Fine-Tuning of Neural Network Approximate MPC without Retraining via Bayesian Optimization

Henrik Hose[1]   **Paul Brunzema**[1]   **Alexander von Rohr**[2]   **Alexander Gräfe**[1]
Angela P. Schoellig[2]   **Sebastian Trimpe**[1]

[1]Institute for Data Science in Mechanical Engineering (DSME), RWTH Aachen, Germany
[2]Learning Systems and Robotics Lab (LSY) and Munich Institute for Robotics and Machine Intelligence (MIRMI), TU Munich, Germany

Abstract: Approximate model-predictive control (AMPC) aims to imitate an MPC's behavior with a neural network, removing the need to solve an expensive optimization problem at runtime. However, during deployment, the parameters of the underlying MPC must usually be fine-tuned. This often renders AMPC impractical due to the need to repeatedly generate a new dataset and retrain the neural network. Recent work addresses this problem by adapting AMPC without retraining using approximated sensitivities of the MPC's optimization problem. However, currently, this adaption must be done by hand, which is labor-intensive and can be unintuitive for high-dimensional systems. To solve this issue, we propose using Bayesian optimization to tune the parameters of AMPC policies based on experimental data. By combining model-based control with direct and local learning, our approach achieves superior performance to nominal AMPC on hardware, with minimal experimentation. This allows automatic and data-efficient adaptation of AMPC to new system instances and fine-tuning to cost functions that are difficult to implement in MPC. We demonstrate the proposed method in hardware experiments for the swing-up maneuver fo a cartpole and yaw control of an under-actuated balancing unicycle robot, a challenging control problem.

Keywords: Model Predictive Control, Bayesian Optimization, Imitation Learning, Neural Network Control

1 Introduction

Model-predictive control (MPC) is a modern optimization-based control method for nonlinear systems that provides theoretical guarantees for constraint satisfaction and stability [1]. MPC has achieved remarkable results in practical robotics applications [2, 3, 4, 5, 6]. However, MPC requires solving an optimization problem periodically at runtime, making real-world deployment unfeasible for fast dynamical systems, even when dealing with mildly complicated dynamics, cost functions, and constraints. Approximate MPC (AMPC) is one way to solve this challenge: A fast-to-evaluate function approximator, typically a neural network (NN), is trained in an imitation-learning fashion on a large dataset of samples from the MPC, i.e., a dataset of states and optimal actions (see [7] for a recent survey on AMPC with NNs). Computing this large dataset can be done offline and in parallel on large computation clusters, but it can easily take tens of thousands of core-hours, especially for high-dimensional systems. This poses a problem when deploying AMPC in practice, as often multiple iterations over parameter values of the MPC in the model, cost functions, and constraint sets are required to achieve the desired real-world control performance. For every parameter change, the entire dataset must be regenerated. In our opinion, this is one of the reasons why applications of AMPC in fast-moving dynamical and robotics systems are rare. In a recent paper [8], it is shown
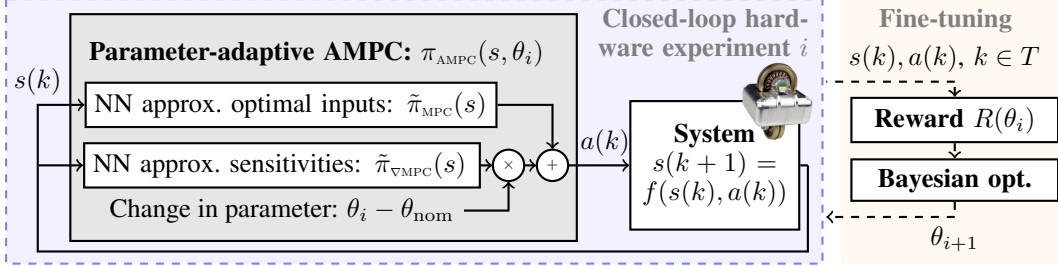
Figure 1: Automatic tuning of parameter-adaptive AMPC with Bayesian optimization (BO). *Approximate nominal MPC inputs are linearly adapted by approximate sensitivities to deviations from nominal parameters θ. Parameters are directly tuned with a few experiments using BO, maximizing a closed-loop reward R on the real system.*

that a second neural network approximating the gradients of the optimal actions with respect to parameters of the MPC problem (also known as sensitivities of the MPC optimization problem) can be used to adapt an AMPC to changes in system parameters online – without recomputing large datasets or training neural networks. However, the current approach requires the parameters to be chosen by hand, making deployment of the AMPC to multiple system instances labour intensive.

Contribution. We use Bayesian optimization (BO) [9] to find optimal parameters for the recently proposed parameter-adaptive AMPC [8] based on closed-loop experiments (Fig. 1) in a data-efficient manner. The reward given to the BO reflects our true control objective and allows us to optimize the AMPC in this direction. The true objective can be sparse in states and time, or binary such as success or failure, which is difficult to implement in MPC. We show the effectiveness of our method in two hardware experiments: a common cartpole system and yaw control of a balancing reaction wheel unicycle robot (Fig. 2). The latter problem is *linearly* uncontrollable, making it a challenging use case for nonlinear AMPC. On both systems, we achieve stabilization and disturbance rejection in the closed loop after only a handful of experiments. Automatic tuning is able to overcome model mismatch between simulation and hardware with only 20 experiments to achieve desirable performance. In summary, our contributions are as follows:

1. Automatic tuning of a parameter-adaptive AMPC to new system instances in a direct, data-driven manner using BO, and without retraining neural networks.
2. Fine-tuning of the parameter-adaptive AMPC to new reward functions that are difficult to implement with classic MPC (e.g., due to sparsity).
3. Experimental validation on two unstable systems: a classic cartpole swing-up and stabilization benchmark and yaw control of an underactuated reaction wheel unicycle robot.

