Beyond Constant Parameters: Hyper Prediction Models and HyperMPC

Jan Węgrzynowski[1,2,3] **Piotr Kicki**[1,2,3] **Grzegorz Czechmanowski**[1,2,3]
Maciej Krupka[1], **Krzysztof Walas**[1,2]

[1]Institute of Robotics and Machine Intelligence, Poznan University of Technology, Poland
[2]IDEAS Research Institute, Warsaw, Poland, [3]IDEAS NCBR, Warsaw, Poland
jan.wegrzynowski@put.poznan.pl

Abstract: Model Predictive Control (MPC) is among the most widely adopted and reliable methods for robot control, relying critically on an accurate dynamics model. However, existing dynamics models used in the gradient-based MPC are limited by computational complexity and state representation. To address this limitation, we propose the Hyper Prediction Model (HyperPM) - a novel approach in which we project the unmodeled dynamics onto a time-dependent dynamics model. This time-dependency is captured through time-varying model parameters, whose evolution over the MPC prediction horizon is learned using a neural network. Such formulation preserves the computational efficiency and robustness of the base model while equipping it with the capacity to anticipate previously unmodeled phenomena. We evaluated the proposed approach on several challenging systems, including real-world F1TENTH autonomous racing, and demonstrated that it significantly reduces long-horizon prediction errors. Moreover, when integrated within the MPC framework (HyperMPC), our method consistently outperforms existing state-of-the-art techniques.
Videos and code are available at hyper-mpc.github.io/

Keywords: Dynamics Model Learning, Model Predictive Control, MPC

1 Introduction

One of the fundamental challenges in robotics is determining optimal actions to achieve specific objectives, a task generally referred to as control. This challenge is particularly pronounced for systems that are underactuated or operate at the limits of their physical capabilities. Many of these complexities have been recently resolved with learning-based controllers [1, 2, 3, 4]. However, if the model of the system is available, MPC has demonstrated outstanding performance across demanding tasks, such as agile drone flight [5, 6, 7], autonomous racing [8, 9], and legged locomotion [10, 11].

The effectiveness of any MPC-based approach fundamentally depends on the quality of the underlying dynamical system model, as its accuracy, robustness, and computational complexity critically influence the controller's performance. In general, the accuracy of the dynamics model can be improved at the expense of the increased computational effort by the use of data-driven components, such as Gaussian Processes (GP) [12, 13, 14], polynomials [15, 7] or even small neural networks [16, 17, 18]. However, to preserve the sparsity of the optimization problem, each of these models predicts the system's evolution based on its current state and control. This may not be enough to accurately capture the influence of unmodeled dynamics on the outputs we want to control, e.g., mass transfer during braking, tire deformation, suspension movement during aggressive car racing, or the position of the payload suspended from a drone by a rope. While recurrent models [19, 20] can capture these effects with a hidden state, the expansion of system state results in cubic growth in MPC complexity, which is often prohibitively expensive.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

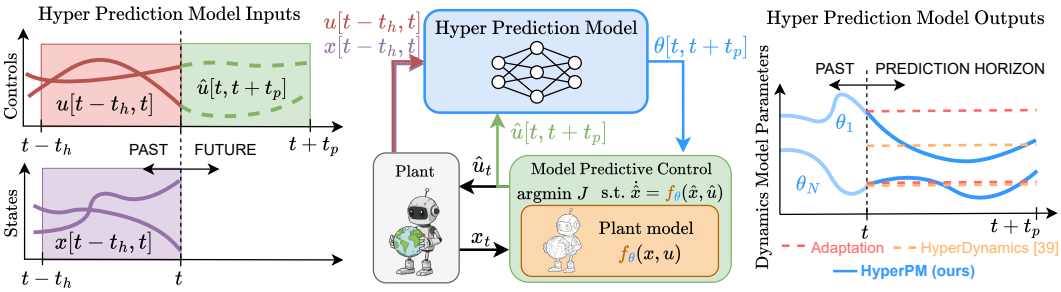

Figure 1: **HyperPM** leverages the recent state $x[t_c - t_h, t_c]$ and control $u[t_c - t_h, t_c]$ history, together with the planned controls $\hat{u}[t_c, t_c + t_p]$, to predict a trajectory of time-varying model parameters $\theta[t_c, t_c + t_p]$. Injecting these predicted parameters into the nominal dynamics yields more accurate long-horizon state forecasts, allowing **HyperMPC** to proactively anticipate and compensate for previously unmodeled effects.

To address these limitations, we propose encoding the unmodeled components of the system dynamics as variations in the parameters of the existing model. This allows us to increase the model's accuracy without incurring additional computational overhead during optimization. However, the essence of MPC lies in leveraging the aforementioned models to predict the system's behavior over a finite but preferably long horizon. Therefore, predicting only the static parameters of the model may not be sufficient to work over extended prediction horizons. Thus, in this paper, we propose a Hyper Prediction Model (HyperPM) – a framework that uses a neural network to predict the trajectories of the expected evolution of the model parameters, based on the history of observations and the planned trajectories of future control inputs (see Fig. 1). To the best of our knowledge, this is the first approach that goes beyond inferring the model based on its history, but also anticipates the expected evolution of the system. This allows us to obtain lightweight dynamics models with higher accuracy over the prediction horizon, enhancing MPC performance.

We evaluated the proposed approach on three challenging tasks: (i) swing-up of the simulated pendulum with backlash, (ii) trajectory tracking with a simulated drone with a payload suspended on a rope, and (iii) autonomous racing using a real F1TENTH car [21]. We show that the proposed HyperPM achieves over 33%, 12%, and 62% improvement in prediction accuracy in the pendulum, drone, and racing tasks, respectively, compared to an analytical model with constant parameters. Moreover, we show that the use of HyperMPC results in a 300% improvement over MPC with an analytical model in pendulum swing-up, 9% improvement in the case of the drone task, and almost 19% improvement over MPC with a residual model in autonomous racing.

The contributions of this paper are the following:
1) Hyper Prediction Model, a novel framework that predicts future trajectories of dynamics model parameters, based on the history of observations and planned actions,
2) HyperMPC, which integrates HyperPM with MPC to improve control performance without increasing the computational complexity of the dynamics model inside the optimization loop.

2 Related work

Model Predictive Control is a widely adopted approach for controlling complex dynamical systems [22]. In general, MPC algorithms can be categorized into two distinct groups: (i) sampling-based MPC and (ii) optimization-based MPC. The sampling-based MPC approaches, such as MPPI [23, 24] or iCEM [25], offer great flexibility in terms of the dynamics models they use. However, this flexibility comes at the cost of increased computational demand and typically the need to use powerful GPUs to enable parallel simulation of thousands of candidate control trajectories [23, 24, 26, 27, 28]. An alternative approach is to take advantage of numerical optimization instead [22]. However, gradient-based MPC approaches are typically limited to the use of physics-derived models of the controlled system [11, 29] or small function approximators [8, 7, 30]. In this paper, we focus on a crucial component of the previously mentioned MPC frameworks — developing models that yield accurate long-horizon predictions for non-Markovian systems.

Dynamics models play a key role in MPC, as they are expected to accurately predict the evolution of the system within a limited computational budget. A conventional approach in MPC is to use reduced-order analytical dynamics models derived from first principles [31, 32, 22]. At the other end of the spectrum, there are neural-network-based dynamics models [33, 34, 28]. Somewhere between these two extremes, we find physics-based analytical models augmented with some general function approximators, such as Gaussian Processes [12, 13, 14], polynomial basis expansions [7, 15], or neural networks [16, 17, 18]. These hybrid approaches aim to balance the accuracy of the data-driven models with the computational efficiency and robustness of the analytical ones.

Nevertheless, the approximation capabilities of all these models are limited by the system's partial observability. This limitation may be overcome by including the history of observations [16, 28], but doing so is infeasible in the context of optimization-based MPC [17, 18]. An intriguing approach to address this issue is to use hyper-neural networks [35, 36, 37], which can predict the parameters of other neural networks that act as the dynamics model [37]. A similar concept was presented in [9], where a neural network was used to predict certain parameters of the analytical model. However, both of these approaches assume predicting static parameters across the prediction horizon.

In our work, we build upon the ideas of [37] and [9], and exploit analytical models with parameters identified by a neural network, as in [9]. However, we demonstrate that one can improve the long-horizon prediction accuracy of a dynamics model by capturing the unmodeled dynamics through the expected evolution of model parameters over the prediction horizon.

Although the proposed technique resembles online parameter adaptation or incremental model learning [38, 13], these existing methods assume a single, time-invariant model throughout the entire prediction horizon. Our approach is orthogonal to these methods: it exploits knowledge of the planned control sequence to synthesize a time-varying dynamics model anticipating nominally unmodeled dynamics, thereby improving its accuracy over the prediction horizon. Crucially, nothing in the framework prevents online learning or weight adaptation of HyperPM, allowing further refinement while retaining its expressiveness.

