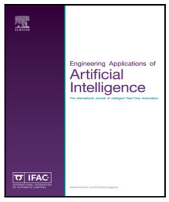
Engineering Applications of Artificial Intelligence 123 (2023) 106211

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Optimization of the model predictive control meta-parameters through reinforcement learning

Eivind Bøhn [a,*], Sebastien Gros [b], Signe Moe [c], Tor Arne Johansen [b,d]

[a] Artificial Intelligence and Data Analytics, SINTEF Digital, Oslo, Norway
[b] Department of Engineering Cybernetics, NTNU, Trondheim, Norway
[c] Sopra Steria Applications, Oslo, Norway
[d] Centre for Autonomous Marine Operations and Systems, NTNU, Trondheim, Norway

ARTICLE INFO

Keywords:
Reinforcement learning
Model predictive control
Event-triggered control
Linear quadratic regulator

ABSTRACT

Model predictive control (MPC) is increasingly being considered for control of fast systems and embedded applications. However, MPC has some significant challenges for such systems, such as its high computational complexity. Further, the MPC parameters must be tuned, which is largely a trial-and-error process that affects the control performance, the robustness, and the computational complexity of the controller to a high degree. This paper presents a multivariate optimization method based on reinforcement learning (RL) that automatically tunes the control algorithm's parameters from data to achieve optimal closed-loop performance. The main contribution of our method is the inclusion of state-dependent optimization of the meta-parameters of MPC, i.e. parameters that are non-differentiable wrt. the MPC solution. Our control algorithm is based on an event-triggered MPC, where we learn when the MPC should be re-computed, and a dual-mode MPC and linear state feedback control law applied in between MPC computations. We formulate a novel mixture-distribution RL policy determining the meta-parameters of our control algorithm and show that with joint optimization we achieve improvements that do not present themselves with univariate optimization of the same parameters. We demonstrate our framework on the inverted pendulum control task, reducing the total computation time of the control system by 36% while also improving the control performance by 18.4%.

1. Introduction

Model predictive control (MPC) is a powerful optimizing control technique, capable of controlling a wide range of systems with high control proficiency while respecting system constraints. Nonlinear MPC can even handle nonlinear dynamics and constraints, and while not as simple as its linear counterpart, is increasingly being considered for embedded systems applications with fast dynamics (Gros et al., 2012; Albin et al., 2015; Johansen, 2017). However, one of the main drawbacks of the MPC is its high computational complexity, which makes it ill-suited for applications with low-powered hardware or battery energy restrictions, necessitating some form of compromise in its implementation (Feng et al., 2012; Gondhalekar et al., 2015).

The high computational complexity of the MPC comes as a result of its online operation consisting of solving a numerical optimal control problem (OCP) at every time step, executing the first control input of the solution, and then solving the OCP again at the subsequent time step. Several adjustments to this basic paradigm have been suggested in order to reduce the computational complexity, e.g. early termination (suboptimal) MPC (Scokaert et al., 1999), semi-explicit MPC (Goebel and Allgöwer, 2015), and explicit MPC (Bemporad et al., 2002).

A further challenge of the MPC method is the need to tune its parameters to the task at hand, greatly affecting the control proficiency, robustness, and computational complexity of the controller. The main tunable parameter in these regards is the prediction horizon, which essentially controls how far into the future the MPC evaluates the optimality of its solution. Further, the performance sensitivity to the prediction horizon varies over the state space, and this observation motivated the adaptive-horizon MPC technique, in which the prediction horizon is varied according to some criteria. There exist several suggestions in the literature on how to design these criteria, e.g. terminal conditions (Michalska and Mayne, 1993; Krener, 2018), as decision variables of the OCP (Scokaert and Mayne, 1998), and learning approaches (Gardezi and Hasan, 2018; Bøhn et al., 2021b). Other parameters of the MPC scheme are also subject to tuning, e.g. discretization step size, objective functions, optimality tolerances, and constraints. How to tune all these parameters for the MPC scheme is a non-trivial question.

* Corresponding author.
E-mail addresses: eivind.bohn@sintef.no (E. Bøhn), sebastien.gros@ntnu.no (S. Gros), signe.moe@soprasteria.no (S. Moe), tor.arne.johansen@ntnu.no (T.A. Johansen).

https://doi.org/10.1016/j.engappai.2023.106211
Received 1 July 2022; Received in revised form 14 March 2023; Accepted 22 March 2023
Available online 30 March 2023

Learning can be an important tool in assisting the tuning process of the MPC scheme. In this paper, we propose the novel idea of tuning the meta-parameters of the MPC scheme using reinforcement learning (RL). By meta-parameters, we mean parameters that are non-differentiable wrt. the MPC outputs such as the prediction horizon and deciding when the MPC should be computed. This separates our work from tuning of other non-meta-parameters such as cost functionals and the dynamics model, the tuning of which has previously been demonstrated in the literature (Amos et al., 2018; Gros and Zanon, 2019). By utilizing RL we get the benefits of learning from self-supervised closed-loop data, comparative data efficiency (since RL uses gradient information), and state-dependent optimization of the chosen parameters. It might seem counter-productive to introduce a learning framework such as RL to the MPC scheme from a computational complexity standpoint. We suggest in this paper to use small neural networks (with a few thousand parameters) to implement the RL components, and evaluating these in a forward pass is computationally negligible compared to solving a nonlinear MPC problem. Thus, the potential savings from tuning the MPC scheme far outweigh the overhead from the tuning framework. Further, there is no requirement that the parameter updates from RL must be computed in real-time or even on the control system itself, and could therefore be offloaded to a remote computer and communicated back to the control system whenever convenient.

This work extends our previous works (Bøhn et al., 2021a), in which we propose to learn when it is necessary to recompute the MPC solution, and Bøhn et al. (2021b), in which we suggest learning the optimal prediction horizon of the MPC scheme as a function of the state. Here, we propose a unified framework in which these meta-parameters, as well as any other parameters (e.g. cost functionals) of the control algorithm are jointly optimized according to a user-configurable objective. In this paper, we set the objective to simultaneously maximize the control performance and reduce some measure of the computational complexity, i.e. resources that are spent as a product of control algorithm calculation such as network transmission in networked control systems, CPU load, or energy consumption.

The control algorithm we propose consists of a recomputation policy that decides when the event-triggered MPC solution should be computed, a state-feedback controller (i.e. the linear quadratic regulator (LQR)) that is applied on the predicted state trajectory produced by the MPC in between MPC computations, and an RL algorithm that incorporates the parameters of these controllers and optimizes them according to the specified objective. Although we can enforce input constraints, there is no mechanism to ensure that the state constraints hold in our control algorithm, and the constraint satisfaction properties will be determined by the behavior identified as optimal through RL. While we do not treat the topics of closed-loop stability and feasibility guarantees in this paper, several works exist in the literature on how such guarantees can be given when combining RL and MPC (Aswani et al., 2013; Fisac et al., 2018; Gros et al., 2020). We demonstrated in our aforementioned works the effectiveness of learning these meta-parameters in isolation, however, it is clear that the questions of *when* and *how* (wrt. its tunable parameters) to compute the control algorithm are related, and treating them separately fails to consider the interactions and indirect effects at play.

To the best of our knowledge, no other works optimize the described meta-parameters (i.e. recomputation and prediction horizon) together in a state-dependent manner. In Gardezi and Hasan (2018) the authors use supervised learning to learn a state-dependent predictor for the optimal prediction horizon. Thus, they are required to in advance construct a rich dataset of MPC solutions using numerous choices of prediction horizons covering the whole state space, as opposed to automatically and iteratively converging to the optimal horizon as is possible with our RL approach. Other works propose learning a single global (i.e. state-independent) value for MPC parameters including the prediction horizon using derivative-free optimization (Bansal et al., 2017; Piga et al., 2019; Edwards et al., 2021; Makrygiorgos et al., 2022). Of these, the most similar method to our own is the AutoMPC framework presented in Edwards et al. (2021), a method where several parameters of a path integral MPC such as cost functionals, prediction horizon, and dynamics model can be learned in a configurable manner using Bayesian optimization. Their framework is however not state-dependent or event-triggered as ours is, and therefore does not support optimizing the recomputation parameter. See also Mesbah et al. (2022) for an overview of Bayesian optimization for MPC and its connection to our RL-based approach. For event-triggered MPC, several methods have been proposed for how to design the triggering policy (i.e. the recomputation parameter) based on monitoring the tracking error and designing hand-crafted rules derived from domain knowledge about the system and the controller (Berglind et al., 2012; Li and Shi, 2014; Yoo and Johansson, 2019). In Yoo and Johansson (2019) the authors introduce an element of learning through a model predicting the unknown system noise, which is used to adaptively set the triggering threshold. However, compared to our framework these methods are fundamentally manually designed and do not consider the interactions of the recomputation parameter with the rest of the control algorithm's parameters in a multivariate optimization procedure.

