ZipMPC: Compressed Context-Dependent MPC Cost via Imitation Learning

Rahel Rickenbach^{*,1} `rrahel@ethz.ch` **Alan A. Lahoud**^{*,2} `alan.lahoud@oru.se` **Erik Schaffernicht**^{3} `erik.schaffernicht@thws.de`

Melanie N. Zeilinger^{†,1} `mzeilinger@ethz.ch` **Johannes A. Stork**^{†,2} `johannesandreas.stork@oru.se`

Abstract:
The computational burden of model predictive control (MPC) limits its application on real-time systems, such as robots, and often requires the use of short prediction horizons. This not only affects the control performance, but also increases the difficulty of designing MPC cost functions that reflect the desired long-term objective. This paper proposes ZipMPC, a method that imitates a long-horizon MPC behaviour by learning a compressed and context-dependent cost function for a short-horizon MPC. It improves performance over alternative methods, such as approximate explicit MPC and automatic cost parameter tuning, in particular in terms of i) optimizing the long-term objective; ii) maintaining computational costs comparable to a short-horizon MPC; iii) ensuring constraint satisfaction; and iv) generalizing control behaviour to environments not observed during training. For this purpose, ZipMPC leverages the concept of differentiable MPC with neural networks to propagate gradients of the imitation loss through the MPC optimization. We validate our proposed method in simulation and real-world experiments on autonomous racing. ZipMPC consistently completes laps faster than selected baselines, achieving lap times close to the long-horizon MPC baseline. In challenging scenarios where the short-horizon MPC baseline fails to complete a lap, ZipMPC is able to do so. In particular, these performance gains are also observed on tracks unseen during training. Further information[3] at zipmpc.github.io/.

Keywords: Imitation Learning, Model Predictive Control, Context-Dependent Control

1 Introduction

Model predictive controllers (MPCs) rely on optimization-based control actions that minimize an MPC cost while respecting environmental and system-specific constraints. This allows for their successful deployment in a range of safety-critical control applications, such as robot locomotion [1, 2, 3, 4] and manipulation [5, 6], autonomous racing [7, 8, 9], or biomedical robotics [10, 11, 12] and applications [13, 14]. However, to make MPC computationally tractable, its prediction horizon is commonly chosen significantly smaller than the horizon required for task completion, resulting in an information gap. Therefore, MPC applications face two challenges; (i) the selection of a tractable MPC prediction horizon length, i.e., the time frame considered during optimization, (ii) the design of an MPC cost function that accurately reflects the desired performance objective. Applying MPC with a long prediction horizon, good performance can commonly be achieved with an intuitively

[1]Institute for Dynamic Systems and Control, ETH Zurich, Switzerland. [2]Center for Applied Autonomous Sensor Systems (AASS), Örebro University, Sweden. [3]Technology Transfer Center Kitzingen, Technical University of Applied Sciences Würzburg-Schweinfurt. [*]These authors contributed equally (shared first authorship). [†]These authors contributed equally (shared last authorship). [3] The datasets generated and/or analysed are available in the eth research collection repository, https://doi.org/10.3929/ethz-c-000783174.

and manually designed MPC cost, as the control policy is optimized considering vast knowledge of future environmental features, such as upcoming curvature in autonomous racing applications or glucose intake in insulinic pumps. However, given that the computation time increases with the number of optimization variables, i.e., the horizon length [15, 16], applications with long-horizon MPCs are often not real-time feasible. Therefore, MPC with a short prediction horizon length is generally more common. Short-horizon MPC formulations, on the other hand, render the design of the cost function challenging and often suffer from reduced performance due to the limited incorporation of future information. Available approaches to address these challenges include approximate explicit MPC (eMPC), which learns efficient closed-form policies to imitate MPC behaviour, learning the cost-to-go, which learns a cost term to represent the information gap until task completion, as well as automatic cost parameter tuning via inverse optimal control (IOC) or Bayesian optimization (BO). While each of them offers individual advantages, from constraint satisfaction and generalization capabilities to computational efficiency, the goal of this paper is to propose a framework that combines these strengths.

Contribution. We propose ZipMPC, an algorithm designed to combine the computational advantages of short MPC horizons with the richer contextual information associated with longer horizons. Specifically, as depicted in Figure 1 for racing applications, ZipMPC employs a learnable function, e.g., using neural networks (NN), to encode long-horizon contextual information, such as track curvature, into a shorter horizon MPC cost. Leveraging differentiable MPC techniques, ZipMPC enables offline imitation learning of optimal long-horizon MPC solutions by propagating informative gradients through the MPC optimization procedure. By utilizing the computational efficiency of NN inference, our method places itself between eMPC and autonomous parameter tuning of MPC cost parameters by combining their strengths, i.e., it reduces the computational time, while considering the dynamics to ensure constraint satisfaction. Due to the fact that we learn a cost function instead of a behavioural cloning policy, ZipMPC demonstrates a generalization capability, leading to improved performances in environments both seen and unseen during training. We validate ZipMPC in simulation and on real-world (hardware) autonomous racing scenarios.

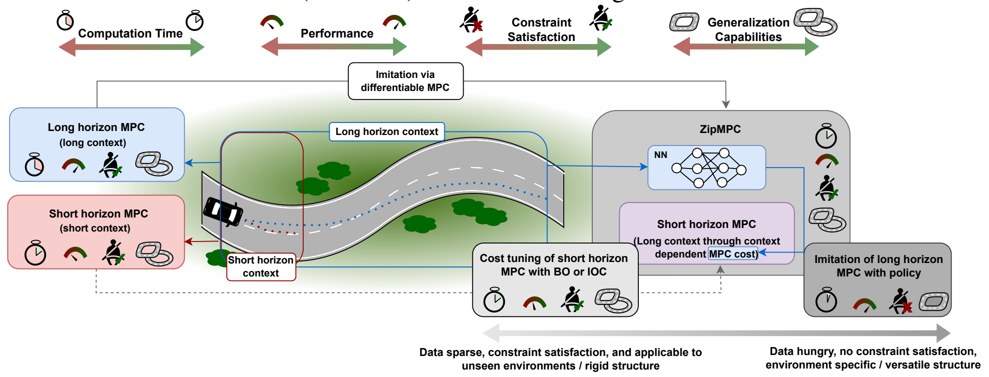

Computation Time
Performance
Constraint Satisfaction
Generalization Capabilities
Imitation via differentiable MPC
Long horizon MPC (long context)
Long horizon context
ZipMPC
NN
Short horizon MPC (short context)
Short horizon MPC (Long context through context dependent MPC cost)
Short horizon context
Cost tuning of short horizon MPC with BO or IOC
Imitation of long horizon MPC with policy
Data sparse, constraint satisfaction, and applicable to unseen environments / rigid structure
Data hungry, no constraint satisfaction, environment specific / versatile structure

Figure 1: ZipMPC imitates long-horizon MPC behaviour by learning a compressed and context-dependent cost for a short-horizon MPC. It thereby combines the faster inference times of eMPC with the generalization and constraint recognition capabilities of BO and IOC.

Related Work. Automatic tuning of MPC cost parameters is generally explored using BO to minimize the unknown mapping of MPC cost parameter values to a desired performance measurement [17], employing optimality conditions to estimate the unknown objective vector from available optimal demonstrations via IOC [18, 19], or employing RL techniques [20, 21, 22]. While increasing the performance when applied to long-horizon MPC, they do not alleviate the computational demand. When applied to short-horizon MPC, current methods are not designed to include additional information to overcome the information gap and fall short in achieving a satisfying imitation behaviour. An exception describes contextual BO [23], which, however, does not scale well with context dimension. To reduce computational complexity, several works have explored to reduce the horizon and to learn the missing information until task completion, commonly referred to as learning the cost-to-go [24, 25, 26, 27], as well as replacing MPCs with trained NNs (eMPC) due to their quick inference.

For instance, [28, 29, 30] use deep learning models to approximate MPC solutions through offline training, aiming to replicate MPC control policies in real-time. However, these approaches struggle with generalization to unseen environments during training. Furthermore, as NNs naturally fail to output structured predictions [31], e.g., predictions that satisfy MPC hard constraints, eMPC commonly does not offer constraint satisfaction properties. Recently, there has been growing interest in combining NNs with constrained optimization [32]. A key direction uses mathematical solvers as layers within deep learning models to enforce hard constraints in the outputs. These methods often involve differentiating solver outputs with respect to their inputs or parameters [33, 34]. Specifically, methods for the integration of MPCs as a differentiable block within deep learning frameworks have been developed by [35], which propose adding ReLU-based penalty terms in the loss function to penalize constraint violations, or [36], which use iterative Linear Quadratic Regulator (iLQR) as an approximation to MPC, and differentiates through the iLQR's fixed point by leveraging gradients through KKT conditions. Differentiable MPCs are, for example, currently employed for the imitation of human behaviour [37]. With our proposed framework, we leverage them for reducing computational demands.