A video of our experiments is available under https://youtu.be/R96QpD5iusI.

2   Related Work

The method developed herein draws on two active research areas in robot learning: approximating MPC (i.e., imitation learning from MPC) and BO for controller tuning. We review related works in each of these, highlighting how we combine the two in a new way.

Approximate MPC. AMPC is a technique that finds a fast-to-evaluate but approximate explicit representation of a MPC through NN training [7]. Unlike other explicit representations, for example, memory-intensive look-up tables [10, 11] or explicit MPC for linear systems [12, 13], AMPC is applicable to general, nonlinear systems and requires only small NNs [14, 15]. These networks can be evaluated on small microcontrollers in milliseconds [8]. Nonetheless, very few publications [14, 15] apply AMPC in real-world robotics tasks, which we attribute to a significant practical issue: Even though the dataset synthesis is performed offline, it can take tens of thousands of CPU core hours. Additional computation time is required for training the NN approximation. In classic AMPC, a

new dataset must be computed and a new NN trained for every tuning iteration. This is not practical in many applications as system instances have slightly different physical parameters, for example, masses, lengths, or friction parameters [16].

The first way to overcome this issue is to approximate a MPC that is robust against parameter uncertainties [14], leading, however, to conservatism and requiring a-priori known uncertainty bounds. Alternatively, a nominal AMPC can be used to warm-start an optimizer online in hopes to speed up computations [17, 18]; this, however, is not always faster [19] and much slower than NN inference.

Hose et al. [8] recently introduced parameter-adaptive AMPC, described in detail in Sec. 3. It allows adapting the output of an AMPC to changes in MPC parameters (e.g., parameters of the system dynamics model or cost function) with a locally linear predictor based on approximated sensitivities of the MPC's optimization problem. Practical experiments indicate that this provides intuitive tuning nobs that generalize an AMPC to system instances with quite different parameters. Further, the NNs required in parameter-adaptive AMPC are small enough to be evaluated in milliseconds on common microcontrollers that cost only a few dollars, making this method particularly appealing for real-world applications at scale. However, the proposed method [8] relies on expert knowledge to tune the parameters correctly to achieve desired performance. This manual approach can be cumbersome for systems with many parameters or mass production. In this work, we showcase the efficiency of BO in automatically tuning parameter-adaptive AMPC for systems with many parameters through only a few hardware experiments. We demonstrate this on an eleven-dimensional tuning task for yaw control of a unicycle robot, for which manual tuning as in [8] is infeasible.

Bayesian Optimization for MPC Tuning. BO is a sample-efficient black-box optimization method that gained popularity for automatic controller tuning in recent years [20], for example, to tune the cost matrices in LQR control [21], optimize gaits [22], or the controller of a quadcopter [23]. In the context of classic MPC, BO has been used to optimize hyperparameters [24, 25, 26, 27]. BO can also be used to tune the prediction model to optimize closed-loop performance [28, 29]. Crucially, all aforementioned BO methods rely on solving MPC optimization problems online at control frequency. We overcome this problem by leveraging parameter-adaptive AMPC, which allows us to quickly obtain approximated optimal solutions through forward passes of the NNs and enables the usage of low-cost hardware.

Local BO methods such as GIBO [30, 31] and TuRBO [32] can cope with the increasing complexity of the optimization problem in higher dimensions. Such approaches have proven to be especially useful in finding good optima in a data-efficient manner by restricting exploration to a local region. As our approach focuses on fine-tuning the AMPC to the task at hand, we will resort to a local BO method, specifically TuRBO [32], to find an optimal configuration of parameters that are then used in the parameter-adaptive AMPC.

3 Fine-Tuning of Parameter-Adaptive AMPC with Bayesian Optimization

In this section, we first describe how to define a set of parameterized policies using parameter-adaptive AMPC and, second, how to solve the policy search problem data-efficiently with BO.

We consider general, nonlinear, discrete-time dynamical systems

$$s(k + 1) = f(s(k), a(k)), \quad s(0) = s_0, \tag{1}$$

where s are the states and a the actions. While we do not explicitly account for process and sensor noise in (1), the later experiments naturally include such uncertainties. The system (1) is controlled by an AMPC policy π_θ parameterized with θ, i.e., $a(k) = \pi_\theta(s(k))$. Here, θ are parameters of the MPC that is imitated and explained in detail in Sec. 3.1. The closed-loop system generates trajectories $\{(s(k), a(k))\}_{k=0}^T$ of length T that depend on the policy parameters θ. The novelty and goal of this paper is to automatically fine-tune the AMPC policy π_θ such that

$$\pi_{\theta^*} = \underset{\pi_\theta \in \Pi_{\mathrm{AMPC}}}{\arg\max} \; R(\theta), \tag{2}$$

based on trajectories of the closed-loop system from hardware experiments. We do not assume any properties of the reward R, for example, it can be sparse or non-Markovian.

We structure the rest of this section as follows: Sec. 3.1 describes the parameterized MPC problem, which π_θ imitates. Then, Sec. 3.2 defines the search space Π_{AMPC} in problem (2) as a parameter-adaptive AMPC [8] that keeps the parameterization in θ intact. Finally, in Sec. 3.3, we fine-tune θ using local BO to find the optimal parameters for the AMPC, such that π_{θ^*} is a solution to (2).