The contributions of this paper can be summarized as:

1. We propose the novel idea of optimizing the meta-parameters of the MPC scheme using RL, i.e. parameters that are non-differentiable wrt. the MPC solution.
2. We develop a novel MPC formulation whose parameters are optimized through RL according to a configurable objective. The benefits of RL for MPC tuning include multivariate optimization, state-dependency for tuned parameters, and automatic design of data-collection.
3. By configuring the RL tuning objective to maximize control performance and minimize a measure of computational complexity we demonstrate considerable reduction in total computation time of the control algorithm while improving control performance.

The rest of the paper is organized as follows. Section 2 introduces the problem formulation and the necessary theoretical background for our method. Section 3 then describes our method in detail, before we in Section 4 demonstrate our method on a numerical example with the inverted pendulum.

2. Preliminaries and problem formulation

We consider control problems on the form:

$$x_{t+1} = f(x_t, u_t), \ \min_{x,u} \sum_{t=0}^{T} C(x_t, u_t) \tag{1}$$

where $x \in \mathbb{R}^{n_x}$ is the state vector, $u \in \mathbb{R}^{n_u}$ is the control input vector, the function f defines the (discrete-time) system dynamics and C defines the cost objective to be minimized. The system runs in an episodic fashion, beginning in some initial state x_0 and terminating after some predetermined time T has passed. The control inputs are produced by a parameterized control algorithm (or equivalently, a policy) π_θ, for which we want to optimize the parameters θ in the sense of minimizing the objective in (1) using RL. The defining characteristic of this work is that some of these parameters are non-differentiable wrt. the control algorithm's outputs, which we will address further in Section 3.

For any statement that holds regardless of the time, we omit the time subscript. To denote a contiguous sequence of points we use the subscript $x_{t:t+n}$, i.e. the sequence of states from time t to time $t+n$. We denote the time dimension of variables internal to any controller scheme with a subscript k. Finally, matrices are denoted with bold uppercase letters, e.g. $\mathbf{A}$.

2.1. Control algorithm

2.1.1. Model predictive control

In this paper, we consider adaptive-horizon nonlinear state-feedback discrete-time MPC (Allgöwer et al., 1999; Rawlings et al., 2017). The MPC receives as arguments the current state of the plant, $\bar{x}_t$, exogenous input variables (e.g. reference signals, forecasts), $\hat{p}_t$, as well as the prediction horizon, N_t, for the OCP. We label the MPC control law (for a given horizon selected by the horizon policy $\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}$, more on this in Section 3.2.3) as:

$$\left(u^{\mathrm{M}}_{t:t+N_t-1},\ \hat{x}_{t:t+N_t}\right) = \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(\bar{x}_t, \hat{p}_t, N_t) \tag{2}$$

where the first return value is the optimal input sequence, the second return value is the predicted optimal state trajectory, and θ^{M} are the tunable parameters of the MPC scheme. The OCP is formulated as:

$$\min_{x,u} \quad \sum_{k=0}^{N_t-1} \rho^k \ell_{\theta^{\mathrm{M}}}(x_k, u_k, \hat{p}_k) + \rho^{N_t} m_{\theta^{\mathrm{M}}}(x_{N_t}), \tag{3}$$

$$\text{s.t.} \quad x_0 = \bar{x}_t \tag{4}$$

$$x_{k+1} = \hat{f}_{\theta^{\mathrm{M}}}(x_k, u_k, \hat{p}_k) \quad \forall\, k \in 0, \dots, N_t - 1 \tag{5}$$

$$h_{\theta^{\mathrm{M}}}(x_k, u_k) \leq 0 \quad \forall\, k \in 0, \dots, N_t - 1 \tag{6}$$

Here, $\ell_{\theta^{\mathrm{M}}}$ is the stage cost, $m_{\theta^{\mathrm{M}}}$ is the terminal cost, $\rho \in (0, 1]$ is the discount factor, $\hat{f}_{\theta^{\mathrm{M}}}$ is the MPC dynamics model, and $h_{\theta^{\mathrm{M}}}$ is the constraint vector. The stage cost evaluates the computed solution locally up to $N_t - 1$ steps, the terminal cost $m_{\theta^{\mathrm{M}}}(x_{N_t})$ should therefore provide global information about the desirability of the terminal state of the computed trajectory, which helps the MPC avoid sub-optimal performance. Therefore, the more accurate the terminal cost is wrt. the total cost of the infinite horizon solution, the shorter horizons can be used in the MPC scheme while still delivering good control performance (Lowrey et al., 2018; Zhong et al., 2013). We therefore propose to learn the value function of the MPC, which measures the total cost accrued over an infinite horizon, and use it as the terminal cost. For brevity, we refer the reader to Bøhn et al. (2021b), Lowrey et al. (2018), Zhong et al. (2013) for more information on this topic. Note that while we assume state-feedback for simplicity, the formulation could easily be extended with an estimator such as the moving horizon estimator which can be tuned in unison with the MPC (Nejatbakhsh Esfahani et al., 2021).

We further employ a modification to the MPC framework called event-triggered MPC, in which the OCP is not recomputed at every time step, but rather a recomputation policy $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}$ decides at every time step whether the OCP should be recomputed. Thus, not only the first input of the MPC input sequence u^{M}_t is applied to the plant, but rather a variable number of inputs $u^{\mathrm{M}}_{t:t+n}$, $n < N_t$ are applied sequentially at the corresponding time instance until the recomputation policy triggers the recomputation of the OCP at time step $t + n$. We will detail this recomputation policy in Section 3.2.1.

2.1.2. Note on feasibility

There is no explicit mechanism in our method that ensures the recursive feasibility of the MPC scheme, or the stability of the control algorithm. However, the control algorithm and tuning framework we present are agnostic wrt. the implementation of the underlying controllers. As such, one could modify the MPC scheme by adding e.g. assumptions on the form and magnitude of the disturbances, adding terminal constraints, or entirely replacing the MPC scheme with more complex formulations, e.g. robust MPC (Bemporad and Morari, 1999) or tube MPC (Mayne et al., 2005).

The prediction horizon is one of the most important tunable parameters to achieve stable control with the MPC scheme (Grüne and Pannek, 2011; Mayne et al., 2000). Learning approaches can therefore be an important tool in identifying prediction horizons that yield a stable control system. As discussed in Section 3.5, RL is an optimization procedure that seeks optimality wrt. its reward function, therefore, if it produces non-stabilizing solutions this would suggest that the learning problem itself is ill-posed.

2.1.3. The linear quadratic regulator

The LQR (Bertsekas, 1995) is a state-feedback controller that arises as the optimal solution to unconstrained control problems where the dynamics are linear, and the cost is quadratic. The role of the LQR in our control algorithm is to act as the linear feedback correction of the MPC, which can be applied to compensate for errors in the open loop predicted state trajectory between MPC recomputations. To accomplish this, we employ a time-varying LQR where the $\mathbf{A}_t$ and $\mathbf{B}_t$ matrices are obtained from the MPC scheme as the linearization of the MPC model each time it is computed[1]:

$$\mathbf{A}_{t+1:t+N_t}, \mathbf{B}_{t+1:t+N_t} = \text{linearize}(\hat{f}_{\theta^{\mathrm{M}}})\Big|_{x_{t+1:t+N_t}, u_{t:t+N_t-1}} \tag{7}$$

$$\mathbf{A}, \mathbf{B} = \text{linearize}(\hat{f}_{\theta^{\mathrm{M}}})\Big|_{x^s_t, u^s_t} \tag{8}$$

After the horizon end (i.e. when $t > i + N_i$ where i is the time of last MPC computation), we set the LQR dynamics matrices as the time-invariant matrices corresponding to the steady state (equilibrium) of the system x^s_t, u^s_t (8). The specific linearization procedure depends on the implementation of the dynamics model in the MPC, i.e. discrete vs continuous model and the accompanying discretization schemes. The $\mathbf{Q}$ and $\mathbf{R}$ cost-weighting matrices are initialized from the MPC objective as follows and then tuned further as described in Section 3.

$$\mathbf{Q} \leftarrow \frac{\partial^2 \ell_{\theta^{\mathrm{M}}}(x, u, \hat{p})}{\partial x^2}\Big|_{x^s_0, u^s_0},\ \mathbf{R} \leftarrow \frac{\partial^2 \ell_{\theta^{\mathrm{M}}}(x, u, \hat{p})}{\partial u^2}\Big|_{x^s_0, u^s_0} \tag{9}$$