2 MPC Preliminaries

In this section, we introduce the MPC formulation serving as the presented framework's basis. Let $x(k+1) = f(x(k), u(k), \zeta(k))$ represent some context-dependent discrete-time dynamics at time step k, where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ and $\zeta \in \mathbb{R}^a$ denote the state, control-input and environmental context variable, respectively. With N indicating the prediction horizon, a general MPC controller solves the following problem at time step k given the current state $x(k)$:

$$\begin{aligned} [X^*_{k,N}, U^*_{k,N}] := MPC_N(C_N, x(k)) = \underset{\{x_i\}_0^{N+1}, \{u_i\}_0^N}{\arg\min} \sum_{i=0}^{N} \ell(x_i, u_i, c_i) \\ \text{s.t.} \quad x_0 = x(k), \quad x_{i+1} = f(x_i, u_i, \zeta_i), \quad x_i \in \mathcal{X}, \quad u_i \in \mathcal{U}, \quad i = 0, \dots, N. \end{aligned} \tag{1}$$

The function ℓ defines the stage cost with parameters $c_i \in \mathbb{R}^b$, summarized in $C_N := \{c_i\}_0^N$. Furthermore, state and control-input constraints are indicated with $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^m$, respectively. Solving Eq. 1 returns an optimal state and control-input trajectory for all prediction steps of horizon N, denoted as $X^*_{k,N} := \{x^*_i\}_0^{N+1}$ and $U^*_{k,N} := \{u^*_i\}_0^N$, where $x^*_i \in \mathcal{X}$ and $u^*_i \in \mathcal{U}$ indicate the optimal state and input at prediction step i. Only the first optimal control-input u^*_0 is applied before the optimization is reinitialized and resolved, referred to as receding horizon control. To address theoretical and practical issues potentially resulting from the finite horizon implementation, such as missing *convergence* and *recursive feasibility* guarantees [38, 39], an additional cost term and constraint set on x_{N+1}, known as terminal ingredients, are often used to imitate infinite horizon behaviour. However, given that the design of terminal ingredients can be nontrivial, theoretical investigations on how to overcome their necessity for long-horizon MPCs have found interest in the recent literature [40, 41, 42]. This further simplifies the MPC cost design with long-horizon length and motivates the use of an MPC formulation without terminal ingredients in this work. Note, however, that the inclusion of a terminal cost would be straight forward.

3 ZipMPC: Compressed MPC Cost via Imitation Learning

In this section, we first define the underlying problem formulation, and then introduce ZipMPC, our method to learn an MPC cost via imitation learning by efficiently compressing long-horizon context information into a short-horizon MPC cost, followed by its detailed algorithm.

3.1 Problem Formulation

Motivated by the improved performance of an intuitively tuned MPC with increasing horizon length (see Appendix C.1), we formulate the following problem. Let N_L and $N_S < N_L$, both $\in \mathbb{Z}$, represent a long enough (but expensive) and a short (but computationally efficient) prediction horizon length, respectively. We denote $[X^*_{k,N_L}, U^*_{k,N_L}] := MPC_{N_L}(C^M_{N_L}, x(k))$ as the optimal state and control-input variables given a long-horizon N_L with well tuned cost parameters $C^M_{N_L}$ and current state $x(k)$. Similarly, we denote $[X^\theta_{k,N_S}, U^\theta_{k,N_S}] := MPC_{N_S}(C^\theta_{N_S}, x(k))$ as the optimal state and control-input variables given a short-horizon N_S with learned cost parameters $C^\theta_{N_S}$, where θ represents learnable parameters (e.g., NN weights). We aim to find θ^* that minimizes an imitation loss $\mathcal{L}$ between the predicted reference trajectories of the long-horizon MPC, and the predicted trajectories of the short-horizon MPC with the inferred costs, i.e., we aim to imitate the behaviour of a well-tuned long-horizon MPC. The problem formulation is mathematically defined as

$$\theta^* := \arg\min_\theta \mathbb{E}_k[\mathcal{L}([X^\theta_{k,N_S}, U^\theta_{k,N_S}], [X^*_{k,N_L}, U^*_{k,N_L}])], \tag{2}$$

where k encodes information of initial states sampled from a set of feasible states, further detailed in the next section.

3.2 Learning framework

To overcome the information gap between the optimization procedures of MPC_{N_S} and MPC_{N_L}, we propose modeling the short-horizon MPC cost parameters $C^\theta_{N_S}$ as functions of the current state $x(k)$ and some contextual information sequence Z_{k,N_L} that ideally contains at least as much information as provided to MPC_{N_L}. Thus, we define a parametric nonlinear function h^θ, learned using NN weights θ, such that $C^\theta_{N_S} = h^\theta(x(k), Z_{k,N_L})$. The trajectory $[X^\theta_{k,N_S}, U^\theta_{k,N_S}]$ is then provided by the optimization procedure in $MPC^\theta_{N_S} := MPC_{N_S}(C^\theta_{N_S}, x(k))$, and the loss of Eq. 2 is computed against the reference trajectory $[X^*_{k,N_L}, U^*_{k,N_L}]$. Since trajectories from MPC_{N_L} and $MPC^\theta_{N_S}$ differ in horizon length, the imitation loss is computed using only the first N_S steps of both trajectories. This ensures that $MPC^\theta_{N_S}$ anticipates behaviour within these steps, which would be unaccounted for without explicitly incorporating the long-horizon context Z_{k,N_L} into our learning framework. A sampling method provides initial states $x(k)$ for both the parametric function h^θ and the MPCs, $MPC^\theta_{N_S}$ and MPC_{N_L}. Although various sampling methods could be used, a simple strategy is to apply uniform random sampling from a "feasible" region. We ensure that every considered $x(k)$ in the learning process is "feasible" by only keeping the ones that adhere to MPC_{N_L} feasibility. A high-level diagram of our learning approach is illustrated in Figure 2.

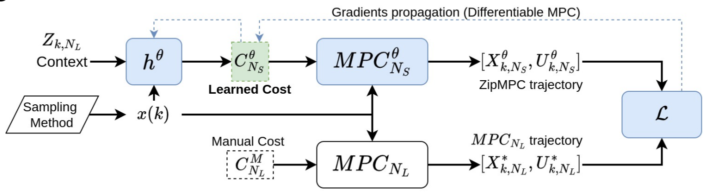


Figure 2: Simplified diagram of our learning approach. A NN parametrized by θ receives a sample state $x(k)$ and corresponding context Z_{k,N_L} as an input to predict cost parameters of $MPC^\theta_{N_S}$. The weights θ are then optimized to minimize an imitation loss computed between the first N_S steps of the trajectory yielded by MPC_{N_L} and the inferred trajectory of $MPC^\theta_{N_S}$.

In order to solve the optimization in Eq. 2 using gradient-based algorithms, we need to propagate gradients from the imitation loss, i.e., $\mathcal{L}([X^\theta_{k,N_S}, U^\theta_{k,N_S}], [X^*_{k,N_L}, U^*_{k,N_L}])]$, to the NN weights θ, which is formulated by the following equation:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial [X^\theta_{k,N_S}, U^\theta_{k,N_S}]} \cdot \frac{\partial [X^\theta_{k,N_S}, U^\theta_{k,N_S}]}{\partial C^\theta_{N_S}} \cdot \frac{\partial C^\theta_{N_S}}{\partial \theta}. \tag{3}$$

It is well known that the bottleneck of the gradient backpropagation lies in the second term of the right hand side of the above equation [43, 44, 45], since it requires to propagate the gradients through the optimization procedure, which in our case is the $MPC^\theta_{N_S}$ optimization. Therefore, we ensure the flow of informative gradients of the MPC solution with respect to the MPC cost parameters by leveraging differentiable MPC methods. In particular, we use the differentiable method of [36] as a tool, as it fits to general MPC cost formulations by extending the KKT-based differentiation [33] to handle non-convexities by iteratively solving a convex quadratic approximation using iterative Linear Quadratic Regulator.

Detailed Algorithm. The step-by-step process of ZipMPC within a gradient-based learning process is presented in Algorithm 1. It follows the diagram in Figure 2 with some additional details that we highlight as follows: i) We assume the availability of suitable manually designed cost parameters $c^M \in \mathbb{R}^b$; ii) c^M is extended and repeated across the prediction horizon (lines 1 and 2), resulting in $C^M_{N_L}$ and $C^M_{N_S}$, for the considered long and short-horizon MPC, respectively; iii) $C^M_{N_S}$ is used as a warm start for our learning algorithm. Therefore, our NN outputs the correction values $\Delta C^\theta_{N_S}$, which we add to the warm start $C^M_{N_S}$ to infer the learned cost parameters $C^\theta_{N_S}$ (lines 5 and 6); iv) Finally, we adopt the mean squared error (MSE) as an imitation loss between predicted and long-horizon states and control-input trajectories. In practice, we consider only the first $N_D \leq N_S$ steps in the imitation loss, i.e., MSE_{N_D} (see Appendix B.1 for more details.), and use Adam as the gradient-based optimizer to update the NN weights (lines 7 to 10); v) Effectively, algorithm 1 is executed in mini-batches, i.e., a batch of initial states is sampled, and the loss is computed using empirical risk minimization over the batch.