3.1 Parameterized Model-Predictive Control

We formulate the following nonlinear MPC problem [1] depending on parameters $\theta \in \Theta$

$$
\begin{aligned}
a_\theta^* = \arg\min_a \ & \sum_{\kappa=0}^{N} \ell_\theta(\kappa, s(\kappa|k), a(\kappa|k)) \\
\text{s.t.} \ \ s(0|k) = s(k), \quad & s(\kappa+1|k) = \tilde{f}_\theta(s(\kappa|k), a(\kappa|k)), \\
s(\kappa|k) \in \mathcal{X}_\theta(\kappa), \quad & a(\kappa|k) \in \mathcal{U}_\theta(\kappa) \quad \forall \kappa = 0 \dots N,
\end{aligned}
\tag{3}
$$

where ℓ_θ is a cost function, and $\mathcal{X}_\theta(\kappa)$ and $\mathcal{U}_\theta(\kappa)$ are the state and input constraints. The loss function, constraint sets, and MPC's dynamics model $\tilde{f}_\theta$ may depend on the parameter vector θ.

In classic MPC, (3) is solved repeatedly at every time k and the first element of the optimal predicted action sequence applied to the system. Thus, the optimization problem (3) implicitly defines a mapping from states to actions, which we call the policy $\pi_{\text{MPC}}(s(k), \theta) = a_\theta^*(0|k)$. The gradient of this policy with respect to the parameters at a specific state, $\frac{\partial}{\partial \theta} \pi_{\text{MPC}}(s(k), \theta)|_{\theta_{\text{nom}}}$, also known as sensitives, can be computed by commonly used NLP solvers along with a^* [33, 34, 24, 35].

3.2 Parameter-Adaptive AMPC

This section briefly summarizes the AMPC control strategy with sensitivities from [8]. It makes it possible to locally adjust a neural network approximation of (3) to parameters around nominal parameters θ_{nom}. To this end, parameter-adaptive AMPC combines two NNs to a single policy. First, a neural network is trained to imitate the nominal policy π_{MPC} using a large dataset $\mathcal{D}_{\text{nom}} = \{(s_j, \pi_{\text{MPC}}(s_j, \theta_{\text{nom}}))\}$. This yields the approximate nominal policy $\tilde{\pi}_{\text{MPC}}$. Second, when computing the dataset $\mathcal{D}_{\text{nom}}$ we also compute the sensitives which are collected in the dataset $\mathcal{D}_{\nabla\text{MPC}} = \{(s_j, \frac{\partial}{\partial \theta} \pi_{\text{MPC}}(s_j, \theta)|_{\theta_{\text{nom}}})\}$. We train a neural network to approximate the sensitives as $\tilde{\pi}_{\nabla\text{AMPC}}$. The approximate sensitivities can be used as linear predictor around θ_{nom} to adapt the optimal action given a change in the parameters θ. Thus, the parameter-adaptive AMPC policy is

$$
\pi_{\text{AMPC}}(s, \theta) = \tilde{\pi}_{\text{MPC}}(s) + \tilde{\pi}_{\nabla\text{MPC}}(s)(\theta - \theta_{\text{nom}}).
\tag{4}
$$

We define the policy search problem in (2) over the set of policies induced by parameter-adaptive AMPC and indexed by θ as $\Pi_{\text{AMPC}} = \{\pi_\theta : s \mapsto \pi_{\text{AMPC}}(s, \theta)\}$. In the next section, we use BO to automatically find the optimal parameters for a given reward function.

3.3 Task-Specific Fine-Tuning with Bayesian Optimization

We formalize the tuning problem (2) within the policy set Π_{AMPC} as a black-box optimization problem

$$
\theta^* = \arg\max_{\theta \in \Theta} R(\theta).
\tag{5}
$$

Thus, the search over policies in (2) reduces to finding optimal parameters θ^*. Importantly, the reward function R and the MPC cost function ℓ do not need to coincide. We can tune an existing AMPC policy for new systems as well as fine-tune to specific tasks. We use BO to solve the black-box optimization problem (5). This will allow us to formulate a high-level reward function that might not be practical for classic MPC and then *automatically* tune the sensitives to optimally solve the problem at hand. In BO, a probabilistic model of the reward function R, here a Gaussian process (GP), and an acquisition function determine the next parameters θ_i to evaluate at iteration i. At each iteration, we conduct closed-loop experiments and collect noisy evaluations of the reward as

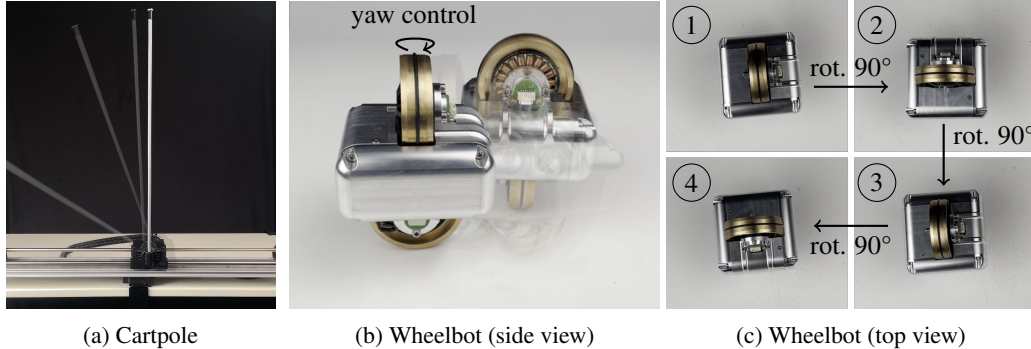| (a) Cartpole | (b) Wheelbot (side view) | (c) Wheelbot (top view) |