3 Proposed Method

3.1 Considered problem

In this paper, we address the problem of identifying a dynamics model $\dot{\hat{x}} = f_\theta(\hat{x}, u)$, parametrized by θ, which accurately predicts the evolution of the observable system state x over the long prediction horizon. This can be formalized by

$$\arg\min_{\theta} \int_{t_c}^{t_c+t_p} \|x(t) - \hat{x}(t)\|_2 dt \quad \text{s.t.} \quad \dot{\hat{x}} = f_\theta(\hat{x}, u), \tag{1}$$

where t_c represents the current time, t_p the duration of the prediction horizon, e.g., 2 seconds, and $\hat{x}$ represents the predicted evolution of the system state. While this problem may be foundational to many downstream applications, we consider it within the context of Model Predictive Control, which exploits the long-horizon predictions to find an optimal control sequence.

To motivate our approach, consider a drone carrying a rope-suspended payload, where the observable state includes only the drone state variables - position, orientation, and linear and angular velocities. Consequently, a model relying solely on (x_t, u_t) will inevitably incur errors, because the influence of the swinging payload cannot be accurately captured without access to its unobserved state. One potential remedy is to employ a model that conditions on a finite history of observations; however, such models are computationally prohibitive inside the optimization loop of solver-based MPC. To effectively use MPC, one would require the payload's influence not just at the current time step, but throughout the entire prediction horizon. Consequently, any domain characterized by partially observed dynamics will encounter analogous prediction errors and degraded control performance when used within the MPC framework.

3.2 Hyper Prediction Model

Continuing with the running example of a drone transporting a rope-suspended payload, one might initially treat the payload-induced forces as exogenous disturbances and entirely omit them from the system model. Although these forces originate from the payload's own swing dynamics and appear indirectly in the drone's measured states. If the payload state were observable, its equations of motion—and hence its influence on the drone's airframe could, in principle, be derived and incorporated into the model. Nevertheless, a model that relies only on the drone's instantaneous state cannot capture those effects. An approach is to compensate for unmodeled states by allowing the dynamics model parameters θ formulated in (1), to vary in response to those effects. However, most existing methods that address model inaccuracies, whether via adaptation laws [39], online parameter prediction [37, 9], or other forms of online learning [38, 13], assume that these parameters remain constant over the entire prediction horizon. We challenge this assumption by instead inferring the entire future trajectory of unmodeled states over the prediction horizon, conditioning not only on past observations but also on the expected control sequence. The impact of these unmodeled states is then captured as a trajectory of time-varying model parameters $\theta[t_c,\ t_c + t_p]$, yielding the Hyper-Prediction Model (HyperPM):

$$\theta[t_c,\ t_c + t_p] \ = \ \text{HyperPM}_{\Theta}\Big(x[t_c - t_h,\ t_c],\ u[t_c - t_h,\ t_c],\ \hat{u}[t_c,\ t_c + t_p]\Big), \tag{2}$$

which ingests the recent state history $x[t_c - t_o, t_c]$, the corresponding control history $u[t_c - t_o, t_c]$, and the planned control sequence $\hat{u}[t_c, t_c + t_p]$. The scheme of HyperPM is presented in Figure 1. The HyperPM model (i) extracts a latent representation of the unobserved unmodeled state from this history, (ii) propagates that representation forward under the planned actuation, and (iii) expresses the resulting influence of unmodeled states as a trajectory of time-varying parameters $\theta[t_c,\ t_c + t_p]$ across the prediction horizon. Additional implementation details are provided in the Appendix A.1.

It is important to emphasize that our approach goes beyond even oracle adaptation. As the experimental results show (see Section 4.2), even a perfect adaptive scheme is limited to simply propagating the obtained parameters as constant over the prediction horizon, e.g., assuming that the payload forces currently acting on the drone frame will remain the same in the future.

In summary, our method addresses scenarios where obtaining a fully Markovian representation of the system state is infeasible or computationally prohibitive. By approximating certain missing aspects of the dynamics within the model's parameter space, HyperPM improves the accuracy of long-horizon predictions by encoding unmodeled dynamics into trajectories of time-varying parameters.

3.3 Training procedure

An essential component of the proposed solution is the training procedure for HyperPM. Because in our proposed solution, we predict the trajectory of model parameters, we cannot rely on the most common approach to training parameter prediction models, i.e., evaluating the accuracy on one-step predictions. Instead, we propose to:

1. predict the trajectory of model parameters $\theta[t_c, t_c + t_p]$, based on the history of states $x[t_c - t_o, t_c]$, actions $u[t_c - t_o, t_c]$ and future controls $u[t_c, t_c + t_p]$,
2. roll out the time-varying model dynamics f_{θ_t} in discrete time steps, e.g., using Runge Kutta 4-th order integrator $\hat{x}_{t+dt} = f^{\text{RK4}}_{\theta_t}(\hat{x}_t, u_t)$,
3. compute the mean square prediction error between the obtained predicted trajectory of states $\hat{x}[t_c + dt, t_c + t_p]$, with the ones from the dataset $x[t_c + dt, t_c + t_p]$,
4. compute the loss as a sum of the prediction error and regularization terms,
5. back-propagate through time [40] the gradients of the loss function w.r.t. HyperPM weights and biases Θ.

By applying this training scheme, we ensure that every trainable variable is directly optimized to improve long-horizon prediction performance, which is crucial for effective MPC. A detailed description of the training procedure is provided in the Appendix A.1.

3.4 HyperMPC

The downstream application of HyperPM, which we focus on in this paper, is optimization-based Model Predictive Control [22, 41]. In this framework, an optimization algorithm computes trajectories of states and control actions that minimize the objective function, while satisfying the constraints stemming from the robot's dynamics and imposed on its states and control signals. In order to close the feedback loop, only the first control action is applied to the plant, and the whole optimization process is repeated at the next control interval.

The key aspect of the MPC paradigm is the use of a model capable of accurately predicting the evolution of the system over the optimization horizon. Typically, MPC uses a single constant model of the system [22]. However, as shown in [37], it is possible to infer a new model at each control interval, so that it can capture local variations in the dynamics of the system. In our approach, called HyperMPC, we propose to go further and exploit HyperPM to predict the trajectory of model parameters over the MPC horizon and increase predictive accuracy not only locally but also in the expected near future.

We obtain a horizon-long trajectory of dynamics-model parameters using HyperPM that is fed with both the recent history of observations and the planned control sequence generated by the previous MPC iteration. These predicted parameters are then injected into the nominal dynamics model and supplied to the MPC solver, which resolves the optimal-control problem for the current horizon. Only the first action of the refreshed solution is applied to the plant; the state and planned-action buffers slide forward, and the entire cycle repeats at the next control interval.

Crucially, the scheme maintains the same computational complexity as the primary dynamics model within the optimization loop, adding only a single forward pass through a lightweight neural network performed before solving the optimal control problem.

4 Experiments and results

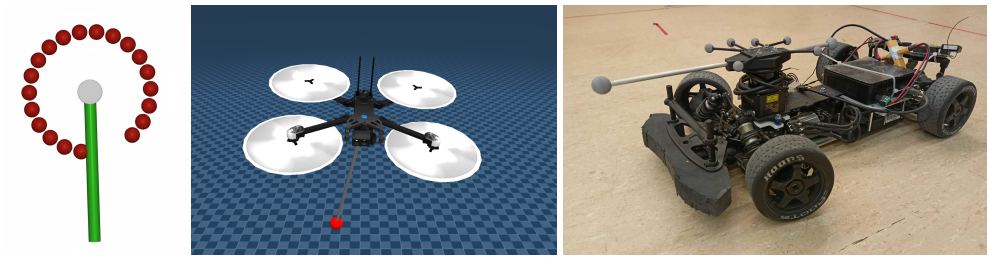

Figure 2: Experimental platforms used to validate the proposed method: (i) pendulum with backlash swing-up, (ii) drone with rope attached payload trajectory tracking, (iii) real-world autonomous F1TENTH [21] racing.

To evaluate the effectiveness of the proposed HyperPM and HyperMPC, we performed experiments on a set of challenging control tasks presented in Fig. 2. We show that forecasting the trajectories of dynamic model parameters using HyperPM enhances the prediction accuracy and improves the control performance in MPC. In each experiment, control performance is evaluated via the accumulation of MPC stage cost across episodes, highlighting differences between the modeling approaches.

4.1 Baselines

To establish baselines for our method, we selected a group of approaches to model the dynamics of the considered systems: $\mathbf{const}_s$ – a dynamics model with constant parameters, obtained using single-step prediction error minimization. $\mathbf{const}_l$ – a dynamics model with constant parameters, optimized using long sequences to minimize the error between simulated system states and ground-truth states across entire trajectories. $\mathbf{HD}_s$ – referred to as HyperDynamics, inspired by [9, 37]. This

is a dynamics model in which parameters are predicted by a neural network and held constant for the entire prediction horizon of the roll-out. It is trained using single-step prediction error minimization according to [9, 37]. **HD**$_l$ – the same architecture as HD$_s$, but trained using long sequences. **HyperPM** – our proposed method, as described in Section 3.