Algorithm 1 ZipMPC

```
Inputs:
Initial NN weights $\theta$ of a NN function $h$.
Context information $Z$.
Initial MPC manual cost parameters $c^M$.
Horizon lengths $N_S$ and $N_L$.
 1: $C^M_{N_S} \leftarrow$ EXPAND$(c^M, N_S)$
 2: $C^M_{N_L} \leftarrow$ EXPAND$(c^M, N_L)$
 3: for $i = 1$ to $N_{ITERATIONS}$ do
 4:     $x(k) \leftarrow$ SAMPLE()
 5:     $\Delta C^\theta_{N_S} \leftarrow h^\theta(x(k), Z_{k,N_L})$
 6:     $C^\theta_{N_S} \leftarrow C^M_{N_S} + \Delta C^\theta_{N_S}$
 7:     $[X^\theta_{k,N_S}, U^\theta_{k,N_S}] \leftarrow MPC^\theta_{N_S}(C^\theta_{N_S}, x(k))$
 8:     $[X^*_{k,N_L}, U^*_{k,N_L}] \leftarrow MPC_{N_L}(C^M_{N_L}, x(k))$
 9:     $\mathcal{L} \leftarrow \text{MSE}_{N_D}([X^*_{k,N_L}, U^*_{k,N_L}], [X^\theta_{k,N_S}, U^\theta_{k,N_S}])$
10:     $\theta \leftarrow$ Adam$(\theta, \mathcal{L})$
11: end for
```

4 Experiments

Autonomous racing poses a safety-critical application in robotics, requiring fast computation times of the applied control action. Furthermore, designing an MPC cost function that makes a car complete a lap in minimal time proves challenging for short-horizons, while convincing results have been shown with longer horizons (Appendix C.1). This makes it a compelling application to demonstrate the usefulness of our proposed framework. Therefore, we specify the used system dynamics f with two different models. The simpler *kinematic bicycle*, and the more complex *Pacejka* model [46]. Furthermore, representing the car state and race track in Frenet frame [47], we design the controller as a model predictive contouring controller (MPCC) [9] with a quadratic stage cost function ℓ and corresponding cost parameter vector $c_i = [q_i, p_i]$, where $q_i, p_i \in \mathbb{R}^{n+m}$. Accordingly, for later reference, we define $Q_N := \{q_i\}_0^N$ and $P_N := \{p_i\}_0^N$, as well as the corresponding parametric mapping functions h^θ_Q and h^θ_p. Furthermore, we chose the context variable to be a long-horizon sequence of track curvatures (for further details, see Appendix B.1). Details on the car models and control approach are provided in Appendix A.

Experiment Design and Baselines. We evaluate our approach conducting experiments in simulation and real-world. In Sec. 4.1, we assess the performance of ZipMPC by comparing it with BO and eMPC, inspired by the approaches in [17] and [48] and adapted to our framework. We also consider two baselines that are given by a long-horizon MPC (MPC_{N_L}) with horizon length N_L, serving as a high-performance but time-consuming baseline, and a short-horizon MPC (MPC_{N_S}) with horizon length N_S, both employing intuitively tuned cost parameters q^M and p^M across the horizon. Details on the choice of q^M and p^M are provided in Appendix A.4. To determine suitable values for the horizon lengths N_S and N_L, we evaluate lap times with horizon lengths (see Tables 9 and 10 in the Appendix) given the manually designed cost. While ZipMPC follows Algorithm 1 to learn a Δ-cost that is context and state dependent, the BO baseline obtains a constant Δ-cost using BO to minimize the imitation loss between MPC_{N_S} and MPC_{N_L}. The eMPC baseline employs a fully learning-based control strategy, where a NN is employed to clone the behaviour of MPC_{N_L}.

In Sec. 4.1, overall performances are measured based on trajectory imitation error and lap completion time, and we also demonstrate that our method generalizes well to unseen tracks. In Sec. 4.2, we analyze the role of the context in our approach and investigate the NN outputs. If not further specified, train and test track are chosen as illustrated in Figure 3. Finally, in Sec. 4.3 we test our proposed method on hardware.

4.1 Simulation Results

Imitation Error. We trained an MPC cost function using the *kinematic model* with various scenarios by varying the horizon lengths N_S and N_L. For each scenario, the learned cost parameters ($Q^{\theta}_{N_S}$ and $P^{\theta}_{N_S}$), conditioned on the sequence of track curvatures and useful variables of the initial state (for further details see Appendix B.1), were used to execute ZipMPC over a fixed set of 1000 validation initial states to obtain 1000 trajectories. Each trajectory was compared with the trajectory resulting from the reference method MPC_{N_L} (root mean squared error (RMSE) between the states plus control variables). The obtained results are presented in Table 1, showing an increased imitation capability of ZipMPC compared to the chosen baselines.

Table 1: Imitation loss (RMSE) between methods and MPC_{N_L} of the first 5 steps using the *kinematic model*. Mean and standard deviation are reported over a validation set of 1000 initial states.

N_S	N_L	MPC_{N_S}	BO	eMPC	ZipMPC
5	18	0.194 ± 0.004	0.089 ± 0.009	0.081 ± 0.007	$\mathbf{0.065 \pm 0.006}$
5	25	0.203 ± 0.004	0.150 ± 0.006	0.086 ± 0.009	$\mathbf{0.068 \pm 0.009}$
10	18	0.061 ± 0.005	0.061 ± 0.005	0.101 ± 0.007	$\mathbf{0.043 \pm 0.016}$
10	25	0.088 ± 0.008	0.086 ± 0.008	0.108 ± 0.007	$\mathbf{0.043 \pm 0.012}$

Lap Time. In Tables 2 and 3, we demonstrate that ZipMPC considerably improves the lap time of MPC_{N_S} in both the *kinematics* and *Pacejka model*, for all tested combinations of N_S and N_L. With the *kinematics model* (Table 2), our method achieves a lap time close to that of the reference method, MPC_{N_L}, and also outperforms the selected baseline models. Additionally, we provide an illustration of all three trajectories (from MPC_{N_S}, MPC_{N_L}, and ZipMPC) using the *Pacejka* model in Figure 3. The figure highlights the substantial improvement from MPC_{N_S} to ZipMPC even though they have the same horizon length in the MPC optimization, as ZipMPC learns to smooth harsh curves present in MPC_{N_S}, similar to MPC_{N_L}, consequently reducing the average lap time. In Table 3, we additionally present the optimization time reduction during the execution of ZipMPC compared to MPC_{N_L}.

Table 2: Lap time (sec.) comparison using the *kinematic model*. Mean and standard deviation are reported over ten runs with noise on the initial state. Results marked with $-$ did not complete the lap.

N_S	N_L	MPC_{N_L} (reference)	MPC_{N_S}	BO	eMPC	ZipMPC
5	18	8.358 ± 0.015	9.103 ± 0.043	8.847 ± 0.016	$-$	$\mathbf{8.436 \pm 0.012}$
5	25	8.286 ± 0.012	9.103 ± 0.043	8.955 ± 0.028	$-$	$\mathbf{8.394 \pm 0.012}$
10	18	8.358 ± 0.015	8.556 ± 0.018	8.580 ± 0.013	$-$	$\mathbf{8.370 \pm 0.000}$
10	25	8.286 ± 0.012	8.556 ± 0.018	8.610 ± 0.013	$-$	$\mathbf{8.325 \pm 0.015}$

Table 3: Lap time in seconds (left) and execution time reduction for each MPC step (right) using the *Pacejka model*. Mean and standard deviation are reported over ten lap runs with slight noise on the initial state. Results marked with $-$ never completed a lap (infeasible). Results marked with * completed the lap at least 80% of the attempts.

		Lap time (seconds)			Execution time reduction (per step)
N_S	N_L	MPC_{N_L} (ref.)	MPC_{N_S}	ZipMPC	$(1 - \frac{t(\text{ZipMPC})}{t(MPC_{N_L})}) * 100\%$
6	35	11.09 ± 0.07	$-$	$\mathbf{13.83 \pm 0.43}^*$	$82.5 \pm 1.1\%$
6	45	9.89 ± 0.10	$-$	$\mathbf{12.82 \pm 0.33}^*$	$87.9 \pm 0.4\%$
12	35	11.09 ± 0.07	20.36 ± 0.20	$\mathbf{13.36 \pm 0.47}$	$68.5 \pm 1.6\%$
12	45	9.89 ± 0.10	20.36 ± 0.20	$\mathbf{12.82 \pm 0.28}$	$76.6 \pm 0.5\%$

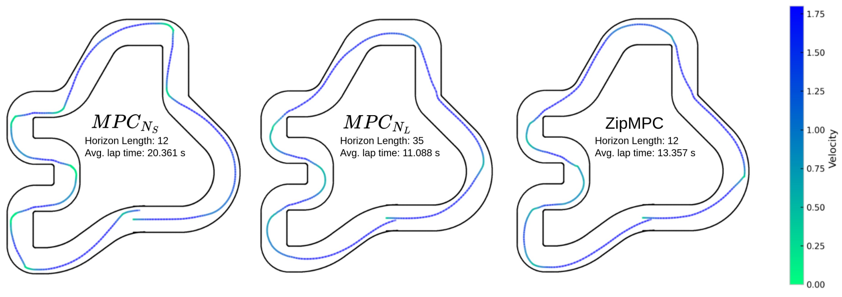


Figure 3: Comparison of three closed-loop trajectories (MPC_{N_S}, MPC_{N_L}, and ZipMPC) with velocity information on the train track using the *Pacejka model*. MPC_{N_S} and MPC_{N_L} correspond to the optimal trajectory given a manual cost set to the MPC with a horizon of 12 and 35 steps, respectively. Trajectories in this and all other figures are counterclockwise.

Track Generalization. To demonstrate the capability of our framework to generalize inferred trajectories across different tracks, we trained the model on a track that is complex enough to include a variety of curve sequences. We then tested the lap performance not only on the same track but also on two different tracks: one longer and one shorter, containing curvatures of different steepness and directions, without extrapolating the maximum steepness. The results, obtained using the *kinematic* model, are presented in Table 4 and in Figure 4. In all scenarios, even in unseen tracks during training, ZipMPC was able to decrease the lap time compared to MPC_{N_S}, specifically using the *kinematic* model, where the lap times of ZipMPC were very close to the target MPC_{N_L}.

