the rear and front axle respectively, F_x is the longitudinal force developed by the traction system, and $F_{y,R}$ and $F_{y,F}$ are the lateral friction forces applied on the rear and front tires respectively.

The longitudinal force is usually modeled as

$$(C_{m1} - C_{m2} v_x) T - C_{r0} \tanh(C_{r3} v_x) - C_{r2} v_x^2 \tag{3}$$

while the lateral forces are expressed using a Pacejka tire model [Pac06] as

$$F_{y,j} = D_j \sin(C_j \arctan(B_j \alpha_j)),\ j \in \{R, F\}$$

where

$$\alpha_R = \arctan\left(\frac{v_y - l_R r}{v_x}\right),\ \alpha_F = \arctan\left(\frac{v_y + l_F r}{v_x}\right) - \delta$$

are the slip angles of each tire. A notable limitation of this model lies in its inadequate characterization of slip angles at low velocities, rendering it unsuitable for optimal control problems because of the significant numerical inaccuracies induced in the computation of sensitivities.

To mitigate these issues, one can use a 4 DoF kinematic model [Ver+14] defined as follows:

$$\dot{X} = v_x \cos(\varphi + \beta) \tag{4a}$$
$$\dot{Y} = v_x \sin(\varphi + \beta) \tag{4b}$$
$$\dot{\varphi} = \frac{v}{l_R} \sin(\beta) \tag{4c}$$
$$\dot{v} = \frac{F_x}{m} \cos(\beta) \tag{4d}$$

where v is the absolute velocity and $\beta = \arctan\left(\frac{l_R}{l_R + l_F} \tan(\delta)\right)$ is the kinematic approximation of the car's slip angle. This model assumes that δ (and therefore β) are small to approximate $v \approx v_x$.

This model also has the advantage of having less parameters to identify (only the ones defining the longitudinal model (3), the other parameters being directly measurable on the car), which is the reason why it was chosen for the NMPC controller previously developed at by the EPFL Racing Team.

In the following, the continuous-time models such as (2) or (4) will be discretized using an explicit Runge-Kutta scheme of order 4 (RK4), yielding the discrete time model $\tilde{x}^+ = \tilde{f}_1(\tilde{x}, \tilde{u})$. Such a discrete-time model can then be unrolled into a N_f-step open loop prediction denoted by $\tilde{x}_{1:N_f} = \tilde{f}_{N_f}(\tilde{x}_0, \tilde{u}_{0:N_f-1})$ where $a_{i:j}$ denotes the sequence $(a_i, a_{i+1}, \dots a_j)$.

3.2 System identification with NODEs

While the first-principle models presented in subsection 3.1 provide a foundational understanding of dynamics, they often struggle to perfectly capture the intricate and non-linear behaviors present in high-performance racing scenarios. By leveraging the flexibility and adaptability of Neural Ordinary Differential Equations (NODEs), we can approximate the complex dynamics of autonomous racing vehicles more accurately, leading to improved trajectory planning, control, and optimization.

Compared to other neural-network-based system identification methods such as Neural State-Space Models (NSSMs), NODEs often achieve better performance [Rah+22] by learning the continuous-time dynamics instead of the discrete-time and therefore adding more inherent stability in the learning process, at the expense of more complex computational graph arising from the integration process with Runge-Kutta schemes. What's more, in our case, the system's nature made it easier to incorporate prior physical knowledge of the system in the NODE to obtain a *graybox* model.

Concretely, we consider the 6 DoF state $\tilde{x} = (X, Y, \varphi, v_x, v_y, r)$, use the same dynamics as in equation (2) for (X, Y, φ) and focus on learning the dynamics of (v_x, v_y, r) with a feed-forward neural network

$$\nu_\theta : \begin{array}{ccc} \mathbb{R}^5 & \to & \mathbb{R}^3 \\ (v_x, v_y, r, T, \delta) & \mapsto & (\dot{v}_x, \dot{v}_y, \dot{r}) \end{array}$$

so that the final dynamics read

$$\dot{\tilde{x}} = \begin{pmatrix} v_x \cos(\varphi) - v_y \sin(\varphi) \\ v_x \sin(\varphi) + v_y \cos(\varphi) \\ r \\ \nu_\theta(v_x, v_y, r, T, \delta) \end{pmatrix}$$

These simplifications allow the model to leverage the natural translational and rotational invariance of the car dynamics.

We chose a very simple architecture for ν_θ: a batch-norm layer (to avoid the need to normalize the input data and output data) followed by three dense layers with 128 neurons each and Leaky-ReLU activation functions. We further initialized every dense layer using the Xavier-uniform. This Multi-Layer-Perceptron (MLP) structure makes our architecture a good candidate to learn the car dynamics, since MLPs are known to be universal approximators [HSW89].

The training phase was based on a dataset of the form

$$\widetilde{\mathcal{D}} = \left\{ (\tilde{x}_0^i, \tilde{u}_{0:N_f-1}^i, \tilde{x}_{1:N_f}^i) \right\}_{i=1,\dots,I} \tag{5}$$

and was formulated as follows:

$$\begin{aligned} \min_\theta \quad & \frac{1}{IN_f} \sum_{i=1}^{I} \sum_{k=1}^{N_f} \|\tilde{x}_k^i - \tilde{x}_k^{\text{pred},i}\|_P^2 \\ \text{s.t.} \quad & \tilde{x}_{1:N_f}^{\text{pred},i} = \tilde{f}_{N_f}(\tilde{x}_0^i, \tilde{u}_{0:N_f-1}^i),\ i = 1, \dots, I \end{aligned} \tag{6}$$

where $P = \text{diag}(q_{XY}, q_{XY}, q_\varphi, q_{v_x}, q_{v_y}, q_r)$ is a diagonal weighting matrix. We took these weights all equal to 1 except $q_\varphi = 100$.