In addition, we tested extending the nominal dynamics of the system with a residual neural network [17, 18], $\dot{x} = f(x, u) + \mathrm{NN}(x, u)$, trained using long sequences, which we refer to as **res**. For drone experiments, the residual network predicts the residual forces acting on the drone frame.

To ensure a fair comparison with baselines, the training and architectural hyperparameters (e.g., neural network architecture, batch size, and learning rate) were optimized for each method by grid search, using the accuracy of the validation subset as the evaluation criterion. Full details of the training setup, including hyperparameters, exact MPC formulation, and dataset descriptions, are provided in the Appendix.

4.2 Simulated pendulum with backlash

An illustrative example of a system where all existing methods are constrained by the assumption of constant model parameters across the prediction horizon is a pendulum with backlash. Backlash refers to a clearance-induced dead zone causing temporary torque loss during motion reversal. To effectively plan a swing-up, the controller needs to anticipate the effect of unmodeled state, i.e., when the gears transfer torque, which static parameter models inherently fail to capture.

We evaluated the predictive performance of the models using a dedicated test set comprising sequences of 250 samples at 100 Hz. The results of this comparison are presented in Table 1. All models, except HD$_s$ and our HyperPM, performed comparably well in the long-horizon prediction task. The HD$_s$ model, trained on short sequences, was unable to produce coherent long-sequence predictions due to overfitting to single-step predictions. In contrast, only our method demonstrated significantly better performance, as it could recover the underlying state of the backlash and effectively compensate for it by leveraging time-varying dynamics model parameters, demonstrating the strength of forecasting parameter trajectories over the prediction horizon.

Table 1: Prediction and MPC performance for a pendulum with backlash

	Long-horizon prediction		**MPC performance in the swing-up task**		
Model	Error (↓)	Improvement [%] (↑)	Cost (mean ± std) (↓)	Success Rate [%] (↑)	Solver Failure Rate [%] (↓)
const$_s$	0.670	0.00	105.57 ± 10.96	0.0	5
const$_l$	0.588	12.23	106.17 ± 1.92	0.0	15
HD$_s$	28.766	-4195.94	115.54 ± 10.08	0.0	45
HD$_l$	0.626	6.51	85.92 ± 0.54	0.0	95
oracle adaptation	—	—	94.10 ± 1.24	0.0	0
res	0.618	7.77	—	—	—
HyperPM (ours)	**0.447**	**33.19**	**34.98 ± 7.21**	**100**	**0**

We tested the aforementioned dynamics models within an MPC framework to evaluate their impact on a downstream control task. We measured the total cost of each episode, the success rate (pendulum stabilized in upright position), , and the failure rate of the MPC numerical solver. The results are presented in Table 1, highlighting the advantages of time-varying parameter models in terms of both control performance and solver reliability. We omitted the reimplementation of the res model for MPC as it does not show any improvement over the const$_l$ model in the prediction task. To further illustrate the advantages of our method, we implemented oracle adaptation on top of a model that was trained on a dataset without backlash. Based on the simulator's internal state, we change the gear ratio parameter to 0 if the lever is not in contact with the gears. None of the baselines handled the unmodeled dynamics or successfully swung the pendulum. Even the oracle adaptation failed, as the controller couldn't accurately anticipate the unmodeled dynamics over the entire MPC horizon. In contrast, our methods, which incorporate time-varying parameter trajectories, successfully produced an accurate model and stabilized the pendulum in the upright position across all test cases. Moreover, our solutions were also the most stable solutions that did not cause any solver failures, which were common for the baseline models, especially HD.

4.3 Simulated drone with rope suspended payload

Drone trajectory tracking is a fundamental component of an aerial vehicle's autonomy stack and presents a particularly challenging modeling problem, especially when the drone is tasked with transporting a rope-suspended payload. In our setup, the rope was attached to the center of mass of the drone frame, allowing us to model the behavior using only forces. It is important to note that the state of the system does not include the position or velocity of the payload. This partial observability introduces complex, time-varying disturbances not evident from the drone's state alone, making accurate long-horizon prediction and control challenging for traditional MPC approaches.

In our experiments, we used two dynamics models, nominal [42], and a residual model in which a neural network predicts drone frame residual forces. Dynamics model parameters that can be changed by HD_l or HyperPM are the additional forces acting on a drone's frame. Additionally, we include an oracle-adaptation baseline that retrieves the exact payload-induced forces from the simulator and assumes that these forces remain constant throughout the prediction horizon. This represents an idealized scenario with access to perfect information, albeit with the limitations of time-invariant parameterization. We omitted the models trained using single-step prediction as they proved unstable inside the MPC loop and failed to deliver reliable control performance.

Nominal Model	Error (mean ± std) (↓)	Improvement [%] (↑)
const_l	0.0655 ± 0.132	—
HD_l	0.0252 ± 0.083	61.52
HyperPM (ours)	$\mathbf{0.0176} \pm 0.076$	**73.12**
Residual Model	**Error (mean ± std) (↓)**	**Improvement [%] (↑)**
const_l	0.0186 ± 0.058	—
HD_l	0.0184 ± 0.057	1.08
HyperPM (ours)	$\mathbf{0.0164} \pm 0.060$	**11.82**

Table 2: Long-horizon prediction error for drone with payload. Improvement is calculated w.r.t. const_l model with adequate dynamics model.

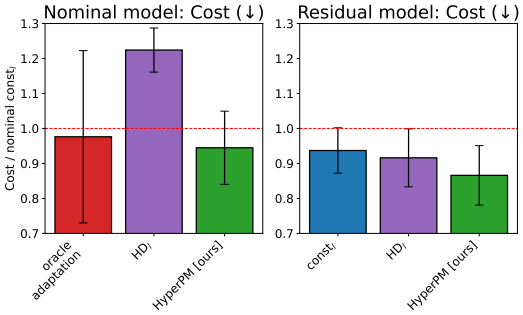

Figure 3: Comparison of modeling approaches drone on trajectories tracking task relative to nominal const_l.

The reported results are based on a setup that employs a 0.5 kg payload suspended from a 1 m rope attached to a quadrotor of mass 1.325 kg; additional rope-length ablation studies are presented in Appendix C.5. In the long-horizon prediction task (1 s horizon, 100-sample episodes), both HD_l and HyperPM substantially improve upon the nominal model. However, for residual dynamic modeling, only HyperPM delivers a clear gain, exceeding all other methods by more than 10%, underscoring its ability to anticipate complex, time-varying unmodeled effects. In the zero-length rope configuration (that is, the payload rigidly attached to the vehicle), we did not observe any measurable improvement, suggesting that HyperPM's benefit is primarily due to its ability to compensate for previously unmodeled dynamics associated with payload swinging. For the downstream task evaluation, we used the test split of the generated dataset and formulated a trajectory-tracking problem via MPC following the formulation in [42]. As presented in Figure 3, our approach surpasses both the oracle-adaptation baseline and HD_l in both the nominal and residual dynamics model settings. These results further confirm the advantage of modeling time-varying parameter trajectories in capturing complex, unobserved dynamics and translating that capability into superior control performance.

4.4 F1TENTH racing

Autonomous racing represents a unique and highly demanding testbed for evaluating and advancing control algorithms. Unlike conventional autonomous driving scenarios, racing pushes vehicles to operate at the edge of their dynamic limits, requiring algorithms to manage highly nonlinear dynamics and unrecoverable consequences of actions. The subsequent experiments use models trained on a 36-minute dataset collected using a real F1TENTH vehicle, with expert drivers manually navigat-

ing a variety of racetracks to span the entire operational envelope of the vehicle. Our model-based controller objective is to maximize track progress while keeping the vehicle within track boundaries.

Table 3: Long-horizon prediction error for F1TENTH car

Model	Error (mean ± std) (↓)	Improvement [%] (↑)
const_s	0.0240 ± 0.0581	0.00
const_l	0.0177 ± 0.0290	26.25
HD_s	0.0501 ± 0.1880	-108.75
HD_l	0.0137 ± 0.0239	42.92
res	$\mathbf{0.0091} \pm 0.0138$	**62.08**
HyperPM (ours)	$\underline{0.0123} \pm 0.0192$	<u>48.75</u>

Table 4: Comparison of MPC solve time, and parameter inference time on mobile AMD Ryzen 5 4600HS CPU.

Model	Solve time [ms] (↓) (mean ± std)	Inference time [ms] (↓) (mean ± std)
const_s	$\mathbf{10.76} \pm 2.24$	–
const_l	$\mathbf{10.53} \pm 1.97$	–
HD_s	13.42 ± 3.53	1.56 ± 0.13
HD_l	$\mathbf{10.65} \pm 2.05$	1.48 ± 0.14
res	23.73 ± 4.76	–
HyperPM (ours)	$\mathbf{10.61} \pm 1.76$	1.85 ± 0.12

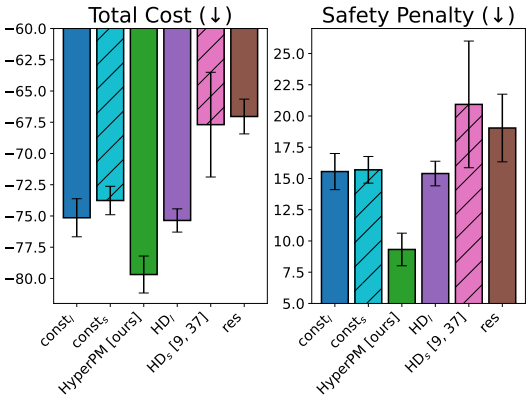

Figure 4: Comparison of modeling in the real-world F1TENTH racing task.