In this paper, we focus on the discrete-time formulation of the LQR. The LQR control law (12) consists of a state-feedback gain matrix $\mathbf{K}$ that is derived from the dynamics matrices $\mathbf{A}$ and $\mathbf{B}$, the cost-weighting matrices $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{N}$, and the solution $\mathbf{S}$ to the discrete-time Riccati-Eq. (10):

$$\begin{aligned}\mathbf{S}_k = \mathbf{Q}_k + \mathbf{A}_k^\top \mathbf{S}_{k+1} \mathbf{A}_k - (\mathbf{A}_k^\top \mathbf{S}_{k+1} \mathbf{B}_k + \mathbf{R}_k) \\ (\mathbf{R}_k + \mathbf{B}_k^\top \mathbf{S}_{k+1} \mathbf{B}_k)^{-1} (\mathbf{B}_k^\top \mathbf{S}_{k+1} \mathbf{A}_k + \mathbf{R}_k^\top)\end{aligned} \tag{10}$$

$$\mathbf{K}_k = -(\mathbf{R}_k + \mathbf{B}_k^\top \mathbf{S}_{k+1} \mathbf{B}_k)^{-1} (\mathbf{B}_k^\top \mathbf{S}_{k+1} \mathbf{A}_k + \mathbf{R}_k^\top) \tag{11}$$

$$u^{\mathrm{L}}_k(x_k) = \pi^{\mathrm{L}}_{\theta^{\mathrm{L}}}(x_k) = -\mathbf{K}_k x_k \tag{12}$$

We consider in this paper the vector θ^{L} containing all the elements of the matrices $\mathbf{Q}, \mathbf{R}, \mathbf{N}$ as parameters of the LQR controller that can be optimized. The finite horizon LQR control problem is stated as:

$$\min_{x,u} \quad \sum_{k=0}^{N_i-1} \left[\|x_k\|^2_{\mathbf{Q}_k} + \|u_k\|^2_{\mathbf{R}_k} + 2 x_k^\top \mathbf{N}_k u_k \right], \tag{13}$$

$$\text{s.t.} \quad \mathbf{Q}_k - \mathbf{N}_k \mathbf{R}_k^{-1} \mathbf{N}_k^\top > 0,\ \mathbf{R}_k > 0 \tag{14}$$

$$x_{k+1} = \mathbf{A}_k x_k + \mathbf{B}_k u_k \tag{15}$$

where N_i is the optimization horizon and $\|x_k\|^2_{\mathbf{Q}_k} = x_k^\top \mathbf{Q}_k x_k$ is the quadratic form. We also consider the infinite horizon problem, i.e. $N_i = \infty$, in which the matrices are constant wrt. time and the discrete-time algebraic Riccati equation (DARE) is solved at its stationary point, yielding the solution $\mathbf{S}_\infty$ and consequent gain matrix $\mathbf{K}_\infty$.

2.2. Reinforcement learning

The system to be optimized in the RL framework (Sutton and Barto, 2018) is formulated as axMarkov decision process (MDP). The MDP is defined by a state space $s \in \mathcal{S}$, a set of actions $a \in \mathcal{A}$, a transition probability matrix $\mathcal{T}$ that governs the evolution of states as a function of time and actions, i.e. $s_{t+1} = \mathcal{T}(s_t, a_t)$, a reward function $R(s, a)$ that describes the desirability of the states of the system, and finally, the discount factor $\gamma \in [0, 1)$ (note the different limits from ρ) that describes the relative importance of immediate and future rewards. Note that rewards, R, are interchangeable with costs, C, through the substitution

[1] Note that the MPC solver generates these matrices when solving the OCP, so in practice they are free to obtain for this purpose.

$R = -C$ and changing maximization of the objective to minimization. We will use rewards when discussing RL as this is customary in RL.

The objective in RL is to develop a policy π_θ, i.e. a function that maps from states to actions (here parameterized by θ) that maximizes the expected discounted sum of rewards. In this paper, we use the policy gradient algorithm proximal policy optimization (PPO) (Schulman et al., 2017). For brevity, we refer the reader to the original paper (Schulman et al., 2017) for details on how PPO updates the parameters to be optimized. In general, policy gradient algorithms updates a parameterized stochastic policy directly in the parameter space, by sampling actions from the policy's action distribution and observing the outcomes in terms of states and rewards. Parameters are adjusted to increase the likelihood of actions leading to high rewards using gradient ascent with gradients from the policy gradient theorem (Sutton and Barto, 2018):

$$\theta \leftarrow \theta_{\text{old}} + \eta \nabla_\theta J(\theta) \tag{16}$$

where

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(t)\right] \tag{17}$$

$$G(t) = \sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, \pi_\theta(s_{t'})) \tag{18}$$

3. Method

The standard modus operandi for optimizing the parameters of a control algorithm using RL is as described in Section 2.2 to obtain the policy gradient of the control algorithm (which gives information on the sensitivity of the output to changes in its parameters), and give this along with control trajectories (i.e. state transitions and accompanying rewards from closed-loop trajectories) to the RL algorithm such that it can iteratively improve the parameters to yield higher rewards.

However, since the meta-parameters that we seek to optimize are non-differentiable wrt. the MPC's outputs, the required information is not readily available through the existing methods for obtaining the policy gradient of the MPC (Gros and Zanon, 2019; Edwards et al., 2021; Gros and Zanon, 2021). Therefore, we need to derive how these meta-parameters affect the MPC's outputs, and further how they interact with each other and the other parameters of the control algorithm (see Fig. 1). To this end, in Section 3.2 we argue for suitable choices of probability distributions for the meta-parameters, and subsequently derive the policy gradient of the control algorithm that includes the MPC scheme and these meta-parameters.

3.1. A state space with the Markov property

For RL theory to hold for the control system we wish to optimize, the state space needs to have the Markov property, i.e. future states should not depend upon past states given the current state. This section outlines such a Markovian state.

Consider the input sequence $u^{\text{M}}_{i:i+N_i-1}$ and the predicted state trajectory $\hat{x}_{i+1:i+N_i}$ computed by the MPC at time i. As discussed in Section 2.1.1, a variable number $n \in \{0, 1, \ldots, N_i - 1\}$ of these inputs are applied to the plant. Since n is not known a priori, the state vector x is not a sufficient state representation to yield the Markov property for the control system consisting of the recomputation policy, the horizon policy, and the MPC and LQR control laws. We define an augmented state s in (19) that contains the current plant state and exogenous variables, labeled $\bar{x}_t$ and $\hat{p}_t$, the state of the system, exogenous variables, and prediction horizon used when the last MPC computation took place, labeled $\bar{x}_i$, $\hat{p}_i$, and N_i, as well as the number of time steps since the MPC computation, $t - i$:

$$s_t = \left[\bar{x}_i, \hat{p}_i, N_i, \bar{x}_t, \hat{p}_t, t - i\right]^\top \tag{19}$$

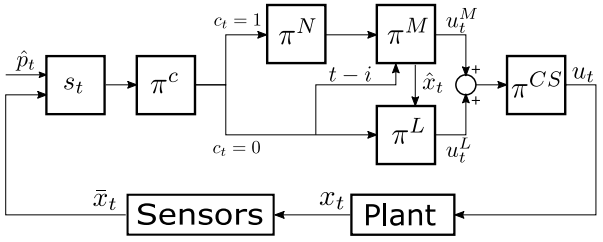


Fig. 1. An overview of the control algorithm. Not shown here is the connection of each policy's output to the RL algorithm that updates the policy's parameters.

where $\bar{x}_t$ has evolved from $\bar{x}_i$ according to the real system dynamics. When the MPC problem is recomputed, the deterministic transition $i \leftarrow t, \bar{x}_i \leftarrow \bar{x}_t, \hat{p}_i \leftarrow \hat{p}_t, N_i \leftarrow N_t$ takes place. The MPC and MPC plus LQR control laws deployed on the plant is then (20) and (21), respectively, while the total control system is defined as (22), where the $\text{proj}_{h_{\theta^{\text{M}}}}$ operator projects the control input onto the constraint vector $h_{\theta^{\text{M}}}$ (6):

$$\pi^{\text{M}}_{\theta^{\text{M}}}(s_t) = \begin{cases} u^{\text{M}}_{t-i}(\bar{x}_i, \hat{p}_i, N_i) & \text{, if } t - i < N_i \\ 0 & \text{, otherwise} \end{cases} \tag{20}$$

$$\pi^{\text{ML}}_{\theta^{\text{M,L}}}(s_t) = \begin{cases} u^{\text{M}}_{t-i}(\bar{x}_i, \hat{p}_i, N_i) + u^{\text{L}}_t(\hat{x}_{t-i} - \bar{x}_t) & \text{, if } t - i < N_i \\ u^{\text{L}}_t(x^s_t - \bar{x}_t) & \text{, if } t - i \geq N_i \end{cases} \tag{21}$$

$$\pi^{\text{CS}}_{\theta^{\text{M,L}}}(s_t) = \text{proj}_{h_{\theta^{\text{M}}}}\left(\pi^{\text{M}}_{\theta^{\text{M}}}(s_t) + \pi^{\text{ML}}_{\theta^{\text{M,L}}}(s_t)\right) \tag{22}$$

and where x^s_t is the steady state equilibrium of the system at time t. Note that the policies in (20) and (21) with the state s as input are the same policies defined in (2) and (12) where they select the required elements from s.