3.3 Dataset creation

A time-series dataset was collected at 20Hz (the frequency of the existing NMPC controller and therefore the target frequency for the DPC controller and the system model) in the FSDS

simulator by manually controlling the car using a video-game controller. The data form described in (5) was then obtained by sliding a window of appropriate size along the time-series. The experimental setup was installed at the EPFL Racing Team's booth during the EPFL Open days on April 29th and 30th, 2023 and publicly available for any visitor to try. Crowd-sourced data collection allowed us to collect a high volume of data (119530 observations in the training dataset and 18525 observations in the testing dataset) covering a wide range of operating scenarios, as shown in figure 7.

The only pre-processing applied to this dataset was performed on the yaw values φ that were always in the interval $(-\pi, \pi]$ and therefore presented some discontinuities points when going beyond those bounds. To make the signal continuous, we simply detected the jumps and translated the values by an appropriate multiple 2π .

3.4 Numerical results

The optimization problem (6) was implemented in PyTorch and executed on a Linux desktop computer with an Nvidia Quadro RTX 4000 graphics card. Using the AdamW optimizer for 500 epochs with a constant learning rate of 5e-4 and a weight decay factor of 5e-2, we trained four variants of the network described in subsection 3.2. We varied the horizon size $N_f \in \{1, 5, 10, 20\}$ to verify if increasing it could add stabilization to the training process by avoiding relatively small errors committed in one time step accumulating over a longer horizon of 40 time steps. Figure 8 shows that the mean square error loss is actually increasing when increasing the training horizon size. However, if we test all models on the test set with a 40 horizon size and compute the committed L^2 errors on each variable (position, yaw, velocities and yaw rate) then we can see that the model performance actually improve when increasing the training horizon size, as depicted in table 1.

We further compare in table 1, figure 9 and figure 10 the trained models with the hand-tuned 4 DoF kinematic bicycle model (referred to as Kin4) that was used in the existing NMPC controller.

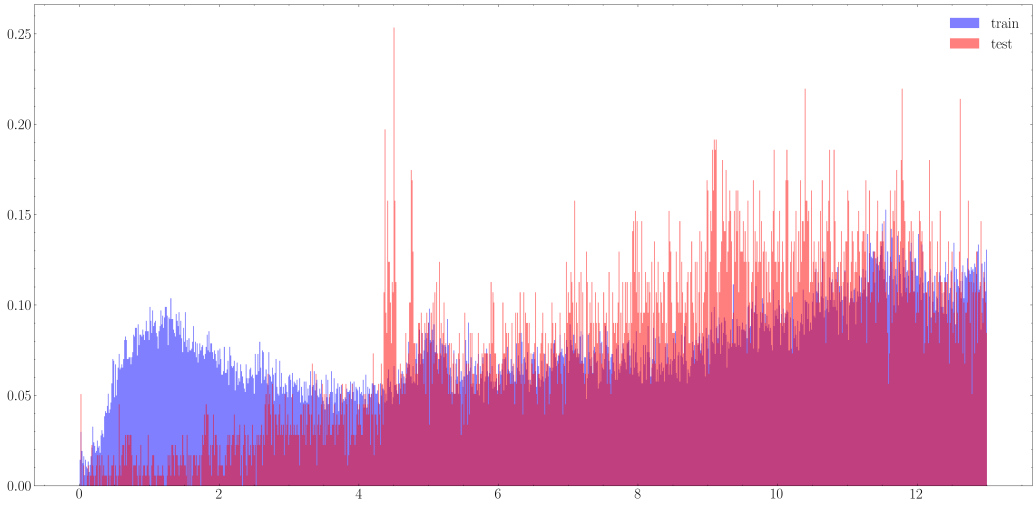
All presented models seem to have roughly equivalent performance, and offer all substantially better open loop predictions than the Kin4 model, as illustrated in 10. In the rest of the project, we are going to use the $N_f = 10$ variant that we will call NeuralDyn6. Its quantitative error improvement is reported in table 2.

	Kin4		NODE with Nf=1		NODE with Nf=5		NODE with Nf=10		NODE with Nf=20	
	mean	std	mean	std	mean	std	mean	std	mean	std
XY	7.3967	7.8335	0.6877	1.7896	0.6205	1.6938	0.5944	1.5599	0.5718	0.7547
φ	1.1875	1.2385	0.0713	0.1840	0.0571	0.1702	0.0544	0.1532	0.0496	0.1405
v_x	0.4330	0.6145	0.1844	0.3425	0.1696	0.3263	0.1632	0.3048	0.1835	0.2965
v_y	N/A	N/A	0.0718	0.0944	0.0714	0.0932	0.0756	0.0941	0.0845	0.1059
r	N/A	N/A	0.2873	0.7396	0.2646	0.7384	0.2619	0.7375	0.2666	0.7547