In our experimental analysis, we first evaluate the prediction performance of the models on long-horizon sequences drawn from the manual driving test set. In Table 3, we present the prediction errors for all evaluated modeling approaches and refer them to the most classical one – const_s. The best-performing model is the residual model, which utilizes the ability to model residual errors of the analytical model with a neural network. However, this approach takes advantage of significantly more parameters than the remaining methods. Among the equally sized models, the proposed HyperPM yields the lowest error, improving 48.75% over the model with constant parameters.

Finally, we evaluated these modeling approaches in real-world F1TENTH racing. We compared them in 30-second runs using two criteria: (i) total cost $\ell(x, u)$ and (ii) safety penalty $\ell_s(x, u)$, which mainly include track bounds violation soft constraint, both of them accumulated over the whole run. In Figure 4, we report these metrics averaged over five runs for each method. HyperPM outperforms all methods, including the residual network with the previously lowest prediction errors. We attribute this performance gap to the lack of generalization ability of a residual neural network to a different distribution of inputs w.r.t. the manual driving dataset. Regarding computational cost, Table 4 shows that feeding a parameter trajectory to the MPC leaves the solver's runtime unchanged.

5 Conclusion

In this paper, we introduced the Hyper Prediction Model (HyperPM) and its integration into the HyperMPC framework to enhance the predictive accuracy and control performance of systems with complex dynamics. Our method goes beyond a reactive paradigm of existing approaches [9, 37], adaptation schemes, as well as online learning in which model parameters are frozen across the prediction horizon. The proposed framework, by dynamically forecasting time-varying parameters of a dynamics model, addresses the limitations of both physics-derived and learning-based approaches, where state representation often is not sufficient to describe all important phenomena. The experimental evaluations demonstrated that HyperPM significantly improves the accuracy of the long-term prediction, with reductions of more than 35% in the prediction errors compared to the models with constant parameters. Moreover, the HyperMPC delivered superior performance in downstream control tasks, outperforming state-of-the-art methods while maintaining the same optimization loop complexity. Our results highlight the importance of incorporating anticipated actions into the prediction of model parameter trajectories, allowing MPC to proactively anticipate and compensate for previously unmodeled effects.

6 Limitations

The main limitation of the proposed approach is the need to have access to relatively accurate predictions of future control inputs. During training, the dataset provides exact future actions, whereas during deployment, we rely on expected action trajectories, typically generated by the controller or planner. These expected trajectories are inherently imperfect in representing true future control sequences, especially in long-horizon predictions, due to the closed-loop nature of control and the system's sensitivity to disturbances and modeling errors. To mitigate these issues, we proposed to randomize future control sequences using Gaussian noise during training. An open problem is how to measure and take into account the plausible distribution and uncertainty of the subsequent controls and their impact on the change of the dynamics model.

Another important limitation of the presented approach is the potential data mismatch. The data collection policy could be significantly different from the one induced by the controller with a learned dynamics model. Consequently, the state-action distribution encountered online may drift away from the training distribution, risking overfitting to the training data and degraded closed-loop performance. We believe that this issue can be mitigated by online learning that continuously adapts the HyperPM model to the data collected with HyperMPC.

Acknowledgments

This paper was supported by PUT internal grant 0214/SBAD/0252. The work of Piotr Kicki was supported by the Foundation for Polish Science (FNP). This paper was created with the use of the infrastructure of the Poznan Supercomputing and Networking Center.

References

[1] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[2] Z.-P. Jiang, T. Bian, and W. Gao. Learning-based control: A tutorial and some recent results. *Found. Trends® Syst. Control*, 8(3):176–284, 2020.

[3] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(Volume 5, 2022):411–444, 2022.

[4] G. Czechmanowski, J. Węgrzynowski, P. Kicki, and K. Walas. On learning racing policies with reinforcement learning, 2025.

[5] Y. Song and D. Scaramuzza. Policy Search for Model Predictive Control With Application to Agile Drone Flight. *IEEE Transactions on Robotics*, 38(4):2114–2130, Aug. 2022.

[6] Y. Song, A. Romero, M. Mueller, V. Koltun, and D. Scaramuzza. Reaching the Limit in Autonomous Racing: Optimal Control versus Reinforcement Learning. *Science Robotics*, 8 (82):eadg1462, Sept. 2023. arXiv:2310.10943 [cs].

[7] M. Krinner, A. Romero, L. Bauersfeld, M. Zeilinger, A. Carron, and D. Scaramuzza. MPCC++: Model Predictive Contouring Control for Time-Optimal Flight with Safety Constraints. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024.

[8] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger. Learning-Based Model Predictive Control for Autonomous Racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, Oct. 2019.

[9] J. Chrosniak, J. Ning, and M. Behl. Deep dynamics: Vehicle dynamics modeling with a physics-constrained neural network for autonomous racing. *IEEE Robotics and Automation Letters*, 9(6):5292–5297, 2024.

[10] M. Neunert, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli. Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds. *IEEE Robotics and Automation Letters*, 3(3):1458–1465, July 2018. Conference Name: IEEE Robotics and Automation Letters.

[11] 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)*, pages 4730–4737, 2019.

[12] L. Hewing, A. Liniger, and M. N. Zeilinger. Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars. In *2018 European Control Conference (ECC)*, pages 1341–1348, Limassol, June 2018. IEEE.

[13] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger. Learning-Based Model Predictive Control for Autonomous Racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, Oct. 2019.

[14] G. Torrente, E. Kaufmann, P. Foehn, and D. Scaramuzza. Data-Driven MPC for Quadrotors, Mar. 2021. arXiv:2102.05773 [cs].

[15] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, Aug. 2023.

[16] N. A. Spielberg, M. Brown, N. R. Kapania, J. C. Kegelman, and J. C. Gerdes. Neural network vehicle models for high-performance automated driving. *Science Robotics*, 4(28):eaaw1975, Mar. 2019.

[17] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh. KNODE-MPC: A Knowledge-based Data-driven Predictive Control Framework for Aerial Robots, Jan. 2022. arXiv:2109.04821 [cs, eess].

[18] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll. Real-time Neural-MPC: Deep Learning Model Predictive Control for Quadrotors and Agile Robotic Platforms. *IEEE Robotics and Automation Letters*, 8(4):2397–2404, Apr. 2023. arXiv:2203.07747 [cs, eess].

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi:10.1162/neco.1997.9.8.1735.

[20] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

[21] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, and M. Bertogna. F1/10: an open-source autonomous cyber-physical platform. *CoRR*, abs/1901.08567, 2019.

[22] M. Schwenzer, M. Ay, T. Bergs, and D. Abel. Review on model predictive control: an engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117 (5):1327–1349, Nov 2021. ISSN 1433-3015.

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

[24] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721, 2017.

[25] C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius. Sample-efficient cross-entropy method for real-time planning. In *Conference on Robot Learning 2020*, 2020.

[26] S. J. Wang, H. Zhu, and A. M. Johnson. Pay Attention to How You Drive: Safe and Adaptive Model-Based Reinforcement Learning for Off-Road Driving, Oct. 2023. arXiv:2310.08674 [cs].

[27] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018.

[28] W. Xiao, H. Xue, T. Tao, D. Kalaria, J. M. Dolan, and G. Shi. AnyCar to anywhere: Learning universal dynamics model for agile and adaptive mobility. *arXiv preprint arXiv:2409.15783*, 2024.

[29] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza. Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors. *IEEE Robotics and Automation Letters*, 7 (2):690–697, 2022. doi:10.1109/LRA.2021.3131690.

[30] T. Salzmann, J. Arrizabalaga, J. Andersson, M. Pavone, and M. Ryll. Learning for CasADi: Data-driven models in numerical optimization. In *Learning for Dynamics and Control Conference (L4DC)*, 2024.

[31] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl. Time-optimal race car driving using an online exact hessian based nonlinear mpc algorithm. In *2016 European Control Conference (ECC)*, pages 141–147, 2016.

[32] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015.

[33] N. Hansen, H. Su, and X. Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.

[34] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine. Solar: Deep structured representations for model-based reinforcement learning. In *International conference on machine learning*, pages 7444–7453. PMLR, 2019.

[35] C. Zhang, M. Ren, and R. Urtasun. Graph hypernetworks for neural architecture search. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[36] S. Hegde, Z. Huang, and G. S. Sukhatme. Hyperppo: A scalable method for finding small policies for robotic control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10821–10828, 2024.

[37] Z. Xian, S. Lal, H.-Y. Tung, E. A. Platanios, and K. Fragkiadaki. HyperDynamics: Meta-learning object and agent dynamics with hypernetworks. In *9th International Conference on Learning Representations (ICLR)*, 2021.