Figure 2: Hardware systems used for evaluation: cartpole and reaction wheel unicycle robot. *The cartpole is a classic control benchmark system on which we perform swing-up and stabilization. The Wheelbot is a reaction wheel unicycle robot on which we demonstrate balancing and yaw control. A video of our experiments is available under https://youtu.be/R96QpD5iusI.*

$R_i = R(\theta_i) + \epsilon_i$ with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ to sequentially build a data set $\mathcal{D}_{\text{tune}} = \{(\theta_i, R_i)\}_{i=1}^K$ that is informative about the optimal policy. Here, K is the number of experiments. The noise with variance σ^2 might come from different initial states $s_i(0)$ or disturbances during the experiments. For details on BO we refer to Garnett [9], in the context of controller tuning to Paulson et al. [20].

We choose TuRBO [32] as our BO method to locally fine-tune the initial solution of the given task. We fully utilize the local idea of TuRBO by only considering one trust region that shrinks over time and collapses to the locally optimal solution. Note that through this local approach we do not rely on explicit parameter bounds Θ as the bound of the local trust region is inferred based on the length scales of the covariance function of the GP (cf. [32, Sec. 2]).

4 Implementation on Benchmark Systems

We implement and evaluate parameter-adaptive AMPC and tuning with BO on two benchmark systems: a cartpole and a reaction wheel unicycle robot (Fig. 2). We chose both systems because they exhibit strong nonlinear dynamics, are unstable, and require fast feedback control, making them predestined for AMPC. Further, we implement the parameter-adaptive AMPC (i.e., neural network controller inference) on the onboard embedded CPUs[1,2]. The CPUs are not powerful enough to solve the implicit MPC optimization problem in real-time. While the NN controllers run on embedded CPUs, we conveniently use the TuRBO BoTorch implementation [36] on a laptop. Our full implementation will be publicly available upon acceptance.

4.1 Cartpole Swing-Up and Stabilization

The cartpole system is a standard benchmark in control [37]. We use a single policy to control the swing-up and stabilization of the pole in the upwards-facing position without violating the constraints on the rail. The AMPC implementation closely resembles the publicly available code used in [8]. Therefore, we only elaborate on details relevant to the fine-tuning with BO.

The cartpole's state consists of cart position, pendulum angle, and their derivatives, thus $s_{\text{cartpole}} \in \mathbb{R}^4$ with voltage applied to the cart's motor as action $a_{\text{cartpole}} \in \mathbb{R}$. The dynamics function used in the MPC optimization problem is parametrized by $\theta_{\text{cartpole}} = [m_{\text{add}}, M, C_1, C_2, C_3] \in \mathbb{R}^5$, where m_{add} is the mass atop the rod, M is the mass of the cart, and C are friction and motor constants.

Reward for Fine-Tuning. The task of the pendulum is to perform a swing-up as fast as possible and stabilize afterwards around the upright position with the cart at the center of the rail

---

[1]The cartpole has a STM32G474 ARM Cortex-M4 CPU running at 170 MHz

[2]The Wheelbot has a Raspberry Pi CM4 with BCM2711 quad-core Cortex-A72 CPU running at 1.5 GHz

for a total of $20\,\mathrm{s}$. We formalize this in the following sparse reward function: $R_{\mathrm{cartpole}}(\theta) = \frac{1}{T}t_{\mathrm{up}} - \frac{w_{\mathrm{pos}}}{T-t_{\mathrm{up}}}\sum_{k=t_{\mathrm{up}}}^{T}s_{\mathrm{pos}}(k)^2$, where t_{up} is the number of time steps that the pendulum has successfully remained in the upright position, i.e., the angle remains within $[-15°, 15°]$. If no swing-up is achieved or constraints are violated ($|s_{\mathrm{pos}}| > \bar{s}_{\mathrm{pos}}$), the reward is set to 0. All weighting factors are in Appendix A. Implementing such an objective would be difficult in classic MPC.

4.2 Balancing and Yaw Control on a Reaction Wheel Unicycle Robot

The Wheelbot is a symmetric, balancing, reaction wheel unicycle robot [38] with two wheels: one driving wheel and an orthogonal reaction wheel. The robot can directly control its pitch and roll angles by applying torques to its wheels. However, the robot does not have a third "turntable" actuator to control its yaw directly. Classic linear control methods fail at controlling the yaw angle for this class of robots [39, 40, 41]. However, a nonlinear MPC, as in this paper, can use the reaction wheel's nonlinear gyroscopic effects to steer the robot's orientation.

MPC Implementation. The robot's state can be described by minimal coordinates consisting of roll, pitch, and yaw orientation, and both wheel encoder values, and all of their derivatives [38], therefore $s_{\mathrm{wheelbot}} \in \mathbb{R}^{10}$. The actions are the torques applied by the motors to both wheels, $a_{\mathrm{wheelbot}} \in \mathbb{R}^2$. The robot's continuous-time, nonlinear dynamics in implicit form are described in detail in [42]. The dynamics are parametrized by $\theta_{\mathrm{wheelbot}} = [m_{\mathrm{B}}, m_{\mathrm{W,R}}, I_{\mathrm{B}}, I_{\mathrm{W,R}}, r_{\mathrm{W,R}}, l_{\mathrm{WB}}, \mu]$, where m denotes masses, $I \in \mathbb{R}^3$ diagonals of mass moment of inertia matrices, r the effective wheel radius, l the distance between the wheels' rotation axis, and $\mu \in \mathbb{R}^2$ friction parameters of the wheel-to-ground contact, and indices W and R the driving and reaction wheels, and B the robots main body. Due to symmetries, the number of free parameters is $\theta_{\mathrm{wheelbot}} \in \mathbb{R}^{11}$. We implement a nonlinear MPC with quadratic cost, a horizon lookahead of $1.2\,\mathrm{s}$ discretized with $20\,\mathrm{ms}$ steps using the implicit integrators from [43], and appropriate action, state, and terminal constraints. The MPC optimization problem is formulated in CasADi [44] with sensitivities by [45], and solved with IPOPT [46].