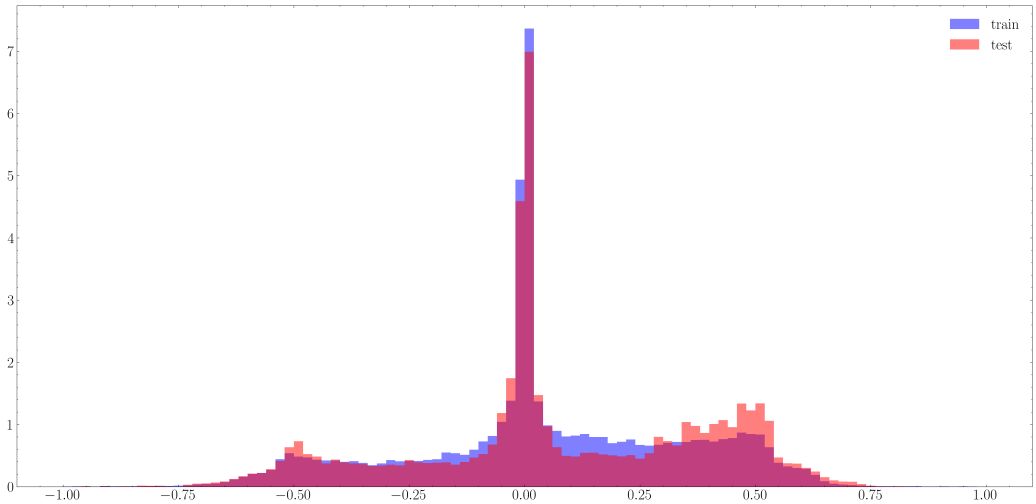
Table 1: L^2 errors distributions of Kin4 and NODE models

Variable	XY	φ	v_x
Improvement	91.96%	95.42%	62.31%

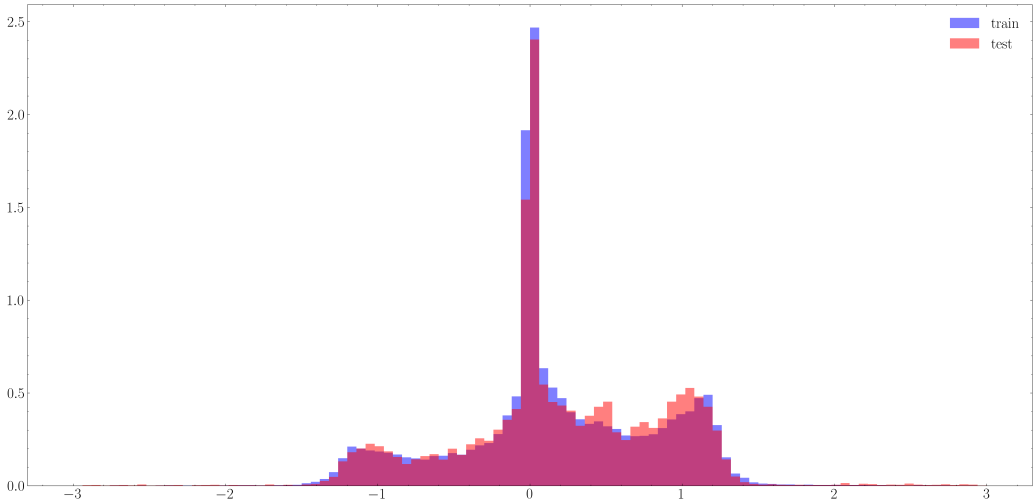
Table 2: Improvements in mean error of NODE model with $N_f = 10$ over Kin4



(a) v_x



(b) v_y



(c) r

Figure 7: Distribution of v_x, v_y and r in system identification time-series dataset