[38] Y. Tsuchiya, T. Balch, P. Drews, and G. Rosman. Online adaptation of learned vehicle dynamics model with meta-learning approach. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 802–809, 2024. doi:10.1109/IROS58592.2024.10801427.

[39] O. Gasparyan and H. Darbinyan. L1 adaptive control of quadcopters. In *2019 Computer Science and Information Technologies (CSIT)*, pages 96–100, 2019. doi:10.1109/CSITechnol.2019.8895217.

[40] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, Oct. 1990. Conference Name: Proceedings of the IEEE.

[41] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl. acados – a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 2021.

[42] B. B. Carlos, T. Sartor, A. Zanelli, G. Frison, W. Burgard, M. Diehl, and G. Oriolo. An efficient real-time nmpc for quadrotor position control under communication time-delay. In *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 982–989, 2020. doi:10.1109/ICARCV50220.2020.9305513.

[43] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[44] S. Bai, J. Z. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling, Apr. 2018. arXiv:1803.01271 [cs].

[45] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Dec. 2014. arXiv:1412.3555 [cs].

[46] P. Kicki, P. Liu, D. Tateo, H. Bou-Ammar, K. Walas, P. Skrzypczyński, and J. Peters. Fast Kinodynamic Planning on the Constraint Manifold With Deep Neural Networks. *IEEE Transactions on Robotics*, 40:277–297, 2024.

[47] M. Fey, J. E. Lenssen, F. Weichert, and H. Muller. SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 869–877, Salt Lake City, UT, June 2018. IEEE.

[48] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv:2407.17032*, 2024.

[49] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[50] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36:628 – 647, 2015.

[51] S. Srinivasan, S. N. Giles, and A. Liniger. A Holistic Motion Planning and Control Solution to Challenge a Professional Racecar Driver. *IEEE Robotics and Automation Letters*, 6(4): 7854–7860, Oct. 2021.

[52] A. J. S. I. J.M. Besselink and H. B. Pacejka. An improved Magic Formula/Swift tyre model that can handle inflation pressure changes. *Vehicle System Dynamics*, 48(sup1):337–352, 2010.

[53] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[54] R. Verschueren, S. De Bruyne, M. Zanon, J. V. Frasch, and M. Diehl. Towards time-optimal race car driving using nonlinear MPC in real-time. In *53rd IEEE Conference on Decision and Control*, pages 2505–2510, Los Angeles, CA, USA, Dec. 2014. IEEE.

[55] T. Novi, A. Liniger, R. Capitani, and C. Annicchiarico. Real-time control for at-limit handling driving on a predefined path. *Vehicle System Dynamics*, 58(7):1007–1036, July 2020.

[56] J. Balkwill. Chapter 3 - weight transfer and wheel loads. In J. Balkwill, editor, *Performance Vehicle Dynamics*, pages 53–73. Butterworth-Heinemann, 2018. ISBN 978-0-12-812693-6. doi:https://doi.org/10.1016/B978-0-12-812693-6.00003-1.

Appendix

A HyperPM and HyperMPC implementation

A.1 HyperPM Architecture

In general, the HyperPM can be implemented with any machine learning model that predicts the expected trajectory of model parameters based on the history of states and actions, and the expected trajectory of actions. In this paper, we utilize an architecture that is based on neural networks, and we present its scheme in Figure 5. The proposed architecture consists of four main components: (i) history encoder, (ii) expected controls preprocessor, (iii) causal parameters predictor, and (iv) parameters interpolation.

The main goal of the history encoder is to recover the underlying latent state of the system based on the history of states and actions. We examined multiple time-series encoders, including MLP, LSTM [43], and TCNN [44], but finally we selected a recurrent neural network with GRU [45] for its superior performance. The GRU's final hidden state is upscaled via a linear layer and is denoted as h_{t_c} in Figure 5.

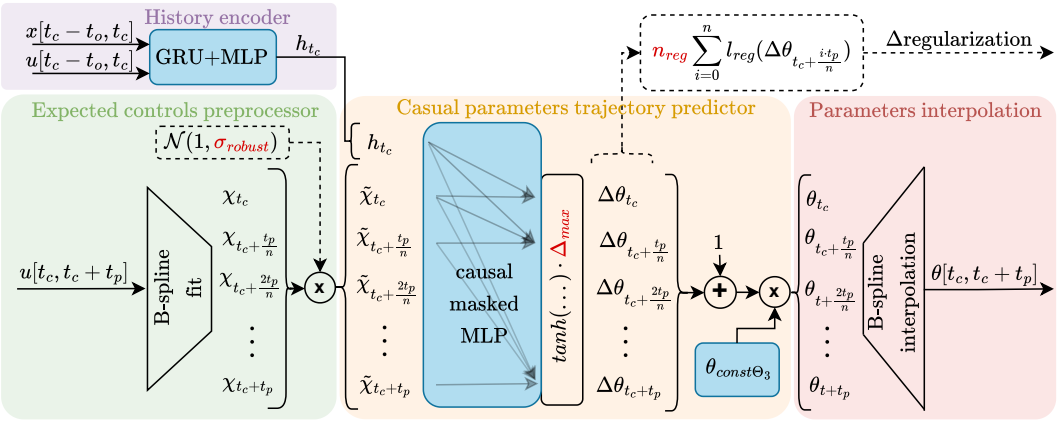

Figure 5: The architecture of Hyper Prediction Model with trainable parts marked in blue and hyper-parameters marked in red. The central part of the data processing is done in the space of the B-spline control points. With a dashed line, we denoted the elements that are active only in the training phase.

The goal of the expected controls preprocessor is to generate a compact representation of the trajectory of future controls $u[t_c, t_c + t_p]$. Inspired by [46, 47], we represent this sequence using control points of a B-spline curve fitted to the input data. Thanks to the curve fitting procedure, we can freely change the sampling time of the input controls without retraining the HyperPM. Moreover, it reduces the number of temporal samples that need to be processed (resulting in a smaller network) and makes the network less sensitive to individual actions and more focused on the general trend.

Before introducing the second part of the control preprocessor, let us note that the inputs to the HyperPM are very different in their nature. Indeed, the history of states and actions is known exactly, modulo measurement noise. Instead, the expected trajectory of future actions is much more undetermined. During training, the dataset provides us with the exact future actions; however, during deployment, we must rely on expected action trajectories, typically generated by the controller or planner. Therefore, to account for the potential distributional shift and to make the HyperPM aware that the future controls are not as certain as the history of observations, we inject noise into the representation of future actions. Specifically, during training, we multiply the B-spline control points by the noise drawn from a normal distribution with a mean of 1 and a standard deviation of σ_{robust}. We discovered that training with perturbed B-spline control points makes the network more robust and reduces overfitting (see Section E.2).

The following processing step in the HyperPM architecture is to generate changes in the parameters of the dynamics model over time, based on the representation of the state history h_t and expected future controls. We do so using a neural network that consists of two MLP layers. We enforce causality to prevent the model from learning spurious correlations or dataset-specific artifacts, such as driving style. This is achieved by masking the weights that connect future actions to the past parameters, i.e., setting them to zero. This ensures that the parameter trajectory generation remains causally dependent on the provided action sequences.

The representation of the changes of the parameters generated by the causal MLP is interpreted using two factors: scale Δ_{max} and bias θ_{const}. The scale Δ_{max} determines the maximum deviation from the nominal values of the model parameters θ_{const}. For example, if $\Delta_{\text{max}} = 0.5$, the final parameter trajectory will be bounded between 50% and 150% of θ_{const}. The bias θ_{const} is the parameter vector that minimizes the average prediction error over the entire dataset.

Finally, the obtained values of the dynamics parameters are interpolated using B-splines and constitute the prediction of the evolution of the model parameters $\theta[t_c, t_c + t_p]$. This design also enables one to extend the prediction horizon beyond the duration used during the training. By replacing $\Delta\theta_{t+tp}$ with zero, the trajectory smoothly transitions to a constant parameter value. The interpolated parameter vector can then be extended with the nominal parameters θ_{const} to achieve the desired prediction horizon.

Last but not least, we found it useful to regularize during training the changes in the parameters predicted by our solution using the L1 norm. This approach encourages HyperPM to predict changes in the parameters only if necessary and prefers to stay close to the nominal parameter values, as long as it does not affect the prediction accuracy too much. The regularization strength is controlled by n_{reg}.

A.2 Training procedure

An essential part of the proposed solution is the procedure that was used to train HyperPM. Because in our proposed solution, we predict the trajectory of model parameters, we cannot utilize the most common approach to training parameter prediction models, i.e., evaluating the accuracy on one-step predictions. Instead, we propose to:

1. predict the trajectory of model parameters $\theta[t_c, t_c + t_p]$, based on the history of states $x[t_c - t_o, t_c]$ and controls $u[t_c - t_o, t_c]$, and expected future controls $u[t_c, t_c + t_p]$,
2. discretize $\theta[t_c, t_c + t_p]$ in subsequent time steps $\theta_{t_c}, \theta_{t_c+dt}, \ldots, \theta_{t_c+t_p-dt}$,
3. roll out the time-varying model dynamics f_{θ_t} in discrete time steps using a 4th-order Runge Kutta integration scheme $\hat{x}_{t+dt} = f^{\text{RK4}}_{\theta_t}(\hat{x}_t, u_t)$,
4. compute the discretized prediction error (see eqn. 1 in the main paper) between the obtained predicted trajectory of states $\hat{x}[t_c{+}dt, t_c{+}t_p]$ and the ones from the dataset $x[t_c{+}dt, t_c{+}t_p]$,
5. compute the loss as a sum of the prediction error and Δregularization term introduced in Section A.1,
6. back-propagate through time [40] the gradients of the loss function w.r.t. HyperPM weights.