Table 4: Generalization to different tracks. Lap times (seconds) comparison are reported using both the *kinematic* and the *Pacejka* model executed on the trained and two new tracks. Mean and standard deviation are reported over ten runs with noise on the initial state.

Dynamics model	N_S	N_L	Track	MPC_{N_L} (reference)	MPC_{N_S}	ZipMPC
Kinematics	5	25	Train	8.286 ± 0.012	9.090 ± 0.048	$\mathbf{8.394 \pm 0.012}$
			Test 1	6.273 ± 0.009	6.891 ± 0.040	$\mathbf{6.312 \pm 0.015}$
			Test 2	8.790 ± 0.000	9.537 ± 0.031	$\mathbf{8.823 \pm 0.009}$
Pacejka	12	45	Train	9.894 ± 0.100	20.361 ± 0.201	$\mathbf{12.822 \pm 0.283}$
			Test 1	8.718 ± 0.034	15.123 ± 0.076	$\mathbf{12.000 \pm 0.075}$
			Test 2	10.245 ± 0.040	17.736 ± 0.439	$\mathbf{15.123 \pm 0.078}$

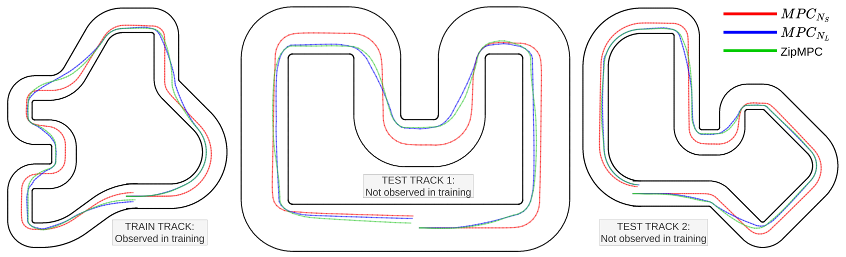
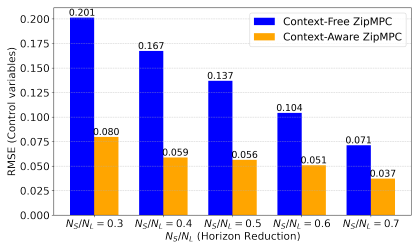


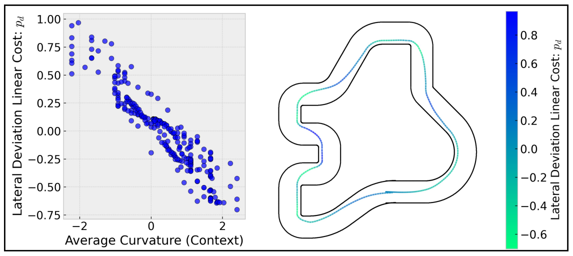
Figure 4: Trajectories from MPC_{N_S}, MPC_{N_L}, and ZipMPC on three tracks using the *kinematics* model with $N_S = 5$ and $N_L = 25$. Only the left-most track was observed in the training process.

4.2 Ablation Study in Simulation

Context relevance. We investigate the importance of incorporating context as input to the NN in our framework. To this end, we compare our proposed method, where the NN optimizes $h_Q^\theta(x(k), Z_{k,N_L})$ and $h_P^\theta(x(k), Z_{k,N_L})$, against a baseline variant of the framework where the context is excluded, and the NN infers $h_Q^\theta(x(k))$ and $h_P^\theta(x(k))$. The results, provided in Figure 5-a, indicate that for shorter prediction horizons (N_S), incorporating context from the long-horizon (N_L) provides a substantial performance improvement. This benefit diminishes as N_S approaches N_L, where the context difference becomes less critical for the learning model. Furthermore, we observe a trend of improved performance with increasing N_S, attributable to better initial estimates of the MPC cost and enhanced capacity of the MPC cost optimization.



(a) RMSE comparison of context-free and aware ZipMPC across different ratios of short (learned) and long-horizons (N_S/N_L). $N_L = 20$ was set in this experiment for the *kinematic* model.



(b) The left graph illustrates the correlation between p_d and the average curvature observed over the prediction horizon. The right graph shows the value of the learned parameter p_d along the trajectory. The experiment was using $N_S = 5$ and $N_L = 18$.

Figure 5: Ablation experiments for the *kinematics bicycle model*

Learned Parameters To better understand the role of context in improving trajectory generation, we analyze how the learned cost component associated with lateral deviation (denoted here as p_d, i.e., the linear cost corresponding to the lateral deviation in $P^{\theta}_{N_S}$) varies along the track. As shown in Figure 5-b, the graph on the right reveals significant variability in the p_d values throughout the track. Specifically, right-hand curves result in learned positive p_d values, while left-hand curves lead to learned negative p_d values, aligning with intuitive expectations. The left graph displays the same p_d values in a scatter plot, plotted against the curvature. This plot highlights a strong correlation between the average observed upcoming curvature and the learned p_d values.

4.3 Real World Results

Using the test bench presented in [49] and illustrated in Figure 6, we verified our simulation results on hardware. Using the *Pacejka* model, we trained the ZipMPC for various combinations of N_S and N_L in simulation and employed the obtained model on hardware without any retraining. Figure 7 illustrates a comparison of the closed-loop trajectories of our proposed ZipMPC and MPC_{N_S} with $N_S = 20$ and $N_L = 40$ for different instances in time. It can be seen how the ZipMPC incorporates the additional contextual information to improve racing performance. Results for further horizon lengths are presented in Appendix C.4. **Accompanying videos are found in the supplementary material.**

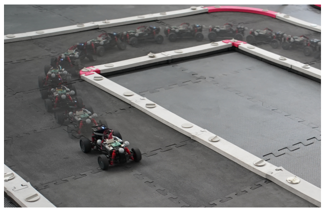


Figure 6: Illustration of the employed miniature racing platform.

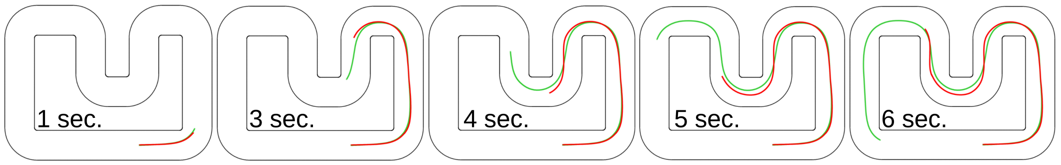


Figure 7: Comparison of closed-loop trajectories of ZipMPC (green) and MPC_{N_S} (red) for different instances in time, executed on hardware. Thereby N_S is chosen as 20 and N_L is set equal to 40.

5 Conclusion

Employing differentiable model predictive control (MPC) and neural networks, we developed a new algorithm to imitate long-horizon MPC solutions using an MPC with a significantly shorter prediction horizon. This is achieved by incorporating long-horizon contextual information into a learned cost vector for the short-horizon MPC. The conducted experiments show that the proposed method combines the strengths of explicit model predictive control with the tools for autonomous tuning of cost parameters. This includes a reduction of the computational time required to optimize controller inputs, the preservation of constraint feasibility with respect to system dynamics, and enhanced generalization capabilities to unseen environments during training. The conducted real world experiments on a miniature racing platform further validate the observations from the simulation.

6 Limitations

In the following section, we discuss on current limitations of the proposed method, as well as potential strategies on how to address them.

Dependency on differentiable MPC method. The performance of a resulting ZipMPC is highly dependent on the capabilities of the chosen method to propagate gradients through the MPC optimization process, and therefore directly inherits its limitations. In the presented case, this includes inaccuracies resulting from the convex approximations at each time step, as well as the usage of exact penalties to allow for the inclusion of inequality constraints on state and not only input variables. However, these limitations are expected to be reduced with future developments in the corresponding field of research.

Sentivity to infeasible training samples. Furthermore, our method has shown sensitive behaviour to the sampling of infeasible initial states during training, i.e., states for which the solver is not capable of finding a solution that fulfills the existing constraints for all time steps of the chosen prediction horizon. While finding an approximation of the feasible region was not too difficult for the problem description at hand, this might be harder for more complex problem descriptions. Nevertheless, such an issue can be addressed by running a standard solver in parallel, throwing an error whenever an infeasible initial state is detected, and neglecting the corresponding sampling point for training.

Nonconvexity of imitation loss. The nonconvexity of the proposed imitation loss with respect to the parameters θ of our parametric mapping function might result in convergence to highly suboptimal local minima. While we did not experience any nonconvexity issues employing the *kinematic* model, we addressed them for the *Pacejka* model by deploying a selection mechanism to save the model with the best lap performance in a fixed number of iterations.

Theoretical guarantees. Our proposed method is partly motivated by the results on long-horizon MPC stability and recursive feasibility without terminal ingredients. However, given the slightly imperfect approximation of MPC_{N_L} with ZipMPC, we cannot assume to inherit the provided theoretical guarantees of the former but only show them empirically. Investigations of this matter, e.g., quantifying the approximation error, present themselves as an interesting topic for future work.

Acknowledgments

This work has been partially supported by the Industrial Graduate School Collaborative AI & Robotics funded by the Swedish Knowledge Foundation Dnr:20190128, and the Knut and Alice Wallenberg Foundation through Wallenberg AI, Autonomous Systems and Software Program (WASP).