Assumption 1. The time-varying exogenous input variables $\hat{p}_t$ are generated by a Markovian process.

Proposition 1. *The state s has the Markov property, i.e. $P(s_{t+1}|s_t) = P(s_{t+1}|s_{0:t})$*

Proof. First, note that by definition from (1) x is Markovian given u:

$$x_{t+1} = f(x_t, u_t) \tag{23}$$

$$\Rightarrow P(x_{t+1}|x_t, u_t) = P(x_{t+1}|x_{0:t}, u_{0:t}) \tag{24}$$

However, the control law $\pi^{\text{CS}}_{\theta^{\text{M,L}}}$ (22) that determines u_t consists of $\pi^{\text{M}}_{\theta^{\text{M}}}$ (20) – which depends on the state $\bar{x}_i$, exogenous variables $\hat{p}_i$, and the prediction horizon N_i at the last MPC computation – and $\pi^{\text{M,L}}_{\theta^{\text{ML}}}$ (21), which depends on the current state x_t and the $(t - i)$th element of the MPC's predicted state trajectory, and x^s_t which depends on $\hat{p}_t$. Therefore, with s as defined in (19) we have:

$$P(u_t|\pi^{\text{CS}}_{\theta^{\text{M,L}}}, s_t) = P(u_t|\pi^{\text{CS}}_{\theta^{\text{M,L}}}, s_{0:t}) \tag{25}$$

Finally, note that the MPC input sequence $u^{\text{M}}_{i+1:i+N_i-1}$ and the predicted state trajectory $\hat{x}_{i:i+N_i}$ follows from $\pi^{\text{M}}_{\theta^{\text{M}}}$ and its arguments (which are all known), and as such does not need to be contained in s for s to be Markovian:

$$\Rightarrow P(s_{t+1}|s_t) = P(s_{t+1}|s_{0:t}) \quad \square \tag{26}$$

3.2. Policies

We use the notation $\pi_\theta(s)$ to label a deterministic function of the state s that is parameterized by θ, and $\pi_\theta(\cdot|s)$ to label the stochastic version of the same function. The notation $\tilde{\cdot}$ signifies that the value is drawn from the policy's probability distribution. We denote a probability distribution by P, and corresponding log probability by $\log P$. For brevity, we omit listing input-independent conditional variables as arguments (e.g. covariance) of the distributions.

3.2.1. The recomputation policy

The recomputation policy $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}$ decides on each step whether the MPC problem should be recomputed, or if the previously computed solution is still acceptable. In other words, it is a binary variable that chooses among two different options: recompute or not. As such, we model it as a Bernoulli-distributed random variable, where the policy outputs the logit of the probability that the OCP should be recomputed (27), from which we can deduce the probability of not recomputing. We label the output of the recomputation policy $c \in \{0, 1\}$, which has the probability mass function (PMF) (28).

$$w = \frac{1}{1 + \exp\left(-\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(s)\right)} \tag{27}$$

$$P^{\mathrm{c}}(c|s) = \begin{cases} 1 - w & , \text{ if } c = 0 \\ w & , \text{ if } c = 1 \end{cases} \tag{28}$$

The log probability can be expressed as follows:

$$\log P^{\mathrm{c}}(c|s) = c \log(w) + (1 - c) \log(1 - w) \tag{29}$$

Finally, the policy gradient of the recomputation policy is:

$$\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(c|s) = \nabla_{\theta^{\mathrm{c}}} \left(c \log(w) + (1 - c) \log(1 - w)\right) \tag{30}$$

$$= -c \nabla_{\theta^{\mathrm{c}}} \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(s) w + \nabla_{\theta^{\mathrm{c}}} \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(s)(1 - c)(-1 - w) \tag{31}$$

$$= \nabla_{\theta^{\mathrm{c}}} \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(s)(c - 1 - w) \tag{32}$$

3.2.2. The horizon policy

The MPC prediction horizon N is a positive integer, which we model with the GP-2 variant of the generalized Poisson distribution (GPD) (Wang and Famoye, 1997). Other models we considered but decided against include categorical classification models, as they do not consider the ordinal information of the horizon variable, and the standard Poisson distribution, which we found to be too inflexible due to its mean and variance being equal. The horizon model outputs the rate parameter of the GPD, μ, as a function of s, while the dispersion parameter α is learned as an input-*independent* variable.

$$P^{\mathrm{N}}(N|s) = \left(\frac{\mu}{1 + \alpha\mu}\right)^{\mathrm{N}} \frac{(1 + \alpha N)^{N-1}}{N!} \exp\left(-\frac{\mu(1 + \alpha N)}{1 + \alpha\mu}\right) \tag{33}$$

$$\mathbb{E}(N) = \mu \tag{34}$$

$$\mathbb{V}(N) = \mu(1 + \alpha\mu)^2 \tag{35}$$

In this model, α is restricted according to $1 + \alpha\mu > 0$ and $1 + \alpha N > 0$. To address these constraints, we introduce two hyperparameters $N_{\min}$ and $N_{\max}$ that correspond to the minimum and maximum horizons that the MPC should operate with. To constrain the rate parameter μ we apply the hyperbolic tangent function, denoted $\tanh \in [-1, 1]$, and then linearly scale it to the limits defined by $N_{\min}$ and $N_{\max}$. Finally, the learned α parameter is clipped according to the restrictions outlined above:

$$\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s) = \mathrm{scale}\left(\tanh(\mu), N_{\min}, N_{\max}\right) \tag{36}$$

$$\alpha_{\mathrm{new}} = \max\left(\alpha, -\frac{1}{N_{\max}}\right) \tag{37}$$

The log probability of the GPD is:

$$\begin{aligned} \log P^{\mathrm{N}}(N|s) = & N \log\left(\frac{\mu}{1 + \alpha\mu}\right) + (N - 1) \log(1 + \alpha N) \\ & - \frac{\mu(1 + \alpha N)}{1 + \alpha\mu} - \log(N!) \end{aligned} \tag{38}$$

and the policy gradient of the horizon policy follows by application of the gradient and some elementary calculus:

$$\begin{aligned} \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(N|s) = \nabla_{\theta^{\mathrm{N}}} \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s) \Bigg(& \frac{N}{\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)} + \\ & \frac{\alpha}{1 + \alpha\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)} + \frac{1 + \alpha N + \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)(\alpha(1 + N + \alpha N) + 1)}{1 + 2\alpha\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s) + (\alpha\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s))^2} \Bigg) \end{aligned} \tag{39}$$

There are several techniques to sample from this distribution (Demirtas, 2017). We favor the normal approximation sampling technique for its low run-time complexity: With a sufficiently high rate parameter (i.e. $\mu \gtrapprox 10$), the GPD is approximately normally distributed with mean and variance as given in (34) and (35):

$$\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(N|s) \approx \mathrm{clip}(\lfloor \mu + \sqrt{\mu(1 + \alpha\mu)^2}\zeta + 0.5 \rfloor, N_{\min}, N_{\max}), \ \zeta \sim \mathcal{N}(0, 1) \tag{40}$$

When optimizing the horizon policy in isolation (that is, not together with the rest of the control system) we find that faster convergence and more stable solutions are obtained by learning on an augmented MDP where every action selected by the policy is repeated $d > 1$ times. Hence the policy only senses every d'th state, and the rewards it receives are the cumulative reward over every step of the control system in the d steps. This technique is known as "frame-skip" in RL and is an effective method to enhance learning for problems with discrete actions, see e.g. learning to play atari-games (Mnih et al., 2015), but also for continuous control (Kalyanakrishnan et al., 2021). While the exact mechanisms behind the improvements stemming from frame-skipping is not fully understood, it is clear that in certain problems it increases the signal-to-noise ratio of every data sample, which simplifies the credit assignment problem. When using $d > 1$ during exploration, we use $d = 1$ during exploitation (evaluation) as this increases performance. We find that the horizon policy is not sensitive to the exact value of d, and all values in $d \in [2, T]$ generally accelerate learning. Finally, note that when optimizing the horizon policy and the recomputation policy together, the latter electing to not recompute the MPC will naturally enforce a form of frame-skipping.