Neural Network Approximation. The dataset that we synthesize contains 3.5 million random samples of states and optimal actions. Computation of the dataset takes 86 thousand core hours[3]. We use fully connected feedforward NNs with 100 neurons per layer, a mixture of tangent hyperbolic and rectified linear activations, and 4 layers and 8 layers for approximating inputs and sensitivities. We implement the NN inference in C++ with Eigen on the Wheelbot's onboard CPU[2]. Inference on both NNs takes less than $300\,\mathrm{\mu s}$; thus, we can evaluate the AMPC at a control frequency of $200\,\mathrm{Hz}$.

Reward for Fine-Tuning. The objective of the Wheelbot is to control its yaw to a sequence of 4 setpoints with $90°$ step responses while balancing in place. Every episode takes $22\,\mathrm{s}$. We chose a sparse reward for fine-tuning that only considers the error in yaw, roll, and pitch angles s_{yrp} and driving wheel angle s_{wheel} as $R_{\mathrm{wheelbot}}(\theta) = -\frac{1}{T}\sum_{k=0}^{T} w_{\mathrm{yrp}}^{\top}(s_{\mathrm{yrp}}(k) - s_{\mathrm{yrp,ref}}(k))^2 + w_{\mathrm{wheel}}s_{\mathrm{wheel}}(k)^2$, where w are appropriate weights, to let the robot reorient in place effectively. A failed experiment in which the robot crashes yields a reward of -1. This is two times smaller than the worst reward achieved without a crash.

5 Experimental Results

In this section, we present the results from simulation and hardware experiments on automatically tuning the parameter-adaptive AMPC with BO for the two systems presented in Sec. 4. The simulations' primary goal is to evaluate our method's generalizability. In the hardware experiments, we aim to demonstrate that our method is capable of running in real-world conditions, on low-cost hardware, and can learn efficiently within a feasible amount of time. Our software implementation will be publicly available upon acceptance.

---

[3]computed in parallel on Intel Xeon 8468 Sapphire Rapids CPUs at $3.8\,\mathrm{GHz}$

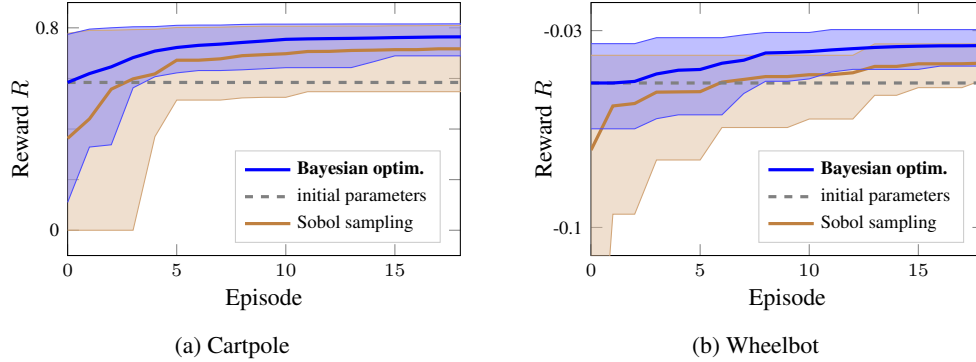| | (a) Cartpole | (b) Wheelbot |

Figure 3: Average, minimum, and maximum reward improvement on 100 random, simulated systems stabilized by the same parameter-adaptive AMPC (no retraining of neural networks). *Using Bayesian optimization (blue) reliably improves performance with a sparse closed-loop objective given a rough initial guess. We include a pseudo-random baseline for comparison (brown).*

5.1 Results in Simulation Experiments

For the simulations, we generate 100 system instances with randomly sampled parameters for cartpole and Wheelbot. We then perform local fine-tuning starting from the nominal parameter values as initial conditions. We use the same neural networks for all system instances, thus no retraining takes place. In all experiments, we add quasi-random Sobol sampling within reasonable bounds around the nominal parameters as the baseline. Sobol sampling is more sophisticated than grid search or random sampling and represents an engineering approach for finding good-performing parameters, providing a benchmark against which we can compare the sample efficiency of our method.

Fig. 3 shows the results of our simulations for cartpole and Wheelbot. In both examples, the same neural network controller is able to stabilize a broad range of random systems, i.e., the minimum and average reward indicates that the parameter-adaptive AMPC generalizes across instances of the same system with different parameters. BO consistently improves over the initial parameters. The Sobol sampling baseline also finds at least stabilizing parameters but requires more trials.

5.2 Results in Hardware Experiments

We deploy our method on the hardware systems shown in Fig. 2 to demonstrate that the proposed method can tune AMPC controllers without retraining, thus compensating for inevitable mismatch between nominal model (used for the MPC) and real hardware. The actual system parameters for both systems differ from the nominal ones used in the MPC, which necessitates parameter-adaptive AMPC. In our experiments, we initialize BO with parameters that successfully complete the task, i.e., the cartpole can perform a swing-up and the Wheelbot stabilizes and controls the yaw orientation. In practical applications, the engineer would typically have good intuition about nominal parameters through direct measurements or average values from other system instances. If such a good initial guess is not available, a set of random trials could also be used to initialize learning.