References

[1] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter. Feedback mpc for torque-controlled legged robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[2] G. Romualdi, S. Dafarra, G. L'Erario, I. Sorrentino, S. Traversaro, and D. Pucci. Online non-linear centroidal mpc for humanoid robot locomotion with step adjustment. In *2022 International Conference on Robotics and Automation (ICRA)*, 2022.

[3] J.-P. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter. A unified mpc framework for whole-body dynamic locomotion and manipulation. *IEEE Robotics and Automation Letters*, 2021.

[4] O. Villarreal, V. Barasuol, P. M. Wensing, D. G. Caldwell, and C. Semini. Mpc-based controller with terrain insight for dynamic legged locomotion. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[5] G. P. Incremona, A. Ferrara, and L. Magni. Mpc for robot manipulators with integral sliding modes generation. *IEEE/ASME Transactions on Mechatronics*, 2017.

[6] E. Arcari, M. V. Minniti, A. Scampicchio, A. Carron, F. Farshidian, M. Hutter, and M. N. Zeilinger. Bayesian multi-task learning mpc for robotic mobile manipulation. *IEEE Robotics and Automation Letters*, 2023.

[7] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger. Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 2019.

[8] G. Costa, J. Pinho, M. A. Botto, and P. U. Lima. Online learning of mpc for autonomous racing. *Robotics and Autonomous Systems*, 2023.

[9] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 2015.

[10] C. A. Rodriguez, P. Ponce, and A. Molina. Anfis and mpc controllers for a reconfigurable lower limb exoskeleton. *Soft Computing*, 2017.

[11] N. Dunkelberger, J. Berning, E. M. Schearer, and M. K. O'Malley. Hybrid fes-exoskeleton control: Using mpc to distribute actuation for elbow and wrist movements. *Frontiers in Neurorobotics*, 2023.

[12] Z. Sun, A. Iyer, K. Lambeth, C. Cleveland, and N. Sharma. Knee extension tracking and fatigue regulation results using a robust mpc approach in a hybrid exoskeleton. *Control Engineering Practice*, 2023.

[13] R. Hovorka, V. Canonico, L. J. Chassin, U. Haueter, M. Massi-Benedetti, M. O. Federici, T. R. Pieber, H. C. Schaller, L. Schaupp, T. Vering, et al. Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes. *Physiological measurement*, 2004.

[14] P. S. Rivadeneira, J. L. Godoy, J. Sereno, P. Abuin, A. Ferramosca, and A. H. González. Impulsive mpc schemes for biomedical processes: Application to type 1 diabetes. In *Control applications for biomedical engineering systems*. Elsevier, 2020.

[15] J. Sawma, F. Khatounian, E. Monmasson, R. Ghosn, and L. Idkhajine. The effect of prediction horizons in mpc for first order linear systems. In *2018 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2018.

[16] C. U. Dogruer. Optimizing computational complexity in mpc with lagrange polynomial-based data reduction. In *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE, 2024.

[17] L. P. Fröhlich, C. Küttel, E. Arcari, L. Hewing, M. N. Zeilinger, and A. Carron. Contextual tuning of model predictive control for autonomous racing. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[18] J. F.-S. Lin, P. Carreno-Medrano, M. Parsapour, M. Sakr, and D. Kulić. Objective learning from human demonstrations. *Annual Reviews in Control*, 2021.

[19] N. Ab Azar, A. Shahmansoorian, and M. Davoudi. From inverse optimal control to inverse reinforcement learning: A historical review. *Annual Reviews in Control*, 2020.

[20] S. Gros and M. Zanon. Data-driven economic nmpc using reinforcement learning. *IEEE Transactions on Automatic Control*, 2019.

[21] M. Zanon, S. Gros, and A. Bemporad. Practical reinforcement learning of stabilizing economic mpc. In *2019 18th European Control Conference (ECC)*. IEEE, 2019.

[22] A. Romero, Y. Song, and D. Scaramuzza. Actor-critic model predictive control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14777–14784, 2024. doi: 10.1109/ICRA57147.2024.10610381.

[23] L. P. Fröhlich, C. Küttel, E. Arcari, L. Hewing, M. N. Zeilinger, and A. Carron. Contextual tuning of model predictive control for autonomous racing. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022.

[24] S. Abdufattokhov, M. Zanon, and A. Bemporad. Learning convex terminal costs for complexity reduction in mpc. In *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021.

[25] K. Seel, A. B. Kordabad, S. Gros, and J. T. Gravdahl. Convex neural network-based cost modifications for learning model predictive control. *IEEE Open Journal of Control Systems*, 2022.

[26] C. A. Orrico, B. Yang, and D. Krishnamoorthy. On building myopic mpc policies using supervised learning. *arXiv preprint arXiv:2401.12546*, 2024.

[27] U. Rosolia and F. Borrelli. Sample-based learning model predictive control for linear uncertain systems. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019.

[28] S. Lucia and B. Karg. A deep learning-based approach to robust nonlinear model predictive control. *IFAC-PapersOnLine*, 2018.

[29] H. Alsmeier, L. Theiner, A. Savchenko, A. Mesbah, and R. Findeisen. Imitation learning of mpc with neural networks: Error guarantees and sparsification. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*. IEEE, 2024.

[30] S. S. P. Kumar, A. Tulsyan, B. Gopaluni, and P. Loewen. A deep learning architecture for predictive control. *IFAC-PapersOnLine*, 2018.

[31] G. BakIr. *Predicting structured data*. MIT press, 2007.

[32] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder. End-to-end constrained optimization learning: A survey. In *International Joint Conference on Artificial Intelligence*, 2021.

[33] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*. PMLR, 2017.

[34] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 2019.

[35] J. Drgoňa, K. Kiš, A. Tuor, D. Vrabie, and M. Klaučo. Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems. *Journal of Process Control*, 2022.

[36] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 2018.

[37] F. S. Acerbo, J. Swevers, T. Tuytelaars, and T. D. Son. Evaluation of mpc-based imitation learning for human-like autonomous driving. *IFAC-PapersOnLine*, 2023.

[38] B. Kouvaritakis and M. Cannon. Model predictive control. *Switzerland: Springer International Publishing*, 2016.

[39] L. Grüne, J. Pannek, L. Grüne, and J. Pannek. *Nonlinear model predictive control*. Springer, 2017.

[40] A. Boccia, L. Grüne, and K. Worthmann. Stability and feasibility of state constrained mpc without stabilizing terminal constraints. *Systems and Control Letters*, 2014.

[41] R. Soloperto, J. Köhler, and F. Allgöwer. A nonlinear mpc scheme for output tracking without terminal ingredients. *IEEE Transactions on Automatic Control*, 2023.

[42] J. Köhler, M. A. Müller, and F. Allgöwer. Constrained nonlinear output regulation using model predictive control. *IEEE Transactions on Automatic Control*, 2022.

[43] P. Donti, B. Amos, and J. Z. Kolter. Task-based end-to-end model learning in stochastic optimization. *Advances in neural information processing systems*, 2017.

[44] B. Wilder, B. Dilkina, and M. Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[45] A. A. Lahoud, E. Schaffernicht, and J. A. Stork. Learning solutions of stochastic optimization problems with bayesian neural networks. In *International Conference on Artificial Neural Networks*. Springer, 2024.

[46] R. Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[47] W. F. Milliken, D. L. Milliken, and L. D. Metz. *Race car vehicle dynamics*. SAE international Warrendale, 1995.

[48] Y. S. Quan and C. C. Chung. Approximate model predictive control with recurrent neural network for autonomous driving vehicles. In *2019 58th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. IEEE, 2019.

[49] A. Carron, S. Bodmer, L. Vogel, R. Zurbrügg, D. Helm, R. Rickenbach, S. Muntwiler, J. Sieber, and M. N. Zeilinger. Chronos and crs: Design of a miniature car-like robot and a software framework for single and multi-agent robotics and control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023.

[50] L. Brunke. *Learning Model Predictive Control for Competitive Autonomous Racing*. PhD thesis, Institute of Mechanics, 2018.

[51] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 2006.

[52] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 2019.

[53] R. Reiter and M. Diehl. Parameterization approach of the frenet transformation for model predictive control of autonomous vehicles. In *2021 European Control Conference (ECC)*. IEEE, 2021.

A Autonomous Racing as an Application Example

In this section, we detail the preliminaries for modeling an autonomous racing application. It includes the Frenet coordinate frame in A.1, the corresponding car dynamics in A.2, as well as the chosen MPC design in A.3. Furthermore, we comment on the employed model and control parameter values in A.4.

A.1 Frenet Frame

The Frenet coordinate frame [47] has emerged as one of the standard coordinate frames in racing applications. It describes a car's position with respect to a given trajectory using its progress along the trajectory, its lateral deviation from the trajectory, and its orientation with respect to the trajectory, indicated with σ, d, and ϕ, respectively. We denote the corresponding state vector as $x = [\sigma, d, \phi]$ and illustrate it in Figure 8.

A.2 Car Dynamics

To model the car's dynamics, we rely on the *kinematic bicycle* and the *Pacejka* model [46, 50], both of which are further detailed below. For this purpose, we denote the car's rear length with l_r and the distance from the center of mass to its front wheel axis with l_f. Additionally, we introduce the progress-dependent curvature model $\kappa(\sigma)$, which later on defines our context variable ζ. All considered tracks are modeled with constant track width 2ω. For an illustration of the introduced parameters, you can once again refer to Figure 8.