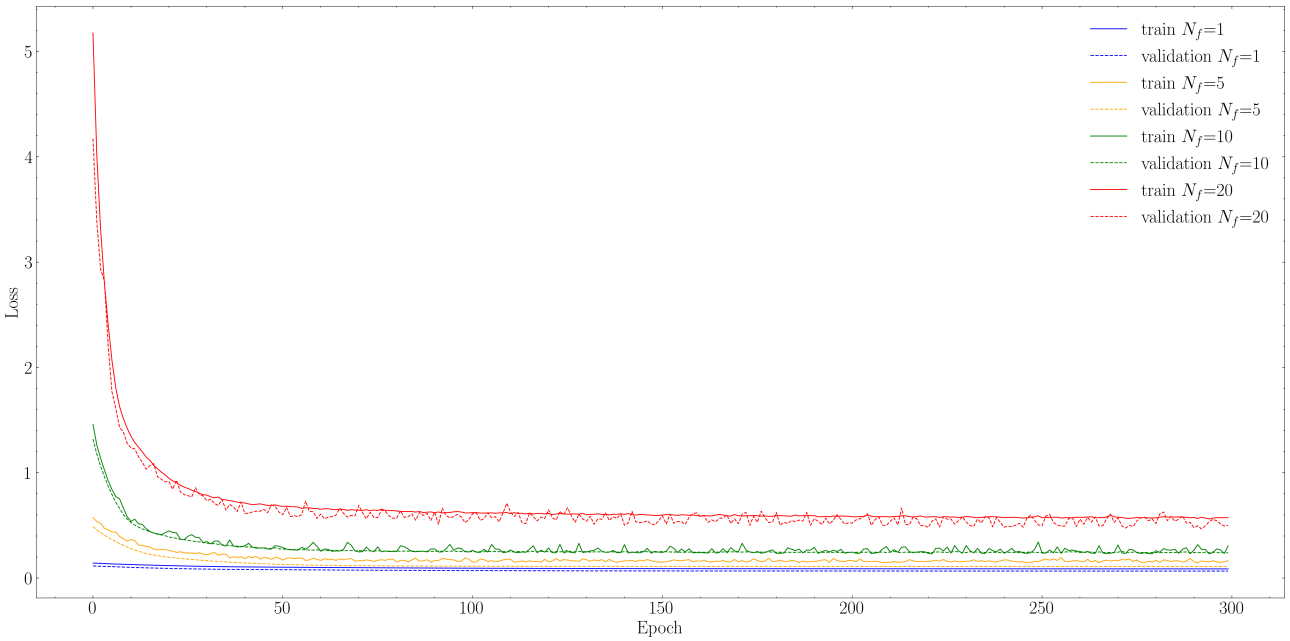
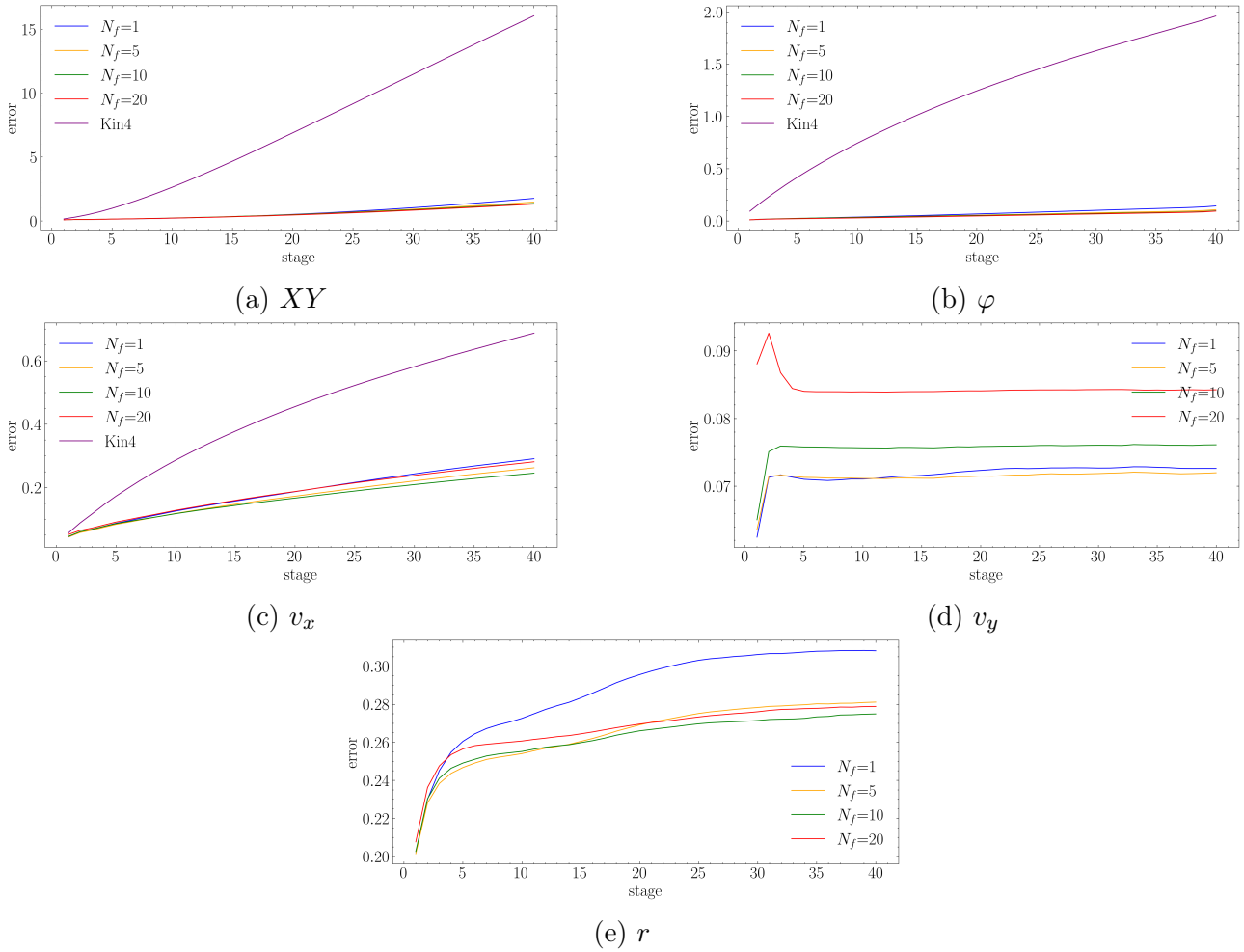


Figure 8: Training and validation losses in system identification training for $N_f \in \{1, 5, 10, 20\}$