The hardware results are summarized in Fig. 4. Compared to the initial guess, BO improves within 20 experiments (10-15 min of hardware interaction), which is consistent across multiple random seeds. Qualitatively, for the pendulum, optimized parameters reduce the time required for the swing-up and drive the cart to the center of the rail during stabilization instead of oscillating around the center as depicted in Fig 4 (b, top). On the reaction wheel unicycle robot, we can observe in Fig 4 (b, bottom) that the optimized policy reduces oscillations of the driving wheel (i.e., less driving back and forth during maneuvers) and minimizes the overshoots during the yaw step response. This improvement in qualitative performance for both systems is also clearly visible in the video of our experiments under https://youtu.be/R96QpD5iusI.
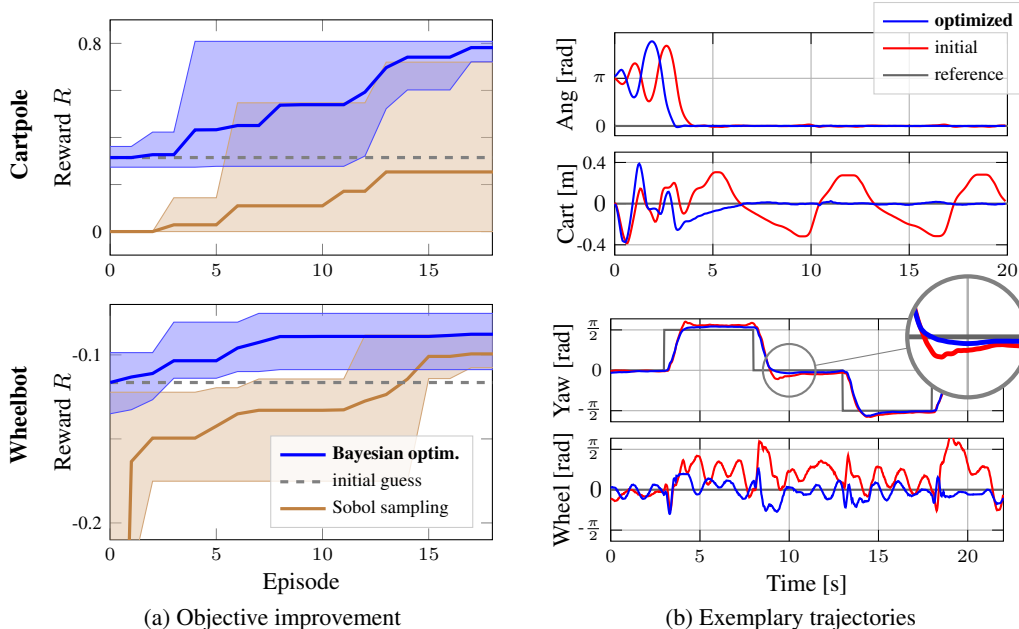
	(a) Objective improvement	(b) Exemplary trajectories

Figure 4: Results in hardware experiments. *Bayesian optimization (blue) tunes the AMPC to sat-isfactory performance in* 20 *hardware experiments ("Episodes"). For comparison, we include a pseudo-random baseline (brown). Improvement from initial (red) to the optimized policy (blue) is illustrated with closed-loop trajectories: on the cartpole, shorter swing-up and zero cart position (top right); on the Wheelbot, reduced yaw overshoot and driving wheel action (bottom right).*

6   Conclusion

In this paper, we proposed a method for automatically fine-tuning an AMPC. This substantially reduces the need for iteratively synthesizing datasets and retraining NN controllers, which is a ma-jor drawback of classic AMPC in practice. With our method, a single NN controller imitating a nominal MPC is sufficient, while the proposed automatic tuning adapts the AMPC to the ac-tual hardware. We achieve this by fine-tuning parameter-adaptive AMPCs [8] to optimal task- and hardware-specific performance with local BO from only a few hardware experiments. We demon-strate the effectiveness of our method on two challenging, nonlinear, and unstable control tasks in simulation and hardware experiments: a cartpole swing-up and balancing task, and a reaction wheel unicycle robot balancing and yaw control task. In both setups, the neural network controller runs on embedded processors and is evaluated within milliseconds, which would not be possible with the classic optimization-based MPC we imitate. BO consistently improves the initial parameters within 20 hardware interactions on both tasks.

Limitations. We see three main limitations of the proposed method. First, the used AMPC scheme can only adapt to parameter changes within a local region around the nominal parameters as it relies on the sensitivities of the MPC problem. However, we empirically show, that they are sufficient to locally adapt policies and – to some extent – transfer to different instances from the same class of systems. However, the sensitivities may not be accurate enough for vastly different systems or control objectives to achieve satisfactory performance. Second, we only evaluated our method on a small and medium-sized system, which is good empirical indication that the method scales well. However, it is unclear, how to scale the AMPC synthesis to very high dimensional states (i.e., en-vironments with hundreds of states or end-to-end learning from image data). Lastly, the proposed method considers time-invariant parameters. This might be an oversimplification in applications where parameters change over time, for example, due to wear and tear. In future work, we will tackle this last issue by investigating the usage of time-varying or event-triggered BO schemes [47, 48].

References

[1] J. B. Rawlings, D. Q. Mayne, and M. Diehl. *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017.

[2] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot. Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4029–4036. IEEE, 2014.

[3] 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.

[4] 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. IEEE, 2016.