3.2.3. The controller policies

To ensure sufficient exploration we formulate a stochastic version of the MPC and the MPC plus LQR control laws, modeling them as Gaussian random variables (41). The mean is then the output of the control laws, and the covariance of each controller is a learned input-*independent* variable of the RL algorithm. To be concise, we use $*$ in place of the superscript for the control laws as in (41). We will first develop these policies for the MPC scheme with a given horizon N, where the output is made stochastic as follows:

$$\pi^*_{\theta^*}(u^*|s, N) = \mathcal{N}\left(\pi^*_{\theta^*}(s, N), \Sigma^*\right), \ * \in \{\mathrm{M}, \mathrm{ML}\} \tag{41}$$

$$P^{\mathrm{u}}(u^*|s, N) = \frac{\exp\left(-\frac{1}{2}\|u^* - \pi^*_{\theta^*}(s, N)\|^2_{\Sigma^{*-1}}\right)}{\sqrt{(2\pi)^{n_u} \det(\Sigma^*)}} \tag{42}$$

Note that the argument N to the policies is redundant (and thus these policies are equivalent to those in Section 3.1), as N_i in s will always reflect the latest N that is decided by the horizon policy ahead of control law calculation. We use this argument here to clarify the derivations. The log probability of the Gaussian distribution and the policy gradient is:

$$\begin{aligned} \log P^{\mathrm{u}}(u^*|s, N) = -\frac{1}{2} \Big(& \|u^* - \pi^*_{\theta^*}(s, N)\|^2_{\Sigma^{*-1}} \\ & + n_u \log(2\pi) + \log(\det(\Sigma^*)) \Big) \end{aligned} \tag{43}$$

$$\nabla_{\theta^*} \log \pi^*_{\theta^*}(u^*|s, N) = \Sigma^{*^{-1}}(u^* - \pi^*_{\theta^*}(s, N)) \nabla_{\theta^*} \pi^*_{\theta^*}(s, N) \tag{44}$$

Then, note that the adaptive-horizon MPC control law is a distribution of the MPC schemes over the range of prediction horizons, with weights P^{N} assigned by the horizon policy. We first query the horizon policy for which prediction horizon to employ, and then the solution to the MPC problem (3) is computed with the selected horizon:

$$\pi^{\mathrm{MN}}_{\theta^{\mathrm{M,N}}}(s) = \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}\left(x_t, \hat{p}_t, \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)\right) \tag{45}$$

where the superscript N indicates that this is the adaptive-horizon MPC policy. Using the indicator function:

$$\mathbb{1}_A = \begin{cases} 1, & \text{if } A \\ 0, & \text{otherwise} \end{cases} \tag{46}$$

we can formulate the probability distributions and log distributions for the stochastic adaptive-horizon control policies:

$$P^{\mathrm{Nu}}(u^*|s) = \sum_{N=N_{\min}}^{N_{\max}} P^{\mathrm{N}}(N|s)P^{\mathrm{u}}(u^*|s,N), *\in \{\mathrm{MN}, \mathrm{MLN}\} \tag{47}$$

$$\log P^{\mathrm{Nu}}(u^*|s) = \log\left(\sum_{N=N_{\min}}^{N_{\max}} P^{\mathrm{N}}(N|s)P^{\mathrm{u}}(u^*|s,N)\right) \tag{48}$$

$$= \sum_{N=N_{\min}}^{N_{\max}} \mathbb{1}_{N=\tilde{N}}\left(\log P^{\mathrm{N}}(N|s) + \log P^{\mathrm{u}}(u^*|s,N)\right) \tag{49}$$

$$= \log P^{\mathrm{N}}(\tilde{N}|s) + \log P^{\mathrm{u}}(u^*|s,\tilde{N}) \tag{50}$$

where the log operator can be applied inside the summation over the prediction horizons in (49), because we know the value $\tilde{N}$ of the horizon variable N that is sampled in advance of the calculation of the control laws (Friedman et al., 2001).

3.2.4. The complete policy

We collect all the parameters of the meta-parameter-deciding recomputation and horizon policies described above, and the parameters of the controllers into a single parameter vector $\theta = \left[\theta^{\mathrm{c}}, \theta^{\mathrm{N}}, \alpha, \theta^{\mathrm{M}}, \theta^{\mathrm{L}}, \Sigma^{\mathrm{M}}, \Sigma^{\mathrm{ML}}\right]^\top$, and define the complete policy π_θ, whose input is the state s and output is the action $a = \left[c, N, u^{\mathrm{M}}, u^{\mathrm{ML}}\right]$

Section 3.2.1 presents the recomputation problem of the MPC and the policy that decides when to compute it. We now view the recomputation policy $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}$ as selecting the active controller between the two control laws. It is then clear that π_θ is a mixture distribution between the Gaussian control policies where the weights of the mixture are assigned by the Bernoulli recomputation policy $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}$. We label the probability distribution of the complete policy π_θ as P^{a} and define it and the log probability as follows:

$$P^{\mathrm{a}}(\tilde{a}|s) = P^{\mathrm{c}}(0|s)P^{\mathrm{Nu}}(\tilde{u}^{\mathrm{MLN}}|s) + P^{\mathrm{c}}(1|s)P^{\mathrm{Nu}}(\tilde{u}^{\mathrm{MN}}|s) \tag{51}$$

$$\log P^{\mathrm{a}}(\tilde{a}|s) = \mathbb{1}_{\tilde{c}=0}\left(\log P^{\mathrm{c}}(0|s) + \log P^{\mathrm{Nu}}(\tilde{u}^{\mathrm{MLN}}|s)\right) + \mathbb{1}_{\tilde{c}=1}\left(\log P^{\mathrm{c}}(1|s) + \log P^{\mathrm{Nu}}(\tilde{u}^{\mathrm{MN}}|s)\right) \tag{52}$$

$$= \mathbb{1}_{\tilde{c}=0}\left(\log P^{\mathrm{c}}(0|s) + \log P^{\mathrm{N}}(\tilde{N}_i|s) + \log P^{\mathrm{u}}(\tilde{u}^{\mathrm{ML}}|s,\tilde{N}_i)\right) + \mathbb{1}_{\tilde{c}=1}\left(\log P^{\mathrm{c}}(1|s) + \log P^{\mathrm{N}}(\tilde{N}|s) + \log P^{\mathrm{u}}(\tilde{u}^{\mathrm{M}}|s,\tilde{N})\right) \tag{53}$$

where we again used the fact that we know the values of the sampled variables $\tilde{c}, \tilde{N}$ to take the logarithm of the sums in (52) and (53), and $\tilde{N}$ represents the output of the horizon policy at the current step t in the case the recomputation policy signals to recompute the MPC. The policy gradient of the complete policy is then:

$$\nabla_\theta \log \pi_\theta(\tilde{a}|s) = \nabla_\theta \log P^{\mathrm{a}}(\tilde{a}|s) \tag{54}$$

$$= \mathbb{1}_{\tilde{c}=0}\Big(\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(0|s) + \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(\tilde{N}_i|s) + \nabla_{\theta^{\mathrm{M,L}}} \log \pi^{\mathrm{ML}}_{\theta^{\mathrm{M,L}}}(\tilde{u}^{\mathrm{ML}}|s,\tilde{N})\Big) + \mathbb{1}_{\tilde{c}=1}\Big(\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(1|s) + \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(\tilde{N}|s) + \nabla_{\theta^{\mathrm{M}}} \log \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(\tilde{u}^{\mathrm{M}}|s,\tilde{N})\Big) \tag{55}$$

$$= \mathbb{1}_{\tilde{c}=0}\Big(\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(0|s) + \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(\tilde{N}_i|s) + (\Sigma^{\mathrm{ML}})^{-1}(\pi^{\mathrm{ML}}_{\theta^{\mathrm{M,L}}}(s,\tilde{N}) - \tilde{u}^{\mathrm{ML}})(\nabla_{\theta^{\mathrm{M}}}\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s,\tilde{N}_i) + \nabla_{\theta^{\mathrm{L}}}\pi^{\mathrm{L}}_{\theta^{\mathrm{L}}}(s))\Big) + \mathbb{1}_{\tilde{c}=1}\Big(\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(1|s) + \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(\tilde{N}|s) + (\Sigma^{\mathrm{M}})^{-1}(\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s,\tilde{N}) - \tilde{u}^{\mathrm{M}})\nabla_{\theta^{\mathrm{M}}}\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s,\tilde{N})\Big) \tag{56}$$

With this policy one can optimize the two meta-parameters we employ, as well any other parameter of the MPC and LQR controllers jointly, by providing the gradient of these controllers wrt. the parameters. In the RL context, we show how one can obtain these gradients for the LQR controller in Section 3.3. For the gradient of the MPC see e.g. Gros and Zanon (2019, 2021).