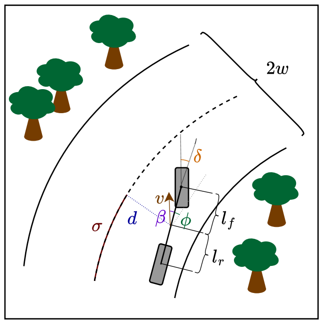


Figure 8: Illustration of the state vector defining a car's position in the Frenet frame, including its progress coordinate σ, its lateral deviation d, and its respective orientation ϕ. Furthermore indicated are its velocity v and the steering angle δ as well as the cars rear and front length, l_r and l_f.

A.2.1 Pacejka Model used in Simulation

The *Pacejka* model is primarily suitable for modeling high-velocity applications but is prone to numerical difficulties when dealing with small velocities. It takes a torque τ and a steering angle of the front wheels δ as control inputs and models the states σ, d, ϕ, r, v_x, and v_y, with the latter defining yaw-rate, longitudinal and lateral velocity, respectively. Then, the lateral slip angles are calculated as

$$\alpha_f(k) = -\arctan 2\left(\frac{-v_y(k) - l_f r(k)}{|v_x(k)|}\right) + \delta(k), \quad \alpha_r(k) = -\arctan 2\left(\frac{-v_y(k) + l_f r(k)}{|v_x(k)|}\right),$$

and the lateral tire forces F_f and F_r, as well as the motor force F_m follow as

$$\begin{aligned} F_f(k) &= -D_f \sin(C_f \tan^{-1}(B_f a_f(k))) \\ F_r(k) &= -D_r \sin(C_r \tan^{-1}(B_r a_r(k))) \\ F_m(k) &= (C_{m1} - C_{m2} v_x(k))\tau - C_d v_x(k) v_x(k) - C_{roll} \end{aligned}$$

with D_f, C_f, B_f, D_r, C_r, and B_r describing the lateral force parameters of front and rear wheels, as well as C_{m1}, C_{m2}, C_d and C_{roll} defining the longitudinal force parameters. Additionally, denoting with m the mass of the car, with I_z its moment of inertia, and with T the chosen discretization time, the discrete-time dynamics, discretized using the Euler method, follow as

$$\begin{aligned}
\sigma(k+1) &= \sigma(k) + T\left(\frac{v_x(k)\cos(\phi(k)) - v_y(k)\sin(\phi(k))}{1-\kappa(\sigma(k))d(k)}\right)\\
d(k+1) &= d(k) + T\left(v_x(k)\cos(\phi(k)) + v_y(k)\sin(\phi(k))\right)\\
\phi(k+1) &= \phi(k) + T\left(r(k) - \kappa(\sigma(k))\frac{v_x(k)\cos(\phi(k)) - v_y(k)\sin(\phi(k))}{1-\kappa(\sigma(k))d(k)}\right)\\
r(k+1) &= r(k) + T\left(\frac{1}{I_z}(F_f(k)l_f\cos(\delta(k)) - F_r(k)l_r)\right)\\
v_x(k+1) &= v_x(k) + T\left(\frac{1}{m}(F_m(k) - F_f(k)\sin(\delta(k)) + mv_y(k)r(k))\right)\\
v_y(k+1) &= v_y(k) + T\left(\frac{1}{m}(F_r(k) + F_f(k)\cos(\delta(k)) - mv_x(k)r(k))\right).
\end{aligned}$$

A.2.2 Pacejka Model used on Hardware

For our obtained real world results we employed a slightly adapted version of the *Pacejka* model used in simulation to increase the model accuracy. Therefore, we additionally define the friction force F_{fr}, which, introducing the longitudinal parameters C_{d0}, C_{d1} and C_{d2}, follow as

$$F_{fr}(k) = \text{sign}(v_x(k))(-C_{d0} - C_{d1}v_x(k) - C_{d2}v_x(k)v_x(k)).$$

Furthermore, we alter the motor force F_m and split F_f and F_r into x and y components, obtaining F_{fx}, F_{fy} and F_{rx}, F_{ry}, respectively. With γ denoting the ratio between front and rear wheel drive force, these modifications result in

$$\begin{aligned}
F_m(k) &= (C_{m1} - C_{m2}v_x(k))\tau\\
F_{fx}(k) &= F_m(k)(1-\gamma)\\
F_{rx}(k) &= F_m(k)\gamma\\
F_{fy}(k) &= D_f\sin(C_f\tan^{-1}(B_f a_f(k)))\\
F_{ry}(k) &= D_r\sin(C_r\tan^{-1}(B_r a_r(k))).
\end{aligned}$$

Correspondingly, these changes cause a different one-step forward prediction of r, v_x, and v_y, detailed in the following.

$$\begin{aligned}
r(k+1) &= r(k) + T\left(\frac{1}{I_z}(F_{fy}(k)l_f\cos(\delta(k)) + F_{fx}(k)l_f\sin(\delta(k)) - F_{ry}(k)l_r)\right)\\
v_x(k+1) &= v_x(k) + T\left(\frac{1}{m}(F_m(k) - F_{fy}(k)\sin(\delta(k)) + F_{fx}(k)\cos(\delta(k)) + mv_y(k)r(k)) + F_{fr}\right)\\
v_y(k+1) &= v_y(k) + T\left(\frac{1}{m}(F_{ry}(k) + F_{fy}(k)\cos(\delta(k)) + F_{fx}(k)\sin(\delta(k)) - mv_x(k)r(k))\right).
\end{aligned}$$

A.2.3 Kinematic Bicycle Model

Differently from the *Pacejka* model, the *kinematic bicycle* model does not model tire forces or lateral velocities, but has shown its usefulness primarily in small to medium-velocity applications. Furthermore, it is computationally cheaper in its application than the *Pacejka* model. The *kinematic bicycle* model takes the acceleration a, as well as the front wheel steering angle δ as control inputs and defines the side slip angle

$$\beta(k) = \tan^{-1}\left(\frac{l_r}{l_f+l_r}\tan(\delta(k))\right).$$

The discrete-time dynamics, discretized using the Euler method, accordingly follow as

$$\begin{aligned}
\sigma(k+1) &= \sigma(k) + T(v(k)\frac{\cos(\phi(k)+\beta(k))}{1-\kappa(\sigma(k))d(k)}) \\
d(k+1) &= d(k) + T(v(k)\sin\phi(k) + \beta(k)) \\
\phi(k+1) &= \phi(k) + T(\frac{v(k)}{l_f}\sin(\beta(k)) - \kappa(\sigma(k))v(k)\frac{\cos(\phi(k)+\beta(k))}{1-\kappa(\sigma(k))d(k)}) \\
v(k+1) &= v(k) + Ta(k).
\end{aligned}$$

A.3 MPC for Racing

In the Frenet frame, the classical Model Predictive Contouring Controller (MPCC), one of the most commonly used planning and control algorithms for autonomous racing in the Cartesian frame [9], can be represented as a nonlinear MPC controller with quadratic cost and polytopic constraint. The latter holds since the track boundary constraint transforms into a two-sided half-space constraint for a constant track width 2ω. Additional limitations on state and control-input variables can be implemented via the polytopic inequality constraint $G[x_i, u_i]^\top \leq g$, where $G \in \mathbb{R}^{c\times(n+m)}$ and $g \in \mathbb{R}^c$. Accordingly, the optimization problem, solved in receding horizon fashion, follows as

$$\begin{aligned}
[X^*_{N+1}, U^*_N] := \underset{\{x_i\}_0^{N+1}, \{u_i\}_0^N}{\arg\min} \; & \sum_{i=0}^{N} \|[x_i, u_i]\|^2_{q_i\mathbb{I}} + \langle p_i, [x_i, u_i]\rangle \\
\text{s.t. } & x_{i+1} = f(x_i, u_i, \kappa(\sigma_i)), \\
& -\omega \leq d_i \leq \omega, \\
& G[x_i, u_i]^\top \leq g, \\
& x_0 = x(k), \\
& i = 0, \dots, N.
\end{aligned}$$

Note that depending on the choice of the car dynamics model, $f(x, u)$ is either replaced with the *Pacejka* or the *kinematic* model. Furthermore, to avoid cost scaling with an increasing progress value σ_i, we introduce an additional state $\sigma_{\Delta,i} = \sigma_i - \sigma_0$, which is independent of the current position on the race track, but only reflects the progress within the current prediction horizon. Finally, to increase clarity, we already indicated the context dependence, i.e., the dependence on track curvature $\kappa(\sigma)$, in the dynamics.

A.4 Model Parameters

In this subsection, we detail the deployed model, constraint, and objective parameter values for the presented experiments in simulation and hardware, respectively. For the experiments conducted in simulation with the *kinematic* model, the manual cost vectors q^M and p^M are chosen identical for the long and short-horizon and are presented in Table 5. For the experiments conducted in simulation with the *Pacejka* model, the manual cost vectors q^M and p^M are chosen as presented in Table 6. Finally, for the experiments conducted on hardware, the manual cost vectors q^M and p^M are chosen dependent on the deployed horizon length and follow as presented in Table 7.

Table 5: Manual cost vectors used in simulation with the *kinematic* model.

Parameter	σ	d	ϕ	v	σ_0	σ_Δ	a	δ
$q^M =$	0	3	1	0.01	0.01	0.01	0.01	1
$p^M =$	0	0	0	0	0	-8	0	0

The model and constraint parameter values employed for the considered models in simulation and on hardware are presented in Table 8.