[5] G. Bledt and S. Kim. Extracting legged locomotion heuristics with regularized predictive control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 406–412. IEEE, 2020.

[6] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82): eadg1462, 2023.

[7] C. Gonzalez, H. Asadi, L. Kooijman, and C. P. Lim. Neural networks for fast optimisation in model predictive control: A review. *arXiv preprint arXiv:2309.02668*, 2023.

[8] H. Hose, A. Gräfe, and S. Trimpe. Parameter-adaptive approximate MPC: Tuning neural-network controllers without retraining. In *6th Conference on Learning for Dynamics and Control (accepted)*. PMLR, 2024.

[9] R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

[10] F. A. Bayer, F. D. Brunner, M. Lazar, M. Wijnand, and F. Allgöwer. A tube-based approach to nonlinear explicit MPC. In *55th Conference on Decision and Control (CDC)*. IEEE, 2016.

[11] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Proceedings of the 5th Conference on Robot Learning*, 2022.

[12] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 2002.

[13] A. Alessio and A. Bemporad. A survey on explicit model predictive control. *Nonlinear Model Predictive Control: towards New Challenging Applications*, 2009.

[14] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe. Safe and fast tracking on a robot manipulator: robust MPC and neural network control. *IEEE Robotics and Automation Letters*, 2020.

[15] J. Carius, F. Farshidian, and M. Hutter. MPC-Net: A first principles guided policy search. *IEEE Robotics and Automation Letters*, 2020.

[16] S. Adhau, S. Patil, D. Ingole, and D. Sonawane. Embedded implementation of deep learning-based linear model predictive control. In *Sixth Indian control conference (ICC)*, 2019.

[17] M. Klaučo, M. Kalúz, and M. Kvasnica. Machine learning-based warm starting of active set methods in embedded model predictive control. *Engineering Applications of Artificial Intelligence*, 2019.

[18] S. W. Chen, T. Wang, N. Atanasov, V. Kumar, and M. Morari. Large scale model predictive control with neural networks and primal active sets. *Automatica*, 2022.

[19] Y. Vaupel, N. C. Hamacher, A. Caspari, A. Mhamdi, I. G. Kevrekidis, and A. Mitsos. Accelerating nonlinear model predictive control through machine learning. *Journal of process control*, 2020.

[20] J. A. Paulson, F. Sorourifar, and A. Mesbah. A tutorial on derivative-free policy learning methods for interpretable controller representations. In *2023 American Control Conference (ACC)*, pages 1295–1306, 2023.

[21] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe. Automatic LQR tuning based on Gaussian process global optimization. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 270–277. IEEE, 2016.

[22] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth. Bayesian optimization for learning gaits under uncertainty: An experimental comparison on a dynamic bipedal walker. *Annals of Mathematics and Artificial Intelligence*, 76:5–23, 2016.

[23] F. Berkenkamp, A. P. Schoellig, and A. Krause. Safe controller optimization for quadrotors with gaussian processes. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 491–496. IEEE, 2016.

[24] O. Andersson, M. Wzorek, P. Rudol, and P. Doherty. Model-predictive control with stochastic collision avoidance using Bayesian policy optimization. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4597–4604. IEEE, 2016.

[25] R. Guzman, R. Oliveira, and F. Ramos. Heteroscedastic Bayesian optimisation for stochastic model predictive control. *IEEE Robotics and Automation Letters*, 6(1):56–63, 2020.

[26] A. Gharib, D. Stenger, R. Ritschel, and R. Voßwinkel. Multi-objective optimization of a path-following MPC for vehicle guidance: A Bayesian optimization approach. In *2021 European Control Conference (ECC)*, pages 2197–2204. IEEE, 2021.

[27] R. Guzman, R. Oliveira, and F. Ramos. Bayesian optimisation for robust model predictive control under model parameter uncertainty. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5539–5545. IEEE, 2022.

[28] D. Piga, M. Forgione, S. Formentin, and A. Bemporad. Performance-oriented model learning for data-driven MPC design. *IEEE control systems letters*, 3(3):577–582, 2019.

[29] F. Sorourifar, G. Makrygirgos, A. Mesbah, and J. A. Paulson. A data-driven automatic tuning method for MPC under uncertainty using constrained Bayesian optimization. *IFAC-PapersOnLine*, 54(3):243–250, 2021.

[30] S. Müller, A. von Rohr, and S. Trimpe. Local policy search with Bayesian optimization. *Advances in Neural Information Processing Systems*, 34:20708–20720, 2021.

[31] K. Wu, K. Kim, R. Garnett, and J. Gardner. The behavior and convergence of local Bayesian optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

[32] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek. Scalable global optimization via local Bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

[33] A. V. Fiacco. Sensitivity analysis for nonlinear programming using penalty methods. *Mathematical programming*, 10(1):287–311, 1976.

[34] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, J. D. Siirola, et al. *Pyomo-optimization modeling in python*, volume 67. Springer, 2017.

[35] H. Pirnay, R. López-Negrete, and L. T. Biegler. Optimal sensitivity based on ipopt. *Mathematical Programming Computation*, 4:307–331, 2012.

[36] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

[37] O. Boubaker. The inverted pendulum benchmark in nonlinear control theory: a survey. *International Journal of Advanced Robotic Systems*, 10(5):233, 2013.

[38] A. R. Geist, J. Fiene, N. Tashiro, Z. Jia, and S. Trimpe. The Wheelbot: A jumping reaction wheel unicycle. *IEEE Robotics and Automation Letters*, 7(4):9683–9690, 2022.