3.3. Optimizing LQR with reinforcement learning

3.3.1. The time-invariant case

To apply the policy-gradient theorem to tune the LQR, we need to compute the gradients of the $\mathbf{K}$-matrix wrt. the $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{N}$ matrices. As Eqs. (10) and (11) show, the feedback matrix $\mathbf{K}$ is only implicitly defined in terms of the these matrices, and so direct differentiation of Eqs. (10) and (11) is not possible. Since we obtain the system matrices $\mathbf{A}$ and $\mathbf{B}$ directly from the MPC scheme and they have a clear purpose in making the MPC and LQR objectives compatible, we assume that they are fixed wrt. $\mathbf{Q}, \mathbf{R}, \mathbf{N}$ such that $\nabla_{\mathbf{Q},\mathbf{R},\mathbf{N}}\mathbf{A} = \nabla_{\mathbf{Q},\mathbf{R},\mathbf{N}}\mathbf{B} = 0$:

Assumption 2. The weighting matrices $\{\mathbf{Q}, \mathbf{R}, \mathbf{N}\}$ values are such that the DARE has a solution $\mathbf{S}_\infty$.

Proposition 2. *We flatten the matrices $\mathbf{S}, \mathbf{K}, \mathbf{Q}, \mathbf{R}$ and $\mathbf{N}$ into vectors, and organize them as follows: $y = \{\mathbf{S}, \mathbf{K}\}$ and $z = \{\mathbf{Q}, \mathbf{R}, \mathbf{N}\}$, such that the DARE and $\mathbf{K}$-matrix equations can be written on the vector form $F(y, z) = 0$. The gradient of $\mathbf{K}$ wrt. the weighting matrices $\mathbf{Q}, \mathbf{R}, \mathbf{N}$ can then be found as:*

$$\frac{\partial y}{\partial z} = -\frac{\partial F}{\partial y}^{-1}\frac{\partial F}{\partial z} = \nabla_{\mathbf{Q},\mathbf{R},\mathbf{N}}\mathbf{S}_\infty, \mathbf{K}_\infty \tag{57}$$

Proof. We rewrite (10)–(11) on the general vector form $F(y, z) = 0$ where $y = \{\mathbf{S}, \mathbf{K}\}$ and $z = \{\mathbf{Q}, \mathbf{R}, \mathbf{N}\}$.

$$\mathbf{A}^\top\mathbf{S}\mathbf{A} - \mathbf{S} - (\mathbf{A}^\top\mathbf{S}\mathbf{B} + \mathbf{N})\mathbf{K} + \mathbf{Q} = 0 \tag{58}$$

$$(\mathbf{B}^\top\mathbf{S}\mathbf{B} + \mathbf{R})\mathbf{K} - (\mathbf{B}^\top\mathbf{S}\mathbf{A} + \mathbf{N}^\top) = 0 \tag{59}$$

We then apply the implicit function theorem (IFT) which states that:

$$\frac{\partial F}{\partial y}\frac{\partial y}{\partial z} + \frac{\partial F}{\partial z} = 0 \Rightarrow \frac{\partial y}{\partial z} = -\frac{\partial F}{\partial y}^{-1}\frac{\partial F}{\partial z} \tag{60}$$

These gradients are easily obtained with automatic differentiation software. □

Assumption 2 holds if $\mathbf{Q} - \mathbf{N}\mathbf{R}^{-1}\mathbf{N}^\top > 0$ and $\mathbf{R} > 0$, and further it is required that the symplectic pencil of the problem has eigenvalues sufficiently far from the unit circle, which is satisfied if the pair $(\mathbf{A}, \mathbf{B})$ is stabilizable and the pair $(\mathbf{A}, \mathbf{Q})$ is detectable (Laub, 1979). These conditions can be ensured by estimating the gradients as described above, and then solving axsemidefinite program (SDP) using the estimated gradients subject to these constraints. However, for simplicity and to avoid constrained optimization, we set $\mathbf{N} = 0$, simplifying the constraints on the positive (semi)-definiteness to $\mathbf{Q} \succ 0$ and $\mathbf{R} > 0$. Further, we write $\mathbf{Q}$ and $\mathbf{R}$ in terms of their Cholesky decompositions: $\mathbf{Q} = \mathbf{Q}_C^\top\mathbf{Q}_C$, $\mathbf{R} = \mathbf{R}_C^\top\mathbf{R}_C$, and let the RL algorithm adjust the elements of $\mathbf{Q}_C$ and $\mathbf{R}_C$.

3.3.2. Time-varying gradients

In what follows we will assume $\mathbf{N} = 0$, for simplicity of the derivation and expressions (and the constraint reasons outlined above). We find the gradients with respect to p, where p is an arbitrary scalar element of the $\mathbf{Q}$ and $\mathbf{R}$ matrices. The full gradients $\nabla_{\mathbf{Q},\mathbf{R}}\mathbf{K}_k$ can then be found by solving (63) for each parameter of $\mathbf{Q}$ and $\mathbf{R}$ and arranging the results into the appropriate matrix structure. The derivations involve repeated application of the chain rule and the following relation for the gradient of the matrix inverse, where we define $\mathbf{E}$ for convenience:

$$\mathbf{E}_k = \mathbf{R} + \mathbf{B}_k^\top\mathbf{S}_{k+1}\mathbf{B}_k, \ \nabla_p\mathbf{E}^{-1} = -\mathbf{E}^{-1}\nabla_p\mathbf{E}\mathbf{E}^{-1} \tag{61}$$

$$\nabla_p\mathbf{S}_k = \nabla_p\mathbf{Q} + \mathbf{A}_k^\top(-\nabla_p\mathbf{S}_{k+1}\mathbf{B}_k\mathbf{E}_k^{-1}\mathbf{B}_k^\top\mathbf{S}_{k+1}\mathbf{A}_k + \mathbf{S}_{k+1}\mathbf{B}_k(\mathbf{E}_k^{-1}\nabla_p\mathbf{E}_k\mathbf{E}_k^{-1}\mathbf{B}_k^\top\mathbf{S}_{k+1}\mathbf{A}_k - \mathbf{E}_k^{-1}\mathbf{B}_k^\top\nabla_p\mathbf{S}_{k+1}\mathbf{A}_k) + \nabla_p\mathbf{S}_{k+1}\mathbf{A}_k) \tag{62}$$

$$\nabla_p\mathbf{K}_k = -\mathbf{E}_k^{-1}\nabla_p\mathbf{E}_k\mathbf{E}_k^{-1}\mathbf{B}_k^\top\mathbf{S}_{k+1}\mathbf{A}_k + \mathbf{E}_k^{-1}\mathbf{B}_k^\top\nabla_p\mathbf{S}_{k+1}\mathbf{A}_k \tag{63}$$

Algorithm 1: Control Algorithm with Learning

while *Running* **do**
- Initialize episode: $x_0, \hat{p}_0$
- Compute initial MPC solution:
  $u^{\mathrm{M}}_{0:N_{\max}-1},\ \hat{x}_{1:N_{\max}} = \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(x_0, \hat{p}_0, N_{\max})$
- Execute first MPC input: u^{M}_0
- Calculate LQR system matrices: $\mathbf{A}_{1:N_{\max}}, \mathbf{B}_{1:N_{\max}}$
- **for** $t = 1, 2, \dots, T$ **do**
  - Measure system state: $\bar{x}_t$
  - **if** $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(c_t \mid s_t)$ draws $\tilde{c}_t = 1$ **then**
    - Compute MPC solution:
      $\tilde{u}^{\mathrm{MN}}_{t:t+N_t-1},\ \hat{x}_{t+1:t+N_t} = \pi^{\mathrm{MN}}_{\theta^{\mathrm{MN}}}(\tilde{u}^{\mathrm{MN}} | s_t)$
    - Execute control input: $\tilde{u}^{\mathrm{MN}}_t$
    - Calculate LQR system matrices: $\mathbf{A}_{t:t+N_t}, \mathbf{B}_{t:t+N_t}$
    - Update last computation states: $i \leftarrow t$
  - **else**
    - Compute input: $\tilde{u}^{\mathrm{CS}}_t = \pi^{\mathrm{CS}}_{\theta^{\mathrm{M,L,N}}}(u^{\mathrm{CS}}_t | s_t)$
    - Execute input: $\tilde{u}^{\mathrm{CS}}_t$
  - Collect data: $D \leftarrow (s_t, \tilde{a}_t, R(s_t, \tilde{a}_t), s_{t+1})$
  - **if** size$(D) = Z$ **then**
    - **for** $e = 1, \dots,$ NumEpochs **do**
      - **for** $\mathcal{B} \in D$ **do**
        - Evaluate PPO objective over minibatch $\mathcal{B}$
        - Update parameters (16)
    - Empty dataset: $D = \{\}$