(a) XY

(b) φ

(c) v_x

(d) v_y

(e) r

Figure 9: Average L^2 errors of Kin4 and NODE models by stage and variable

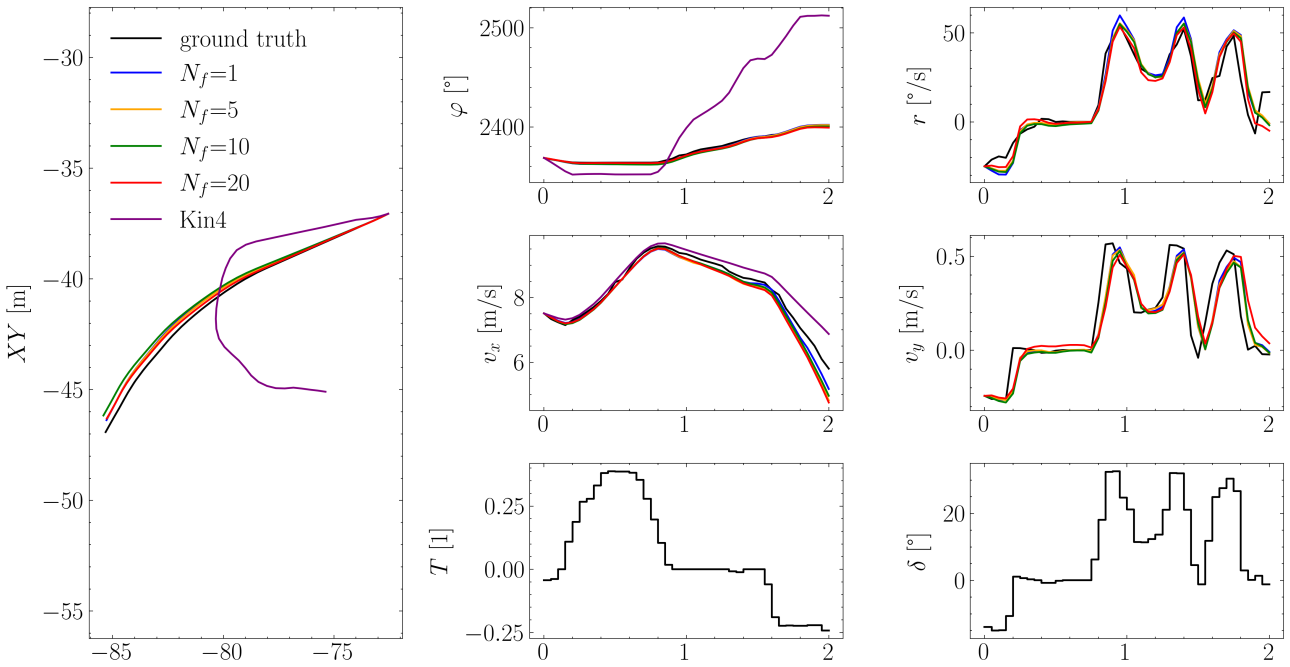


Figure 10: Open loop trajectories of Kin4 and NODE models

4 Control

In this section, we are going to deal with the second objective of our project: devising a control scheme that approximated well the existing NMPC scheme while having a lighter computational and memory footprint (because of the absence of online optimization).

We first formally introduce in subsection 4.1 the formulation of the existing controller, then introduce the Differentiable Predictive Control (DPC) paradigm in section 4.2, the dataset creation in subsection 4.3, and finally our experimental results in subsection 4.4.

4.1 Original NMPC formulation

We designed a path-tracking nonlinear MPC controller that enforces constraints and penalties on the control input rates. In our case, the traction command has virtually no rate limit, so we only consider penalties and constraints on the steering increments $d\delta$ and *lifted* the state by concatenating the last steering input. Concretely, we are going to use the transformed state $x_k = (\tilde{x}_k, \delta_{k-1}) = (X_k, Y_k, \varphi_k, v_{x,k}, v_{y,k}, r_k, \delta_{k-1})$ and the transformed control input $u_k = (T_k, d\delta_k)$ so that the discrete-time dynamics described in the last section read

$$x_{k+1} = f_1(x_k, u_k) = \begin{pmatrix} \tilde{f}_1(X_k, Y_k, \varphi_k, v_{x,k}, v_{y,k}, r_k, T_k, \delta_{k-1} + d\delta_k) \\ \delta_{k-1} + d\delta_k \end{pmatrix}$$

The NMPC controller will then take as input the current state $\hat{x}_0$ and a sequence of reference positions, orientations and velocities $\left\{(X_k^{\text{ref}}, Y_k^{\text{ref}}, \varphi_k^{\text{ref}}, v_{x,k}^{\text{ref}})\right\}_{k=0\ldots,N_f}$ obtained by the motion-planning procedure described in subsection 2.2.3 and formulate the following optimal control

problem in the so-called *multiple shooting* form:

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{u}} \quad & \sum_{k=0}^{N_f-1} \underbrace{\|x_k - x_k^{\text{ref}}\|_Q^2 + \|u_k\|_R^2}_{=:l(x_k,u_k)} + \underbrace{\|x_{N_f} - x_{N_f}^{\text{ref}}\|_{Q_{N_f}}^2}_{=:F(x_{N_f})} \\
\text{s.t.} \quad & x_0 = \hat{x}_0 \\
& x_{k+1} = f_1(x_k, u_k),\ k = 0, \dots, N_f - 1 \\
& \underline{x} \le x_k \le \bar{x}, k = 0, \dots, N_f \\
& \underline{u} \le u_k \le \bar{u}, k = 0, \dots, N_f - 1
\end{aligned} \tag{7}$$

where

$$\begin{gathered}
x_k^{\text{ref}} = (X_k^{\text{ref}}, Y_k^{\text{ref}}, \varphi_k^{\text{ref}}, v_{x,k}^{\text{ref}}, 0, 0, 0),\ u_k^{\text{ref}} = (0, 0), \\
\underline{x} = (-\infty, -\infty, -\infty, 0.0, -\infty, -\infty, -\delta_{\max}),\ \bar{x} = (\infty, \infty, \infty, v_{x,\max}\infty, \infty, \delta_{\max}), \\
\underline{u} = (-1, -d\delta_{\max}),\ \bar{u} = (1, d\delta_{\max}) \\
Q = \text{diag}(q_{XY}, q_{XY}, q_\varphi, q_{v_x}, q_{v_y}, q_r, q_\delta),\ R = \text{diag}(q_T, q_{d\delta}),\ Q_{N_f} = 10Q
\end{gathered}$$