This procedure is schematically depicted in Figure 6. By applying this training scheme, we ensure that every trainable variable is optimized directly to improve long-horizon prediction, which is crucial for MPC. Moreover, the proposed approach has no restrictions regarding the dynamics model structure. Thus, we use the same approach to train all baseline approaches and show that optimization w.r.t. long-term loss and back-propagation through time is beneficial in terms of prediction accuracy and sometimes even necessary to obtain stable predictions in longer horizons.

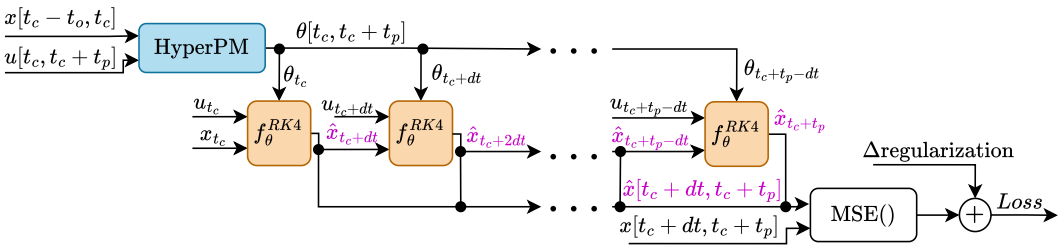

Figure 6: The Hyper Prediction Model is trained to predict trajectories of parameters of the dynamic model that minimize, over the prediction horizon, the mean squared error between simulated system states (pink) and the actual ones.

A.3 HyperMPC

The downstream application of HyperPM, which we focus on in this paper, is optimization-based Model Predictive Control [22, 41]. In this framework, an optimization algorithm computes the trajectory of states and control actions, which minimize the objective function, while satisfying the constraints stemming from the robot's dynamics and imposed on its states and control signals. In order to close the feedback loop, only the first action is forwarded to the plant, and the whole process is repeated at the next control interval.

The key aspect of the MPC paradigm is a model that is able to accurately predict the evolution of the system over the optimization horizon. Typically, MPC utilizes a single constant model of the system [22]. However, as shown in [37], a new model can be inferred at each time step, so that it can capture local changes in the system's dynamics. In our approach, called HyperMPC, we propose to go beyond this and exploit HyperPM to predict the trajectory of model parameters over the MPC horizon and increase its accuracy not only locally but also in the expected near future. In this way, not only does MPC get a better prediction model, but HyperPM may also improve the accuracy of its predictions by exploiting the knowledge of the actions planned by the MPC in the previous iteration. The pseudocode of the HyperMPC algorithm is presented in Algorithm 1.

B Pendulum experiment

B.1 Pendulum simulation

We evaluated the proposed method in a pendulum environment similar to the one of the Gymnasium library [48] extended with backlash, a phenomenon commonly encountered in real-world robotics scenarios. The evaluation was carried out using a MuJoCo simulator [49], where the backlash was implemented as a joint limit, following the approach suggested in the user manual. To better visualize the backlash phenomenon, we added red dots attached to the virtual joint that simulates the backlash. The visualization of the environment is presented in Figure 7. The pendulum system's dynamics model was derived from the Gymnasium implementation, with an added parameter to account for the gear ratio. The complete set of parameters describing the system's dynamics includes mass, length, viscous friction, Coulomb friction, and gear ratio. The values of these parameters and

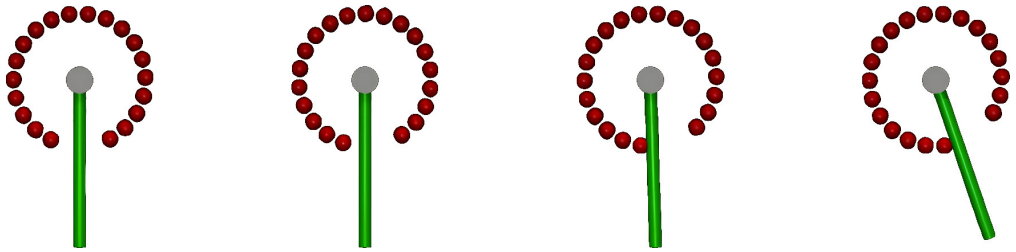

Figure 7: Visualization of the pendulum environment with backlash. Red dots are attached to the virtual joint, simulating the backlash and illustrating the slack between gears.

Algorithm 1 HyperMPC Framework

Input:
Dynamics model: $f_\theta(x, u)$
Objective function ℓ and the terminal objective ℓ_f
History of system states: $\mathcal{H} = \{x_t\}_{t=t_c-t_o}^{t_c-dt}$
Prediction horizon: t_p
Current state: x_{t_c}
Expected controls: $\mathcal{C} = \{\hat{u}_t\}_{t=t_c}^{t_c+t_p-dt}$
HyperPM: HyperPM$(\mathcal{H}, \mathcal{C}) \rightarrow \{\theta_t\}_{t=t_c}^{t_c+t_p-dt}$

Output: Control u_{t_c}

1: **Predict Trajectory of Model Parameters**

$$\{\theta_t\}_{t=t_c}^{t_c+t_p-dt} \leftarrow \text{HyperPM}(\mathcal{H}, \mathcal{C})$$

2: **Solve Optimization Problem**

$$\begin{aligned} \underset{\{u_t\}_{t=t_c}^{t_c+t_p-dt}}{\arg\min} \sum_{t=t_c}^{t_c+t_p} \ell(x_t, u_t) + \ell_f(x_{t_c+t_p}) \\ \text{subject to:} \quad x_{t+dt} = f_{\theta_t}^{\text{RK4}}(x_t, u_t), \\ x_t \in \mathcal{X}, \\ u_t \in \mathcal{U}, \end{aligned}$$

3: **Apply Control**

$$u_{t_c} \leftarrow u_{t_c}^*$$

4: **Update History, State and Expected actions**

$$\begin{aligned} \mathcal{H} &\leftarrow \mathcal{H} \cup \{x_{t_c}\} \\ x_t &\leftarrow x_{t+dt} \\ \mathcal{C} &\leftarrow u_{t+dt}^* \forall t \in [t_c, t_c + t_p - dt] \\ t_c &\leftarrow t_c + dt \end{aligned}$$

5: **Repeat: Go to Step 1**

the simulator settings are presented in Table 5. Furthermore, implemented models were validated using the simulated pendulum without backlash to ensure that they could achieve zero training and validation error.

Table 5: Pendulum simulation setting

Parameter	Value
integrator	RK4
step size	10^{-5}s
link mass	1.0
link length	0.5
link diameter	0.02
damping	0.05
frictionloss	0.0
gear-ratio	1.0
backlash	$30\,\text{deg}$

We evaluated them on 20 episodes with initial position and velocities drawn from the range $[-\frac{\pi}{2}; \frac{\pi}{2}] \times [-0.5, 0.5]$ range.

B.2 Dataset

To create the dataset, we defined a sampling procedure for the control signal (applied torque) and collected 360 ten-second episodes. These episodes were split into training, validation, and test datasets in a 70%, 20%, and 10% ratio, respectively. The control signal was generated by sampling 15 random B-spline control points from a uniform distribution $[-2, 2]$, which were then interpolated. The resulting signal was clipped to the range $[-1, 1]$, making it impossible to swing up the pendulum directly. This sampling procedure enriched the dataset with saturated control inputs.

B.3 Models training

For training models utilizing longer sequence lengths, a prediction horizon of 250 samples was chosen, with a sampling rate of 100 Hz. All methods and models were trained for 2000 epochs, which was sufficient to observe convergence. Before feeding the joint angle to the neural network, it was encoded using $sin(q)$ and $cos(q)$ functions. Furthermore, prior to being fed into the neural network, the angular velocities were normalized by the maximum value of the training dataset.

B.4 Pendulum MPC design

The used cost function is the negated reward from the Gymnasium environment, extended with terminal cost and control derivative penalization in terms of matrix R:

$$\begin{aligned} x &= [q, \dot{q}, \tau] \\ x_{ref} &= [\pi, 0, 0] \\ u &= \dot{\tau} \\ \ell(x, u) &= (x - x_{ref})Q(x - x_{ref})^T + uRu^T \\ \ell_f(x) &= 10 \cdot (x - x_{ref})Q(x - x_{ref})^T \end{aligned} \tag{3}$$

where Q is a diagonal matrix with weights 1, 0.1, 0.001, and R set to $10e^{-8}$. The state of the pendulum is defined by the angle q, where $q = \pi$ represents the upright position, the angular velocity $\dot{q}$, and the torque applied to the joint τ. We follow a common MPC design practice, and we control the time derivative of the torque to smooth out the control actions.