Note that these gradients could easily be extended with time-varying $\mathbf{Q}$ and $\mathbf{R}$ matrices. The time-varying gradients of the LQR policy are then as follows:

$$\nabla_{\mathbf{Q},\mathbf{R}}\mathbf{K}_k = \begin{cases} \nabla_p \mathbf{K}_\infty \big|_{p\in\mathbf{Q},\mathbf{R}} & , \text{ if } k \geq N_i \\ \nabla_p \mathbf{K}_k \big|_{p\in\mathbf{Q},\mathbf{R}} & , \text{ otherwise} \end{cases} \tag{64}$$

$$\nabla_{\theta^{\mathrm{L}}} \pi^{\mathrm{L}}_{\theta^{\mathrm{L}}}(s_t) = \nabla_{\mathbf{Q},\mathbf{R}}\mathbf{K}_{t-i}(\hat{x}_t - \bar{x}_t) \tag{65}$$

3.4. Summary of control algorithm with learning

The complete control algorithm is outlined in Algorithm 1. This shows the control algorithm in the exploration phase, but the exploitation phase is identical with the two exceptions: (1) there is no data collection (and therefore no parameter updates), and (2) we use the deterministic version (i.e. the mode of the probability distribution) of all policies except for the recomputation policy. The Bernoulli distribution of the recomputation policy does not generally tend to quasi-determinism as exploration settles, unlike the other policies. As such, we find that the deterministic version of this policy has worse control performance and less consistency in the plant response than the stochastic version which we are in fact optimizing the response for.

The system to be optimized runs in an episodic fashion, and every Z steps the RL algorithm updates the parameters of the control system.

3.5. Reward function

The RL reward function codifies the behavior that the control algorithm is designed to exhibit, wrt. stability, control performance, and computational complexity. As RL is a trial-and-error based optimization approach, the control algorithm will necessarily have to attempt risky maneuvers and experience the consequences in order to learn how to control the system. However, with a feasible composition of the learning problem in terms of hyperparameters and expressive power of the learned components, the end result of the converged behavior should be stable and yield good control performance. A failure to achieve this would therefore indicate a misalignment between choice of reward function, and the control design requirements. With this in mind, we formulate a reward function on the following form:

$$R(s_t, a_t) = R_\ell(s_{t+1}) + \lambda_h(T-t)R_h(s_{t+1}) + \lambda_c R_c(a_t)R_N(a_t) \tag{66}$$

Here, λ_h and λ_c are weighting factors that represents the relative importance of the different terms. R_ℓ is the control performance term that incentivizes the RL policy to achieve good control performance. We set it to be the same as the stage cost from the MPC objective, i.e. $R_\ell = -\ell$ but this is not a strict requirement. R_h is a term that indicates whether any system constraints are violated. If using hard constraints as is the case in this paper, this term is binary and is further weighted by $T - t$, that is, the number of time steps remaining in the episode, since the episode is prematurely terminated if any constraints are violated. If the system to be controlled has soft constraints, the R_h term could be made continuous. The R_c term indicates whether the MPC was computed at the current step ($R_c = 1$), and as such it should reflect the relative computational complexity of the two modes of the control system. Finally, R_N represents the computational complexity of the MPC as a function of the prediction horizon N. We assume that the computational complexity grows linearly in the prediction horizon, i.e. $R_N(N_t) = N_t$, as a lower bound for the true complexity. We favor a lower bound as this would bias the RL policy towards better control performance rather than lower computation. The relationship between the prediction horizon and the computational complexity depends upon the algorithms used in the MPC implementation. We employ an interior point method, which under the assumption of local convergence and a guess of an initial solution that is reasonable, generally yields linear complexity (Rao et al., 1998), while other methods such as active set methods typically yield quadratic growth in computational complexity (Lau et al., 2015).

3.6. Initialization of the learning procedure

How to initialize the control algorithm, that is, defining its behavior before any learning takes place is a question that has several valid answers. One could favor the most computationally expensive initialization, i.e. compute MPC every step with the maximum horizon, which without any prior knowledge about the task would be the "safest" initialization that is most likely to give the best control performance. This is however not necessarily the initialization that would facilitate the learning process most optimally, and might trap the learned components in local minima. To simplify the learning problem one might therefore instead favor initializing the components in the center of their operating range and with high entropy (wrt. its output) such that exploration is maximal.

We offered some guidance in the case of the LQR in Section 2.1.3, i.e. the LQR should be initialized such that it is compatible with the MPC. We view the purpose of the recomputation policy as finding instances where comparable (or better) control performance can be achieved without computing the MPC, and as such, we choose to initialize it to emulate the MPC paradigm and with a high probability elect to compute the MPC. For the horizon policy, we initialize it equal to the best performing fixed horizon MPC scheme. Initializing the policies can be done by adjusting the bias terms of the policy outputs:

$$\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}} \leftarrow -\log\left(\frac{1}{c_{\mathrm{init}}} - 1\right) \tag{67}$$

$$\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}} \leftarrow \tanh^{-1}\left(\frac{(N_{\max} - N_{\min})(N_{\mathrm{init}} - (-1))}{1-(-1)} + N_{\min}\right) \tag{68}$$

4. Numerical results

This section illustrates the proposed control method as outlined in Section 3 on the simulated inverted pendulum system. Additionally, to aid in highlighting the contributions of the different meta-parameters of the control algorithm we also present experiments for each meta-parameter in isolation.

We set $N_{\min} = 1$ and $N_{\max} = 40$, and note that this horizon does not cover a full swing-up maneuver of the pendulum (requiring $N > 100$). The maximum horizon is chosen to emulate the effects of a computationally limited embedded hardware platform. The weighting terms of the reward function (66) are set to $\lambda_h = -10$ and $\lambda_c = 10^{-2}$. We chose the value of λ_h by considering the total control cost of a typical episode, and then setting the value of λ_h such that violating a constraint (thus ending the episode) will give a higher total cost than continuing the trajectory within the constraints until episode end. Finally, λ_c was set such that the computation term account for approximately 8% of the total cost of the standard MPC scheme (i.e. MPC computed at every step) with the highest prediction horizon. This value is significant enough to leave some room for optimization, while not being so large that it dominates the objective and makes gains trivial to achieve by simply reducing the computation.

While our method supports tuning parameters internal to the MPC (e.g. cost functionals), doing so requires reevaluating the log-probabilities and recomputing the OCP for every data sample after every parameter update (when using minibatches or several passes over a dataset). This adds considerable computation, and optimization of these parameters with RL has been successfully demonstrated in previous works (Amos et al., 2018; Gros and Zanon, 2021). Therefore, since this paper focuses on optimizing the meta-parameters of the MPC scheme (that is, the prediction horizon and when to compute) we do not tune any parameters internal to the MPC in this example. We do however optimize the parameters of the LQR to illustrate the concept, as the LQR is less costly to evaluate. Finally, we omit learning the value function of the MPC, as it adds considerable complexity to the experiments, and was found to not add much benefit for control of similar systems in Bøhn et al. (2021b).

4.1. The inverted pendulum system

The inverted pendulum system is a classic control task in which a pendulum, consisting of a rigid rod with a mass at the end, is mounted on top of a cart that is fixed on a track. The control system exerts a horizontal force on the cart, which moves the cart back and forth on the track, which subsequently swings the pendulum. The control objective is to stabilize the pendulum in the up-position and position the cart at the position reference while respecting the constraint that the cart's position is limited by the physical size of the track. The dynamics of the system are highly nonlinear, and further, the system is unstable, meaning that a controller is necessary to guide the system to stable conditions and then to maintain the stability. We perturb the parameters of the MPC dynamics model such that its dynamics differ from the plant dynamics, as shown in Table 1, and thus the LQR is a useful addition to correct for prediction errors in between the MPC computations. The state x consists of the cart position ψ and velocity v, pendulum angle ϕ, and angular velocity ω.