This approximate controller does not enforce any track constraints and solely relies on the assumption that the reference trajectory is feasible and will be tracked accurately. Track constraints are difficult to properly formulate in Cartesian coordinates and often are expressed after transformation in the Frenet reference frame, as demonstrated in [Klo+20], [Rei+22]. These alternative methods are currently investigated at the EPFL Racing Team, but were not successfully implemented yet.

4.2 Differentiable Predictive Control

The idea behind Differentiable Predictive Control (DPC) [DTV22],[Drg+22a],[Drg+22b] is to train a feed-forward neural network

$$\pi_\vartheta : \begin{array}{ccc} \mathbb{R}^6 \times \mathbb{R}^{(N_f+1)\times 4} & \to & \mathbb{R}^{N_f \times 2} \\ x_0, x_{0:N_f}^{\text{ref}} & \mapsto & u_{0:N_f-1} \end{array}$$

to be our state-feedback control policy describing the solution of the parametric optimization problem (7) (parametrized by the initial state and the tracked reference), similarly to Explicit MPC [TJB03]. However, instead of solving the parametric optimization problem directly, DPC only solves a stochastic approximation of it. It leverages a dataset of the form

$$\mathcal{D} = \left\{ (x_0^j, x_{0:N_f}^{\text{ref},j}) \right\}_{j=1,\dots,J}, \tag{8}$$

and solves the optimization problem

$$\min_\vartheta \quad \frac{1}{JN_f} \sum_{j=1}^{J} \sum_{k=1}^{N_f} l(x_k^j, u_k^j) + p_{S_x}(x_k^j, \underline{x}, \bar{x}) + p_{S_u}(u_k^j, \underline{u}, \bar{u}) \ + \ F(x_{N_f}^j) + p_{S_x}(x_{N_f}^j, \underline{x}, \bar{x}) \tag{9}$$

$$\text{s.t.} \quad \left.\begin{array}{ll} u_{0:N_f-1}^j = & \pi_\vartheta(x_0^j, x_{0:N_f}^{\text{ref},j}) \\ x_{0:N_f}^{\text{pred},j} = & f_{N_f}(x_0^i, u_{0:N_f-1}^j) \end{array}\right\} \ j = 1, \dots, J$$

with $S_x \in \mathbb{R}^{7\times7}, S_u \in \mathbb{R}^{2\times2}$ some diagonal weight matrices. This corresponds to a *single-shooting* formulation of (7) with constraint violation penalties:

$$p_{S_x}(x_k^j, \underline{x}, \bar{x}) = \| \operatorname{ReLU}(\underline{x} - x_k^j)\|_{S_x}^2 + \| \operatorname{ReLU}(x_k^j - \bar{x})\|_{S_x}^2,$$
$$p_{S_u}(u_k^j, \underline{u}, \bar{u}) = \| \operatorname{ReLU}(\underline{u} - u_k^j)\|_{S_u}^2 + \| \operatorname{ReLU}(u_k^j - \bar{u})\|_{S_u}^2$$

These penalties are one of the most important advantages of DPC, which entail stochastic guarantees on the constraint satisfaction [DTV22].

We formulate the control policy network with the exact same hidden layers as the system model (see subsection 3.2) and an additional appropriately scaled tanh layer to explicitly enforce constraints on the network outputs (corresponding to the throttle and steering increments). Exactly as for the system model, we would like to preserve the translational and rotation invariance properties of the control policy (with respect to **all** its inputs, i.e. both current state and reference). We therefore do not provide as is the values $x_0, x_{0:N_f}^{\text{ref}}$ to the network, but eliminate the current pose and provide the reference poses expressed in the local car reference frame, therefore making the input dimension $3 + 4(N_f + 1)$ instead of $6 + 4(N_f + 1)$. This allows to further ensure that the network's inputs will always lie in reasonable ranges.

We chose to use the exact same weight matrices Q, R, Q_{N_f} as those hand-tuned for our MPC whose value are reported in table 3, and slack penalties $S_x = 100I_7, S_u = 100I_2$.

Variable	q_{XY}	q_φ	q_{v_x}	q_{v_y}	q_r	q_T	q_δ	$q_{d\delta}$
Value	100	200	10	0	0	100	200	100

Table 3: Cost weights in MPC and DPC

4.3 Dataset creation

The dataset in the form (8) was collected by running our existing MPC controller in closed loop in FSDS on 4 typical FS tracks and collecting sequences of car states and references provided to the controller. This step was taken to ensure that the inputs accurately reflected the commonly encountered conditions. We collected in total 69094 training scenarios and 10158 test scenarios.

4.4 Numerical results

Exactly as in the previous section, we implemented the DPC training in PyTorch and trained it for 500 epochs with a learning rate of 5e-4 and a weight decay factor of 1e-2. We do not report the full training losses but only the final values of the unscaled constraint violation penalty losses in table 4, where we can see that the constraints are indeed almost always satisfied.