C Drone

C.1 Drone simulation

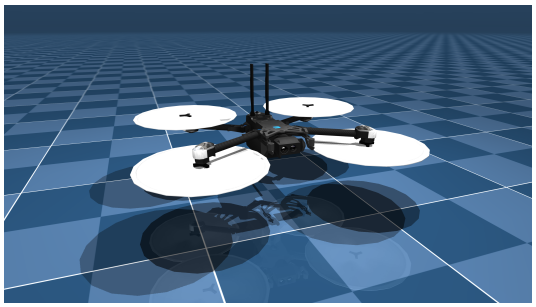

Figure 8: Base drone model Skydio X2 additionally extended with rope attached payload.

The drone environment was built in MuJoCo using the publicly available *Skydio X2* model from the MuJoCo Menagerie repository.[1] To introduce an unmodeled degree of freedom, we attach to the center of mass of the vehicle a massless tendon with a point mass payload placed at its end.

[1] https://github.com/google-deepmind/mujoco_menagerie/tree/main/skydio_x2

C.2 Dataset

Reference trajectories of orientation and altitude were randomly sampled and tracked with a cascaded PID controller. For each length of the rope, we recorded 300 episodes of $20\,\mathrm{s}$ in $100\,\mathrm{Hz}$ and split the data $70\%/20\%/10\%$ into training, validation, and test sets.

C.3 Model training

Similarly as for the pendulum, all models were optimized with a multistep prediction loss computed over 100-step roll-outs. The nominal dynamics of the drone were represented with rigid body dynamics as proposed in [42].

C.4 MPC formulation

We formulate a trajectory tracking problem as in [42], penalizing deviations from the reference position and attitude with quadratic weights $\mathrm{diag}(100, 100, 100, 10, 10, 10, 0.001, 0.001, 0.001, 1, 1, 1)$ in $\langle x, y, z, q_w, q_x, q_y, q_z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z \rangle$, and a control cost of 0.01 on individual motor thrusts. To obtain a similar distribution of data in the training and downstream tasks, we used the test set as the reference trajectory on which the control performance is evaluated. The MPC used a prediction horizon of 1.6 s with a control frequency equal to 100Hz.

C.5 Effect of rope length

In this experiment, we evaluate the hypothesis that our method can anticipate the dynamics of unmodeled system states and partially compensate its effects on the observed states. To this end, we construct four datasets with progressively increasing rope lengths, thereby amplifying the influence of the unmodeled state—the swing of the rope-suspended payload. We employ a residual dynamics model to compensate for effects that can be inferred from the system's instantaneous state. Figure 9 reports the predictive performance across all modeling approaches. When the rope length is zero, i.e., the payload is rigidly attached to the drone frame, our method performs comparably to a model with constant parameters. However, as the influence of the unmodeled state increases, the advantage of our method, HyperPM, over the baselines, const_l and HD_l, becomes evident. These results indicate that our method leverages time-varying parameters to recover hidden system states and predict their evolution, resulting in lower prediction errors.

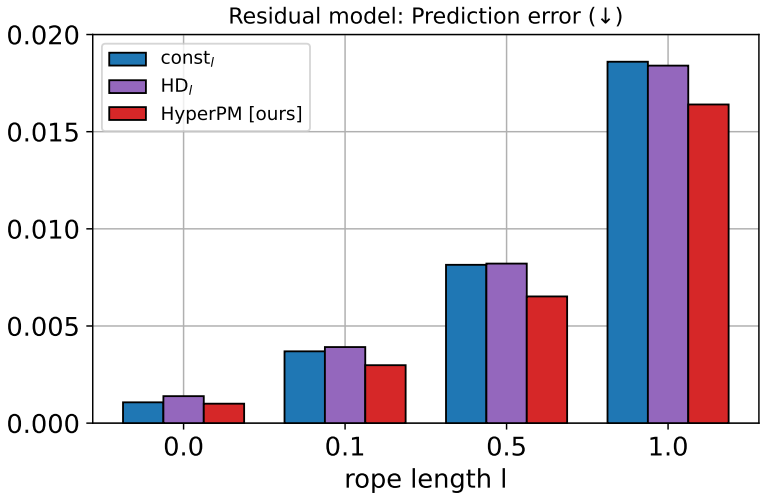

Figure 9: Comparison of prediction performance on datasets with different rope attachment lengths.

D F1TENTH experiment

D.1 Experimental setup

In this paper, we tackle the autonomous racing task using the real F1TENTH vehicle. The experimental vehicle, an Xray GTXE'22, is a 1/8th scale RC car with a four-wheel drive, powered by a single motor. To ensure a fair comparison between methods, we race in an empty space with a virtually defined racetrack. This ensures that the track is exactly the same for all methods and experiments, as it does not depend on the manual setting. Moreover, for accurate pose and velocity measurements during racing, we rely on the high-precision OptiTrack motion capture system. Our experimental setup is presented in Figure 10.

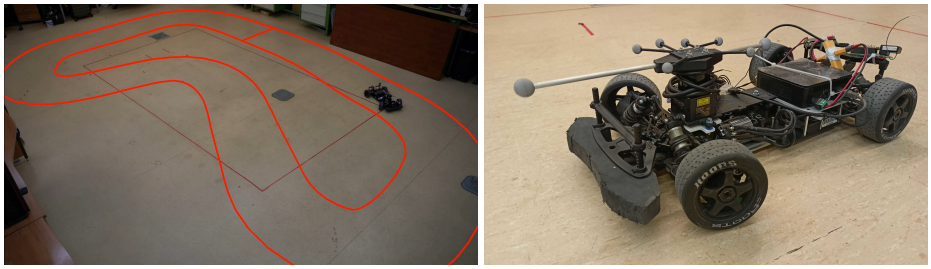

Figure 10: The experimental setup for the F1TENTH racing task. The left image presents the virtual track in the laboratory, while the right one is a close-up of the vehicle we used.

D.2 Analytical model

For the purpose of car racing, we utilize the dynamics model of the single-track vehicle, as in [50, 51], extended with the more accurate tire model MF6.1 [52] referred to in the following section as an analytical model. Given such dynamics, the state vector is expressed as

$$x = [v_x, v_y, r]^T, \tag{4}$$

where v_x, v_y denotes longitudinal and lateral velocities, and r is the yaw rate. The control input $u = [\delta, \omega]^T$ is composed of the steering angle δ and the wheel speed scaled by the tire radius ω. We assume that some parameters are known; this includes mass m, gravity constant g, and wheel base L.

D.3 Dataset

To train and evaluate all considered methods, we collected a dataset using the physical setup introduced in Section D.1. The dataset was collected by an expert human driving on different tracks, with varying ranges of slip angle and slip ratio, trying to cover the entire envelope of the operation region of the vehicle state space. During manual racing, we captured relevant data of the internal state of the car, such as motor RPM, motor q-axis current, and steering angle, as well as the information about its position and orientation using OptiTrack. Moreover, we obtained the velocity in the vehicle frame by differentiating the position and orientation measurements using the Savitzky-Golay filter [53]. All of these data were collected synchronously at 100Hz. In total, we recorded 36 minutes of driving. The recorded dataset was then split into disjoint training, validation, and test sets in the proportion 70/20/10%. To generate sequences, we applied a sliding window with a stride of 5 samples. Each window consisted of 135 samples, 50 of which were allocated to the observation window and 85 to the prediction horizon.

D.4 F1TENTH MPC design

In order to compare prediction models in a downstream application, we formulate the autonomous vehicle racing task by extending the primary system state (4) by Frenet position, following [54,

51, 55], represented by distance along the center-line of the track s, distance perpendicular to the center-line n and heading difference μ, and control signals, resulting in

$$x = [s, n, \mu, v_x, v_y, r, \omega, \delta]^T. \tag{5}$$

To capture the actuator dynamics absent in the primary model $u = [\omega, \delta]^T$, the control input was reformulated as $u = [\omega_{ref}, \delta_{ref}]^T$ and modeled as a first-order linear system. To compensate for the measurement delay and actuator delay, we constrain adequate first control actions in such a way as to restrict the change of u, taking into account the recent history of applied actions.

The controller's task is to minimize a cost function that is divided into two components: $\ell(x, u) = \ell_s(x, u) + \ell_p(x, u)$. The primary objective $\ell_p(x)$ is defined as the negative distance traveled along the centerline of the racetrack. Additionally, the safety objective $\ell_s(x, u)$ is composed of three components: $\ell_s(x, u) = \ell_{track}(x) + \ell_{ctrl}(x, u) + \ell_{slip}(x)$. The track-bound cost $\ell_{track}(x)$ is formulated as a barrier function $p(h(x))$ to incorporate the constraint $h(x) \geq 0$ into the cost function, where $p(h(x)) = \log(1 + \exp(-\alpha h(x)))$ with $\alpha = 200$, following a similar approach to [7]. Since excessive slip may lead to unrecoverable situations, a cost penalizing large wheel slip and slip angles is included: $\ell_{slip}(x) = q_\beta \beta_{reg} + q_{slong}(\omega - v_x)^2$, where β_{reg} is defined as in [8, 51]. To emphasize the predictive behavior of the vehicle, a cost based on the smoothness of the control signal is defined as $\ell_{ctrl}(x, u) = q_u(\omega - \omega_{ref})^2 + q_u(\delta - \delta_{ref})^2$.