[39] J. Lee, S. Han, and J. Lee. Decoupled dynamic control for pitch and roll axes of the unicycle robot. *IEEE Transactions on Industrial Electronics*, 60(9):3814–3822, 2012.

[40] G. P. Neves and B. A. Angélico. A discrete lqr applied to a self-balancing reaction wheel unicycle: Modeling, construction and control. In *2021 American control conference (ACC)*, pages 777–782. IEEE, 2021.

[41] M. A. Rosyidi, E. H. Binugroho, S. E. R. Charel, R. S. Dewanto, and D. Pramadihanto. Speed and balancing control for unicycle robot. In *2016 International Electronics Symposium (IES)*, pages 19–24. IEEE, 2016.

[42] Y. Daud, A. Al Mamun, and J.-X. Xu. Dynamic modeling and characteristics analysis of lateral-pendulum unicycle robot. *Robotica*, 35(3):537–568, 2017.

[43] J. Frey, J. De Schutter, and M. Diehl. Fast integrators with sensitivity propagation for use in casadi. In *2023 European Control Conference (ECC)*, pages 1–6. IEEE, 2023.

[44] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11:1–36, 2019.

[45] J. A. Andersson and J. B. Rawlings. Sensitivity analysis for nonlinear programming in casadi. *IFAC-PapersOnLine*, 51(20):331–336, 2018.

[46] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.

[47] I. Bogunovic, J. Scarlett, and V. Cevher. Time-varying Gaussian process bandit optimization. In *Artificial Intelligence and Statistics*, pages 314–323. PMLR, 2016.

[48] P. Brunzema, A. von Rohr, F. Solowjow, and S. Trimpe. Event-triggered time-varying Bayesian optimization. *arXiv preprint arXiv:2208.10790*, 2022.

Appendix for "Fine-Tuning of Neural Network Approximate MPC without Retraining via Bayesian Optimization"

Following is the appendix for the paper of *Fine-Tuning of Neural Network Approximate MPC without Retraining via Bayesian Optimization*. This includes:

Sec. A: An overview of hyperparameters used for Bayesian optimization in hardware and software experiments

Sec. B: The parameter bounds from which the random systems where synthesized

A Bayesian Optimization Hyperparameters

In the following, we list all the hyperparameters for the reward functions of the cartpole and Wheelbot, as well as the hyperparameters for TuRBO [32], to reproduce the results in Sec. 5. We will only list the hyperparameters from TuRBO that differ from the default values used in the corresponding paper as well as in the TuRBO implementation in BoTorch [36].

Table 1: Hyperparameters of the cartpole and Wheelbot experiments.

Cartpole	Hyperparameter	$\bar{s}_\mathrm{pos}$	w_pos	TuRBO L_initial
	Value	$0.39\,\mathrm{m}$	$\frac{5}{0.39}\,\mathrm{m}^{-2}$	0.4
Wheelbot	**Hyperparameter**	w_yrp	w_wheel	TuRBO τ_fail
	Value	$[1, 0.001, 0.01]^\top$	0.1	3

Number of Simulation Experiments. Fig. 3 results are for cartpole and Wheelbot simulations with 100 random systems each. The parameter bounds for this are listed in Appendix B.

Number of Hardware Experiments. Fig. 4 results on cartpole and Wheelbot are for 5 random seeds for TuRBO and 5 random seeds for Sobol sampling each.

B Parameter Bounds

Below are the parameter bounds used to synthesize random systems around the nominal parameters for the simulation results in Sec. 5 for both cartpole and the Wheelbot.

Table 2: Parameter bounds around nominal parameters θ_nom for cartpole.

Parameter	m_add	M	C_1	C_2	C_3
Upper bound	$0.016\,\mathrm{kg}$	$0.4\,\mathrm{kg}$	$2\,\mathrm{N\,s\,m^{-1}}$	$0.4\,\mathrm{N\,V^{-1}}$	$0.008\,\mathrm{N\,m\,s\,rad^{-1}}$
Lower bound	$-0.016\,\mathrm{kg}$	$-0.4\,\mathrm{kg}$	$-2\,\mathrm{N\,s\,m^{-1}}$	$-0.4\,\mathrm{N\,V^{-1}}$	$-0.008\,\mathrm{N\,m\,s\,rad^{-1}}$

Table 3: Parameter bounds around nominal parameters θ_nom for Wheelbot.

Parameter	m_B	$m_\mathrm{W,R}$	$I_{\mathrm{B},\{x,y,z\}}$	$I_{\mathrm{W,R},\{y,z\}}$
Upper bound	$0.1\,\mathrm{kg}$	$0.05\,\mathrm{kg}$	$100 \cdot 10^{-6}\,\mathrm{kg\,m^2}$	$20 \cdot 10^{-6}\,\mathrm{kg\,m^2}$
Lower bound	$-0.1\,\mathrm{kg}$	$-0.05\,\mathrm{kg}$	$-100 \cdot 10^{-6}\,\mathrm{kg\,m^2}$	$-20 \cdot 10^{-6}\,\mathrm{kg\,m^2}$

Table 4: Parameter bounds around nominal parameters θ_nom for Wheelbot (continued) .

Parameter	$I_{\mathrm{W,R}\{x\}}$	$r_\mathrm{W,R}$	l_WB	μ_1	μ_2
Upper bound	$50 \cdot 10^{-6}\,\mathrm{kg\,m^2}$	$0.005\,\mathrm{m}$	$0.005\,\mathrm{m}$	0.01	50
Lower bound	$-50 \cdot 10^{-6}\,\mathrm{kg\,m^2}$	$-0.005\,\mathrm{m}$	$-0.005\,\mathrm{m}$	-0.01	-50