Constraint	δ lower bound	δ upper bound	v_x lower bound	v_x upper bound
Violation	1.03e-5	2.14e-5	0	0

Table 4: Mean constraint violation committed by DPC on the validation set

We then sample a random scenario in the test set and computed the open-loop predictions of our DPC controller and of the MPC controller, with either the Kin4 or NeuralDyn6 models. The resulting open-loop predictions can be found in 11. The first thing we notice is that although the control inputs computed by the MPC controller seem to follow the provided reference relatively well when predicting with the Kin4 model, it is not the case anymore when using

the NeuralDyn6 model. This can be explained by the fact that the MPC scheme 'overfitted' in some sense the bad system model. We see that from this perspective, the trained DPC policy shows a much more promising open-loop behavior.

Nevertheless, when applying DPC in closed-loop scenarios, the irregularity in the computed control input sequence, coupled with the imperfections of the NeuralDyn6 model, results in diverging system behavior. The closed-loop trajectory and corresponding control inputs on the FSDS test track are depicted in figure 12. Initially, the trajectory is guided by the MPC controller, but at a certain point, the DPC controller takes over. It becomes apparent that the executed actions lack coherence, with instances of unwarranted increases in steering and throttle commands. As a consequence, the vehicle deviates from the desired trajectory, leading to the car inadvertently leaving the track.

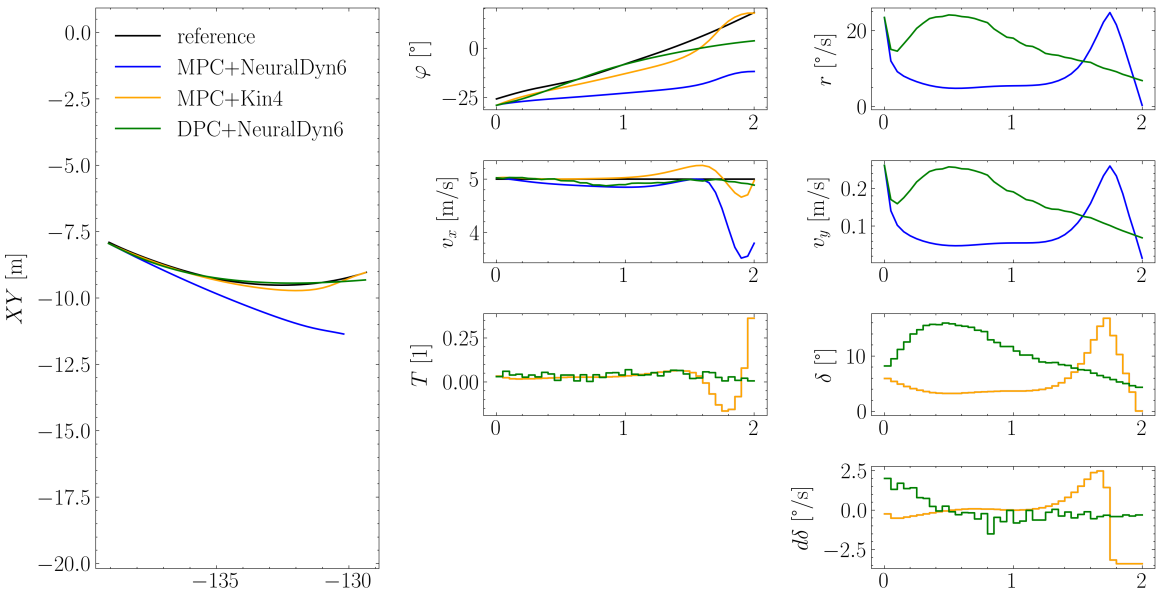


Figure 11: Open-loop predictions of different pairs of controllers and system models. *Remark*: in the subplots T, δ and $d\delta$, the blue and yellow curves are overlapping.