Table 6: Manual cost vectors used in simulation with the *Pacejka* model

Parameter	σ	d	ϕ	r	v_x	v_y	σ_0	σ_Δ	τ	δ
$q^M =$	0	50	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
$p^M =$	0	0	0	0	0	0	0	-8	0	0

Table 7: Manual cost vectors used on the *Pacejka* model for hardware (scaled by N)

Parameter	σ	d	ϕ	r	v_x	v_y	σ_0	σ_Δ	τ	δ
$N * q^M =$	0	500	5	1	1	1	1	1	1	100
$N * p^M =$	0	0	0	0	0	0	0	-40	0	0

Table 8: Model and constraint parameter values employed for the *kinematic* model in simulation, the *Pacejka* model in simulation, as well as the *Pacejka* model on hardware.

Parameter	Kinematic (sim)	Pacejka (sim)	Pacejka (hard)
l_r	0.05	0.05	0.038
l_f	0.05	0.05	0.052
m	–	0.200	0.181
T	0.03	0.03	0.026
ω	0.2	0.2	0.2
D_f	–	0.43	0.65
C_f	–	1.4	1.5
B_f	–	0.5	5.2
D_r	–	0.6	1.0
C_r	–	1.7	1.45
B_r	–	0.5	8.5
C_{m1}	–	0.9803	0.9803
C_{m2}	–	0.0181	0.0181
C_d0	–	0.0275	0.085
C_d1	–	–	0.01
C_d2	–	–	0.0275
C_{roll}	–	0.085	–
$a_{\max}$	1.0	1.0	1.0
$\delta_{\max}$	0.4	0.5	0.4
$v_{\max}$	1.8	1.8	2.0

B Experiment Setup

In the following, we detail the learning framework in B.1 and provide some further information on the chosen hardware setup in B.2. In B.3 we discuss some limitations of the selected setup.

B.1 Learning Framework Setup and Implementation Details

Technical Details. All experiments were executed on a CPU, as the primary computational bottleneck arose from the numerical solvers integrated within the training loop. The hardware platform utilized was an Intel(R) Core(TM) i7-13700KF processor (13th generation), comprising 24 cores. To accelerate training, the generation of target trajectories through simulation was parallelized. A batch size of 80 was employed for the simulation-based experiments, while a reduced batch size of 40 was adopted for the hardware-oriented experiment.

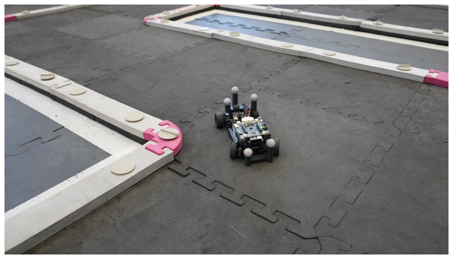
Input to the Neural Network. As we described in the main paper, the input of the NN is composed by Z_{k,N_L} and some state variables from $x(k)$. Z_{k,N_L} is composed by a timeseries of the track curvature from the current state $x(k)$. Specifically, we pick as many curvature values as needed to reach the maximum possible σ_Δ in the long-horizon MPC optimization. For example, with a discretization time $T = 0.03$, a horizon length $N_L = 40$, and a maximum velocity $v_{max} = 2m/s$, we calculate $\sigma_\Delta = 0.03 * 40 * 2 = 2.4$, and pick all curvature points available between $\kappa(\sigma(k))$ and $\kappa(\sigma(k) + 2.4)$ to fill the Z_{k,N_L} vector. Regarding the state variables from $x(k)$, we only choose the variables v, d, and ϕ because these state variables are general enough and useful to be applied to other tracks. This is not true for σ, for example, which would lead to overfitting to the specific training track.

Neural Network and Training. The neural network architecture used to predict the parameters of the MPC cost is composed of a convolutional feature extractor followed by fully connected layers. The model takes as input a concatenation of three global context variables represented by v, d, and ϕ coming from the initial state, and a time series Z_{k,N_L} with a horizon of N_L representing the next curvature steps. We process the data through convolutional layers, batch normalization and dropout layers. We pass this shared representation through four fully connected layers, each with 512 hidden units and LeakyReLU activation. The final shared representation is used to compute two outputs: (1) a global representation layer producing a fixed component of the MPC cost, and (2) a modulation layer generating time-varying components. The modulation layer produces outputs with dimensions $2 \times N_S \times (n + m)$, i.e., we learn both $P^{\theta}_{N_S}$ and $Q^{\theta}_{N_S}$. The final output of the model is scaled to ensure the predicted parameters are between a certain range. Further details can be found in the code provided in the supplementary material.

Imitation Loss and Validation. As detailed in the main text, the imitation loss we implement is the sum of the MSE for the state variables and the MSE for the control variables. Each of these components are computed between the inferred variables from the ZipMPC and MPC_{N_L}. However, the range of different variables varies. Therefore, we manually weigh the different variables to avoid having one or more variables that are much more important to the loss function than others. The weights can be found in the code provided in the supplementary material. Also, we consider only the first N_D steps of the trajectories from ZipMPC and MPC_{N_L} in the imitation loss for two main reasons. The first is to make the loss function less complicated, such that it reaches a better local minimum. The second is that we have observed that the first trajectory steps are the most important to be aligned, which is also in line with the receding horizon implementation of MPC, where only the first optimal control input (the most imminent one) is applied to the system. For validation purposes, we compute lap performance (based on a fixed initial state) using the training track after a fixed number of training iterations. After a large number of iterations, we select the model with the best lap performance on that fixed initial state.

B.2 Hardware Setup

For the obtained hardware results we employed the setup presented in [49]. There, the position and orientation of a 1/28-scaled miniature car (Figure 9a) is measured using a motion capture system, passing the information to a server that runs the control algorithm. The car is connected to the aforementioned server via wifi, obtaining its control input to race a predefined track (see Figure 9b). The experiments were conducted using an HP ProBook 440 G8 with an 11th-generation Intel Core i7 processor.


(a) Picture of the employed miniature race car on race track.

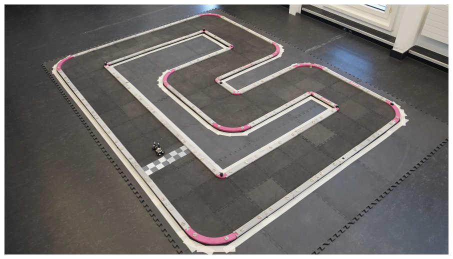

(b) Picture of the race track on which the real world results are obtained.

Figure 9: Pictures of used hardware setup.

B.3 Setup Limitations

Numerical Issues. The differentiable MPC method proposed in [36] does not handle hard constraints in the states. Therefore, we implement a soft constraint by adding a deterministic (non-learnable) high penalty in the MPC cost. By doing this, the solution from the differentiable MPC can be slightly different from the results of standard solvers, in our simulation experiments chosen as IPOPT [51], implemented using casadi [52]. Therefore, to address this during training, we exclude samples where this mismatch exceeds an arbitrary tolerance for gradient computation.

Frenet Transformation Irregularities. When the vehicle state has a significant lateral deviation from the centerline, it becomes increasingly susceptible to singularities, as the Frenet transformation's stability is compromised [53]. To circumvent this issue, we tightened the maximum lateral deviation, limiting the car's range to reduce the risk of approaching these unstable regions.

C Supplementary Results

In this section, we present additional results to add more evidence to the claims we state in the main paper. Therefore, we investigate the lap time correlation with prediction horizon length in C.1, followed by execution time comparisons in C.2, and a visualization of the learning process in C.3. This is concluded by additional hardware results in C.4.

C.1 Lap time with manual cost MPC

Investigating the correlation of control performance with MPC horizon length, we provide lap times, considering the same manual cost we used for the MPC_{N_S} and MPC_{N_L} methods in the reported tables of the main text, for an increasing horizon length. As presented in Tables 9 and 10, the lap time decreases and seems to converge to a certain value as we increase the horizon length in both the *kinematic* and *Pacejka* model. The results are obtained, using the same track as we used in our training process (i.e., Tables 2, 3, 1).

Table 9: Lap time (sec) comparison for various values of N using the *kinematic* model. Mean and standard deviation are reported over ten runs with noise introduced on the initial state.

N	Lap Time
5	9.103 ± 0.043
10	8.556 ± 0.018
15	8.409 ± 0.014
20	8.325 ± 0.015
25	8.286 ± 0.018
35	8.283 ± 0.031

Table 10: Lap time (sec) comparison for various values of N using the *Pacejka* model. Mean and standard deviation are reported over ten runs with noise introduced on the initial state. Results marked with $-$ did not complete the lap.

N	Lap Time
6	$-$
12	20.361 ± 0.201
18	14.088 ± 0.132
25	12.171 ± 0.133
30	11.634 ± 0.187
35	11.088 ± 0.070
40	10.350 ± 0.212
45	9.894 ± 0.100
50	9.615 ± 0.071
55	9.477 ± 0.036
60	9.417 ± 0.016

C.2 MPC Execution time

To illustrate the computational performance of our proposed method, we investigate its execution time employing IPOPT [51] for various combinations of N_S and N_L, and both models considered. In the provided tables (Table 11 and 12) it can be seen, that the inference time of our method is highly comparable to the corresponding short-horizon MPC (MPC_{N_S}). This is mainly due to the fast inference of neural networks.