For a fair comparison, the MPC was tuned to the const$_l$ model baseline and held constant for all models we considered, with $q_u = 2.0$, $q_{slong} = 0.1$, $q_\beta = 1$. We compare the performance of the MPC with different dynamics models by using the same cost function as the MPC stage cost $\ell(x, u)$. For the sake of safety and to evaluate the generalization abilities of the dynamics models, we set the horizon of MPC to 80 shooting nodes with a frequency of 33.3Hz, which results in a 2.67 s horizon, while during training time, a horizon of 0.85 s (at 100 Hz) was used.

E Ablation studies

E.1 Impact of the expected future trajectories

One of the key components of the proposed HyperPM is the exploitation of the expected future trajectories in the process of predicting the time-varying dynamics model parameters. Intuitively, future actions may have a significant impact on the behavior of the unmodeled dynamics that we aim to capture with the predicted parameters. In this experiment, we want to evaluate this hypothesis by comparing the original HyperPM with the one that does not condition on the expected trajectory of actions (HyperPM - EAC). Moreover, for a better perspective, we also report the results obtained by the HyperDynamics approach [37], which not only does not condition on the future controls but also predicts only a single parameter for the entire MPC horizon (as opposed to HyperPM - EAC).

The results of this comparison are presented in Table 6. One can see that removing the dependence on the expected actions results in worse prediction performance for both the Pendulum and F1TENTH. However, the accuracy decrease is different for each system. In the case of Pendulum with backlash, we observe only a slight deterioration of HyperPM - EAC, when compared to HD$_l$. This suggests that in this case, the main source of the HyperPM success lies in the ability to predict the parameter's trajectories, but still knowing the planned future controls gives some advantage. This advantage is even clearer in the case of F1TENTH racing, in which we observe that without the expected future actions HyperPM is less accurate than the HD$_l$. This suggests that the ability to predict time-varying parameters may be misused if there is not enough information about the expected evolution of the system. These results confirm the importance of conditioning the parameters prediction on the trajectory of expected actions.

E.2 Impact of the σ_{robust} hyperparameter

In the proposed HyperPM architecture, we introduced a σ_{robust} hyperparameter, which controls the amount of noise added to the representation of the expected future controls during training. The

Table 6: Long-horizon prediction errors of HyperPM and HyperPM without conditioning on expected actions (HyperPM - EAC), related to the HyperDynamics HD_l.

	Pendulum with backlash		F1TENTH racing	
Model	Error	Deterioration [%]	Error	Deterioration [%]
HyperPM (ours)	**0.447**	-	**0.0123**	-
HyperPM - EAC	0.475	5.89	0.0147	16.32
HD_l	0.626	28.59	0.0137	10.22

rationale behind this is to reduce the focus of the HyperPM on the true controls from the dataset that are observed during the training phase. Instead, we want to make HyperPM robust to some variability in the expected controls, which is expected during the deployment phase, when future controls are predicted with MPC. In Table 7 we present the prediction losses obtained for 3 different choices of the σ_{robust}. As expected, the smaller the noise, the more accurate the HyperPM on the training set. However, the best results on the validation set are obtained for a moderate value of $\sigma_{\text{robust}} = 0.2$.

Table 7: Prediction loss in the F1TENTH task for different σ_{robust} values

σ_{robust}	Train	Validation
0.5	0.0084228	0.01168
0.2	0.0070832	**0.010384**
0.05	0.0062105	0.010678

E.3 Interpretability

An important benefit of choosing a physically derived dynamics model structure is the ability to interpret variations in parameters, which can suggest physical phenomena worth including in the model. For instance, the analytical dynamics model of a F1TENTH car, which we used in the previous experiments, lacks the capability to represent load transfer during braking and acceleration. To evaluate whether HyperPM captures this unmodeled effect, we compare the load-transfer predicted by the HyperPM with the load-transfer calculated based on the physical model of this phenomenon.

Following [56], we represent the normal force acting on the front wheel by

$$F_{zf} = \frac{mgl_r}{L} - \frac{mh_{cg}a_x}{L}, \tag{6}$$

where l_r is the distance from the center of gravity to the rear axle, m is the vehicle mass, h_{cg} is the center of gravity height, a_x is the longitudinal acceleration, and L is the wheelbase.

In turn, the HyperPM considers only a first part of (6), i.e., $F_{zf} = \frac{mgl_r}{L}$. However, by predicting a time-varying l_r parameter, it can effectively capture the physical effect of varying normal force F_{zf}, which is not considered by the single-track model.

In Figure 11, we compare the front tire normal forces F_{zf} computed with (6) (black) and the ones predicted with HyperPM (red). A strong correlation between these predictions suggests that the HyperPM captures an unmodeled effect of the load transfer in an interpretable way, i.e., by virtually changing the position of the center of mass.

E.4 Hyperparameters used in the experimental evaluation

The following tables summarize the hyperparameters used for all models in our experimental evaluation. Tables 8–11 present the details of the pendulum swing-up, while Tables 15–18 provide the corresponding values for the F1TENTH racing.

All models were trained using the AdamW optimizer with a learning rate of 5×10^{-4}. For models trained on long sequences, the gradient clipping by value was set to 5×10^{-4}. Additionally, all relevant models utilized a GRU-based time series encoder.

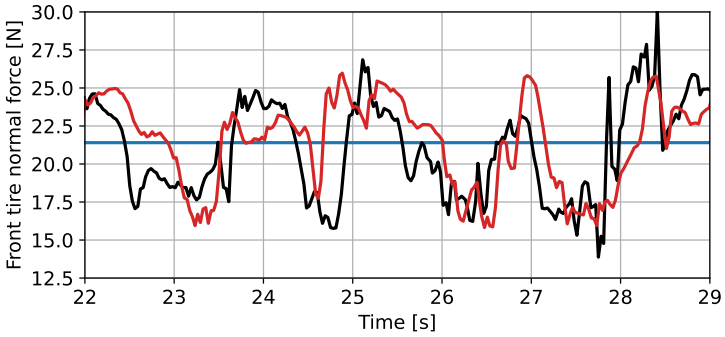

Figure 11: Front tire normal force calculated using different methods. The blue line represents the front tire normal force calculated from the identified const_l model. The black line shows the force calculated using (6), while the red line represents the force obtained from HyperPM.

Table 8: Hyperparameters for HyperPM - Pendulum

Hyperparameter	Value
B-spline control points	25
GRU hidden size	4
Causal MLP features per control point	8
σ_{robust}	0.2
n_{reg}	0.05
Δ_{max}	1.0
Batch size	128
Activation	Tanh

Table 9: Hyperparameters for HD_l - Pendulum

Hyperparameter	Value
GRU hidden size	32
MLP size	2 layers, 128 hidden size
Batch size	128
Activation	ReLU

Table 10: Hyperparameters for HD_s - Pendulum

Hyperparameter	Value
GRU hidden size	4
MLP size	2 layers, 512 hidden size
Batch size	256
Activation	ReLU

Table 11: Hyperparameters for res - Pendulum

Hyperparameter	Value
Residual NN size	Single hidden layer, 16 neurons
Activation	Tanh

Table 12: Hyperparameters for HyperPM residual as well as nominal dynamics - Drone

Hyperparameter	Value
B-spline control points	8
GRU hidden size	16
Causal MLP features per control point	8
σ_{robust}	0.05
Δ_{max}	1.0
Batch size	128
Activation	Tanh

Table 13: Hyperparameters for HD_l residual as well as nominal dynamics - Drone

Hyperparameter	Value
GRU hidden size	16
MLP size	2 layers, 128 hidden size
Batch size	128
Activation	ReLU

Table 14: Hyperparameters for Residual Model - Drone

Hyperparameter	Value
Residual NN size	Single hidden layer, 32 neurons
Activation	Tanh

Table 15: Hyperparameters for HyperPM - F1TENTH

Hyperparameter	Value
B-spline control points	8
GRU hidden size	16
MLP features per control point	32
σ_{robust}	0.5
n_{reg}	0.2
Δ_{max}	0.5
Batch size	128

Table 16: Hyperparameters for HD_l - F1TENTH

Hyperparameter	Value
GRU hidden size	16
MLP size	2 layers, 128 hidden size
Batch size	256

Table 17: Hyperparameters for HD_s - F1TENTH

Hyperparameter	Value
GRU hidden size	16
MLP size	2 layers, 128 hidden size
Batch size	256

Table 18: Hyperparameters for res - F1TENTH

Hyperparameter	Value
Residual NN size	Single hidden layer, 64 neurons
Activation	Tanh