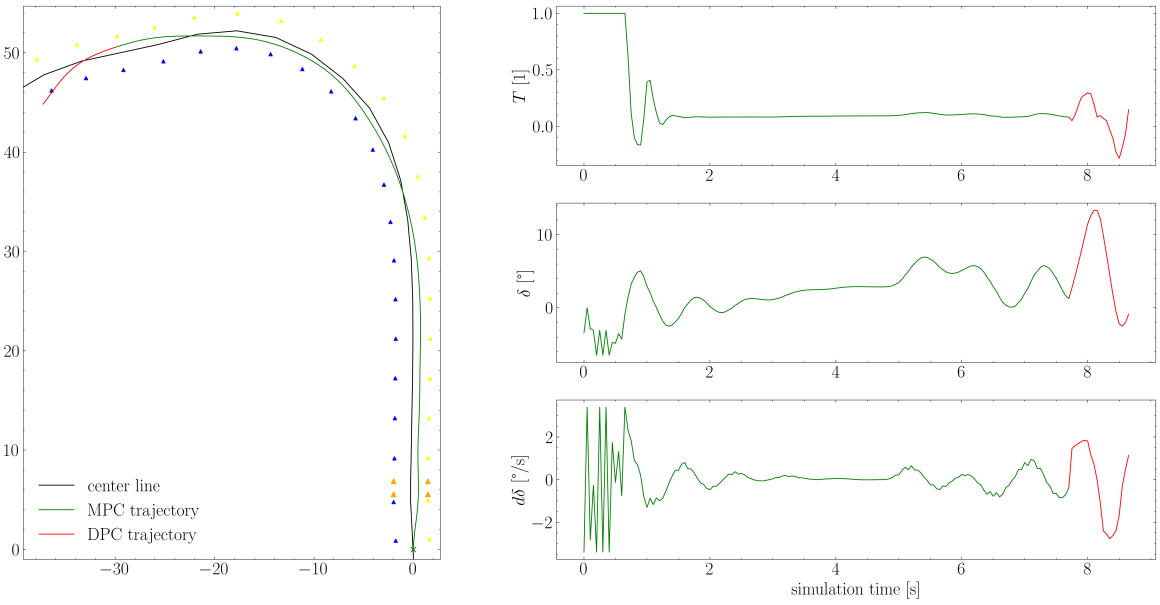


Figure 12: Closed-loop trajectories of MPC followed by DPC in FSDS

5 Conclusion and outlook

In this project, our objective was to develop a robust system model and control policy for autonomous racing, with a particular focus on the utilization of deep neural networks. We have achieved significant progress in the first goal, as evidenced by the improved robustness and accuracy of the NeuralDyn6 model. However, in the realm of control, we encountered challenges with the deployment of DPC in closed-loop scenarios. The inconsistent actions taken by the DPC controller, such as unwarranted increases in steering and throttle commands, ultimately resulted in the vehicle deviating from the intended trajectory.

To further improve the control performance, several modifications could be explored. One potential approach is the utilization of data augmentation techniques such as translation augmentation could introduce additional variations in the training data, enabling the DPC controller to handle a wider range of scenarios and enhance its adaptability, at the cost of a dataset size bigger by orders of magnitude. Clustering techniques could be then be used to reduce redundancies in the dataset.

We could also incorporate other auxiliary objectives in DPC alongside the MPC loss. For instance, we could also add a regression loss to combine DPC with imitation learning procedures such as DAgger [RGB10]. This could even be combined with the usage of a safety filter [WZ19], in particular the recently proposed safety filter [Tea+21], to provide a safe training framework that could even be deployed in real life conditions on actual cars.

6 Bibliography

References

[Drg+22a] Jan Drgona et al. *Learning Stochastic Parametric Differentiable Predictive Control Policies*. 2022. arXiv: 2203.01447 [eess.SY].

[Drg+22b] Ján Drgoňa et al. "Differentiable predictive control : Deep learning alternative to explicit model predictive control for unknown nonlinear systems". In: *Journal of Process Control* 116 (2022), pp. 80–92. ISSN: 0959-1524.

[DTV22] Jan Drgona, Aaron Tuor, and Draguna Vrabie. *Learning Constrained Adaptive Differentiable Predictive Control Policies With Guarantees*. 2022. arXiv: 2004.11184 [eess.SY].

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[Klo+20] Daniel Kloeser et al. "NMPC for Racing Using a Singularity-Free Path-Parametric Model with Obstacle Avoidance". In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 14324–14329. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.1376. URL: https://www.sciencedirect.com/science/article/pii/S2405896320317845.

[LDM14] Alexander Liniger, Alexander Domahidi, and Manfred Morari. "Optimization-based autonomous racing of 1:43 scale RC cars". In: *Optimal Control Applications and Methods* 36.5 (2014), pp. 628–647. DOI: 10.1002/oca.2123. URL: https://doi.org/10.1002%2Foca.2123.

[Pac06] H.B. Pacejka. *Tire and Vehicle Dynamics*. Jan. 2006. DOI: 10.1016/B978-0-7506-6918-4.X5000-X.

[Rah+22] Aowabin Rahman et al. *Neural Ordinary Differential Equations for Nonlinear System Identification*. 2022. arXiv: 2203.00120 [cs.LG].

[Raj06] R Rajamani. *Vehicle Dynamics and Control*. Jan. 2006. ISBN: 0-387-26396-9. DOI: 10.1007/0-387-28823-6.

[Rei+22] Rudolf Reiter et al. *Frenet-Cartesian Model Representations for Automotive Obstacle Avoidance within Nonlinear MPC*. 2022. arXiv: 2212.13115 [eess.SY].

[RGB10] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "No-Regret Reductions for Imitation Learning and Structured Prediction". In: *CoRR* abs/1011.0686 (2010). arXiv: 1011.0686. URL: http://arxiv.org/abs/1011.0686.

[Sha+17] Shital Shah et al. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065. URL: https://arxiv.org/abs/1705.05065.

[Tea+21] Ben Tearle et al. *A predictive safety filter for learning-based racing control*. 2021. arXiv: 2102.11907 [eess.SY].

[TJB03] Petter Tøndel, Tor Arne Johansen, and Alberto Bemporad. "An algorithm for multi-parametric quadratic programming and explicit MPC solutions". In: *Automatica* 39.3 (2003), pp. 489–497. ISSN: 0005-1098. DOI: https://doi.org/10.1016/S0005-1098(02)00250-9. URL: https://www.sciencedirect.com/science/article/pii/S0005109802002509.

[Ver+14] Robin Verschueren et al. "Towards time-optimal race car driving using nonlinear MPC in real-time". In: *53rd IEEE Conference on Decision and Control*. 2014, pp. 2505–2510. DOI: 10.1109/CDC.2014.7039771.

[WBL22] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV].

[WZ19] Kim P. Wabersich and Melanie N. Zeilinger. *Linear model predictive safety certification for learning-based control*. 2019. arXiv: 1803.08552 [cs.SY].