Table 11: Execution time (ms) average for the optimization and inference process of each time step. Comparison for some values of N_S and N_L using the *kinematic* model, considering 1000 initial states in each run (as in the imitation loss experiment). Mean and standard deviation are reported over ten runs with noise introduced on the initial state.

N_S	N_L	MPC_{N_S}	MPC_{N_L}	ZipMPC
5	18	60 ± 16	196 ± 51	61 ± 16
5	25	60 ± 16	287 ± 102	61 ± 16
10	18	102 ± 32	196 ± 51	103 ± 32
10	25	102 ± 32	287 ± 102	103 ± 32

Table 12: Execution time (ms) average for the optimization and inference process of each time step. Comparison for some values of N_S and N_L using the *Pacejka* model, considering 1000 initial states in each run (as in the imitation loss experiment). Mean and standard deviation are reported over ten runs with noise introduced on the initial state.

N_S	N_L	MPC_{N_S}	MPC_{N_L}	ZipMPC
6	35	65 ± 59	476 ± 211	66 ± 59
6	45	65 ± 59	704 ± 337	66 ± 59
12	35	132 ± 56	476 ± 211	133 ± 56
12	45	132 ± 56	704 ± 337	133 ± 56

C.3 Visualizing the learning process

We analyze the effect of varying initial states in the *Pacejka* model to evaluate the robustness of our framework under diverse initial states and to compare its performance against the MPC_{N_S} and MPC_{N_L} baselines. Specifically, we initialize our model in various initial state configurations and observe its ability to approximate the "target" trajectory produced by MPC_{N_L}. To illustrate the progression of learning, we plot results at different stages of training: at 0 iterations (approximately mimicking the MPC_{N_S} method), and after 20, 50, and 100 iterations. The results provided in Figure 11 demonstrate that as the number of iterations increases, the trajectories generated by our model progressively align more closely with the target trajectory, highlighting the effectiveness of the learning process in refining the trajectory over time. Furthermore, in Figure 10, we augment the results illustrated in Figure 5a, visualizing the full learning curves of context-free and context-aware ZipMPC across different ratios of short (learned) and long-horizons (N_S/N_L). It is evident from this that the incorporation of context information enhances the learning performance in all of the selected scenarios.

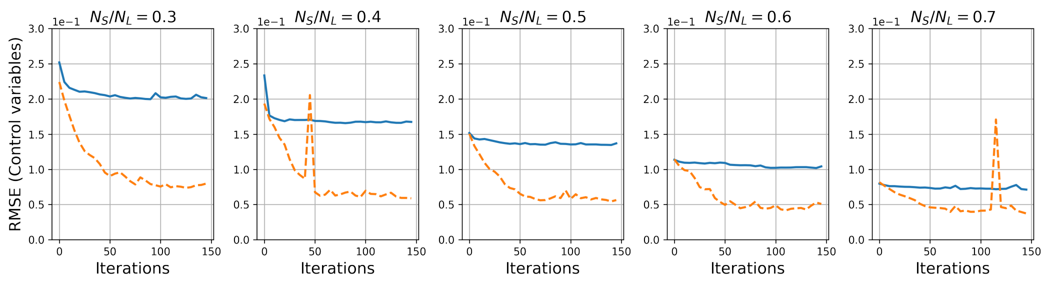


Figure 10: Validation curve comparison of context-free and context-aware ZipMPC across different ratios of short (learned) and long-horizons (N_S/N_L). This graph is analogous to Figure 5a but shows the full learning curve within the validation set. $N_L = 20$ was considered in this experiment for the *kinematic* model.

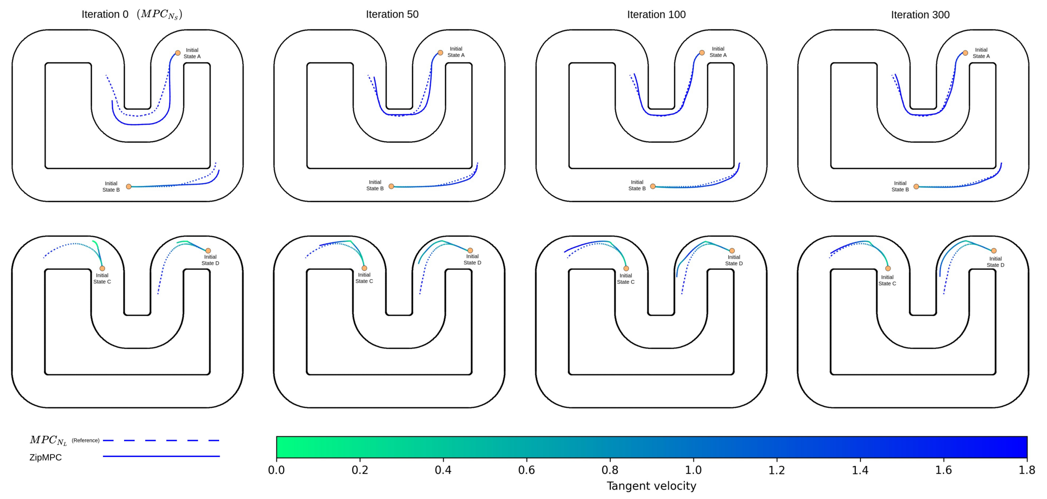
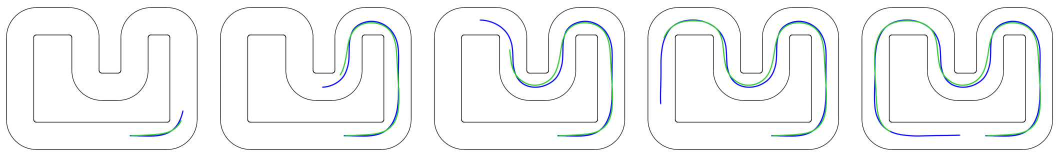


Figure 11: Comparison of trajectories inferred by our framework and the target trajectory from MPC_{N_L}. Top and bottow row illustrate two trajectories with different initial states using the *kinematics bycicle model* and the *Pacejka bycicle model*, respectively. Each column depicts the trajectory after training the model for 0, 50, 100, and 300 iterations.

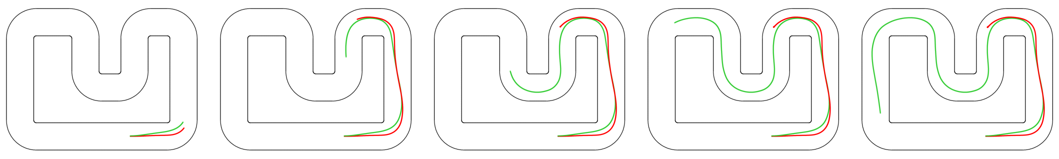
C.4 Additional hardware results

For the presented horizon length combination in Section 4.3, we further illustrate a comparison of the control behaviour of our learned ZipMPC and MPC_{N_L} for the same instances in time. The results are visualized in Figure 12. While the presented choice of $N_S = 20$ and $N_L = 40$ allows for a successful lap completion of both, our proposed ZipMPC method and MPC_{N_S}, we further tested our proposed method with values of N_S for which MPC_{N_S} is not capable of finishing the lap, i.e., we tested the horizon length combination $N_S = 10$ and $N_L = 25$. The comparison of our proposed ZipMPC method with MPC_{N_S} is illustrated in Figure 13 and Figure 14 visualizes the comparison of ZipMPC with the reference MPC_{N_L}. It can be seen that while the application with MPC_{N_S} gets stuck after 4 seconds in the third corner, the corresponding ZipMPC implementation not only outperforms the short-horizon MPC in the first few seconds but also successfully completes the lap. Furthermore, do the slight inaccuracies of the learned imitation behaviour, as well as the faster computation time, lead to an outperformance of MPC_{N_L}.



(a) 1sec. (b) 3sec. (c) 4sec. (d) 5sec. (e) 6sec.

Figure 12: Comparison of ZipMPC (green) and MPC_{N_L} (blue) for different instances in time, executed on hardware. Thereby N_S is chosen as 20 and N_L is set equal to 40.



(a) 1sec. (b) 3sec. (c) 4sec. (d) 5sec. (e) 6sec.

Figure 13: Comparison of ZipMPC (green) and MPC_{N_S} (red) for different instances in time, executed on hardware. Thereby N_S is chosen as 10 and N_L is set equal to 25.

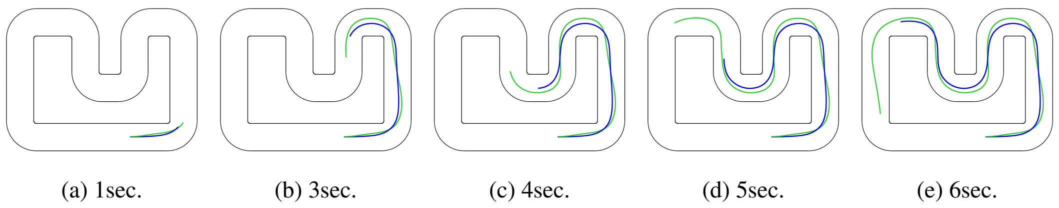

(a) 1sec. (b) 3sec. (c) 4sec. (d) 5sec. (e) 6sec.

Figure 14: Comparison of ZipMPC (green) and MPC_{N_L} (blue) for different instances in time, executed on hardware. Thereby N_S is chosen as 10 and N_L is set equal to 25.