Each episode lasts a maximum of 150 steps (i.e. T=150), or until the position constraint (73) is violated. We sample initial conditions according to (72), and a position reference (71) that is redrawn every 50 steps, where $\mathcal{U}$ is the uniform distribution. The MPC objective is defined as (74), i.e. minimize the kinetic energy of the system and the distance of the cart to the position reference, while maximizing the potential energy.

$$\dot{v} = \frac{mg\sin\phi\cos\phi - \frac{7}{3}\left(u + ml\omega^2\sin\phi - \mu_c v\right) - \frac{\mu_p\omega\cos\phi}{l}}{m\cos^2\phi - \frac{7}{3}M} \tag{69}$$

$$\dot{\omega} = \frac{3}{7l}\left(g\sin\phi - \dot{v}\cos\phi - \frac{u_p\omega}{ml}\right), \dot{\phi} = \omega,\ \dot{\psi} = v \tag{70}$$

$$x = [\psi, v, \phi, \omega]^\top,\ \psi_r(t) \sim \mathcal{U}(-1, 1) \tag{71}$$

$$x_0 \sim [0, \mathcal{U}(-1, 1), \mathcal{U}(-\pi, \pi), \mathcal{U}(-1, 1)]^\top \tag{72}$$

$$x_t^s = \left[\psi_r(t), 0, 0, 0\right]^\top,\ -5 \le u_t \le 5,\ -2 \le \psi_t \le 2 \tag{73}$$

$$\ell(x_t, u_t, \hat{p}_t) = \mathrm{E_k} - 10\mathrm{E_p} + 10(\psi_t - \psi_r(t))^2 + 0.1(u_t - u_{t-1})^2 \tag{74}$$

Table 1
Parameters of the inverted pendulum system.

Name	Plant	MPC	Description
m	0.1	0.2	Mass of pendulum
M	1.1	1.5	Mass of cart and pendulum
g	9.81	9.81	Gravitational constant
l	0.25	0.25	Half the length of the pendulum
μ_c	0.01	0.01	Friction coefficient between track and cart
μ_p	0.001	0.001	Friction coefficient between pendulum and cart
Δ_t	0.04	0.04	Discretization step size in seconds

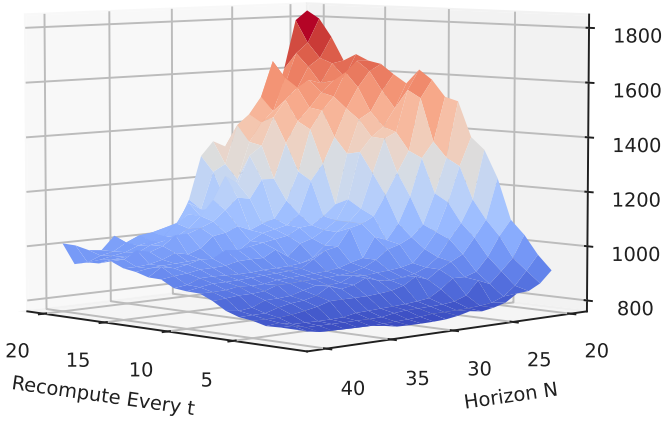


Fig. 2. Total cost over the evaluation set for the MPC as a function of fixed horizons and fixed recomputation schedules. The minimum is found at horizon $N = 31$ and recompute every step with a cost of 781.

4.2. Training and evaluation

The RL hyperparameters, code used to train and evaluate models, as well as the models presented in this section are available at Bøhn (2021). For the horizon experiment we use a frame-skip of $d = 10$, and for the other experiments we use a varying $d \in [1, 2, 3, 4]$ that is drawn at the start of every episode (with $d = 1$ when evaluating).

We initialize the recomputation policy using (67) to compute the MPC with 90% probability, thus within any two time steps there is a 99% probability that the MPC is computed. It therefore initially mimics the traditional MPC scheme closely. The horizon policy is initialized at the best performing fixed-horizon (Fig. 2), i.e. $N = 31$, using Eq. (68).

To evaluate the performance of the control system governed by the RL policy, we construct a "test-set" consisting of 25 episodes where all stochastic elements are drawn in advance (i.e. initial conditions and position references) such that the episodes are consistent across evaluations. This gives us an objective way to compare and order policies on their performance, and to compare against the MPC baselines. We set the cost objective (1) $C = -R$, and evaluate models based on the total sum of costs over the episodes in the test-set.

Moreover, for every experiment, we report the average over five random initial seeds (referred to as models in the subsequent discussion), which impact the initializations of neural networks (NNs) and the randomness in exploration and episodes. Fig. 2 shows the total cost of the control algorithm baselines as a function of a static prediction horizon and recomputation schedule. The minimum cost is achieved with a prediction horizon of $N = 31$ and a schedule of recomputing at every step. It is important to note that while this figure indicates that the optimization landscape as a function of these two variables is monotonic and amenable to optimization, the reward landscape as a function of the parameters θ (which in this example consists of the parameters of the LQR and the parameters of the recomputation and horizon NNs) is likely very different — containing many valleys and hills to overcome in order to minimize the cost objective.

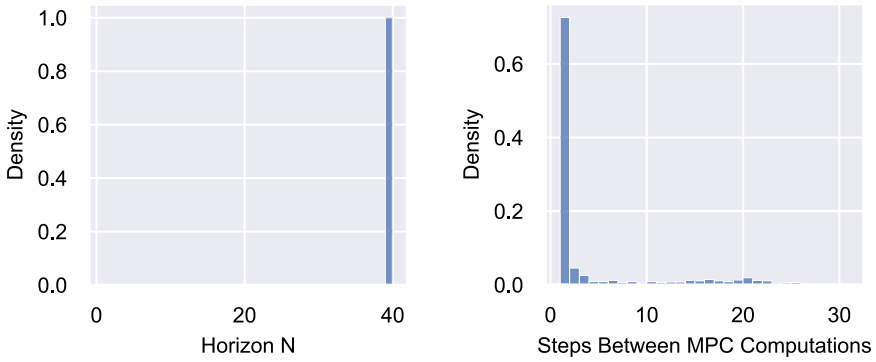


Fig. 3. Distribution of prediction horizons and steps between MPC computations selected by the best performing policy on the evaluation test-set.

Table 2
Ablation analysis: We take the best-performing policy and alter one aspect at a time, observing its effect on the cost.

Scenario	Change
Default LQR tuning	+1.95%
Max horizon 31	+13.41%
Recompute at the average frequency (every $t = 4$)	+35.45%

4.3. Inverted pendulum results and discussion

The results of the experiments are presented in Figs. 3 and 4, and Table 2. When reporting the cost of the tuned system, we average over the best-performing policy of each model, as well as over five different seeds for the stochastic recomputation policy. When describing learned behavior as in Fig. 3 however, we use the best performing policy and one specific seed. We summarize our main findings as follows:

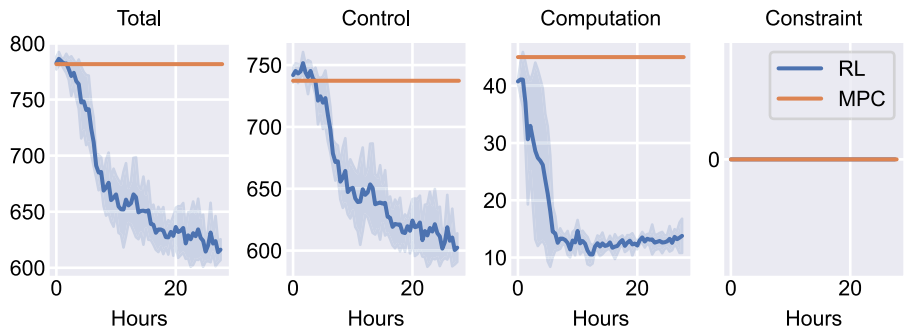
Learning improves both computational complexity and control performance. The RL framework we propose is able to improve upon the total objective by 21.5%, which interestingly is not only due to reductions in the computation term (70.7%) but also a sizable improvement on the control performance objective (18.4%). Fig. 4 shows that the recomputation meta-parameter is the most impactful, reaching a cost of around 700 when optimized by itself, which is about 13% higher than the converged value of the optimized complete policy. It also shows why we favor biasing towards higher computation, as while initializing the recomputation policy to compute the MPC with a 10% probability reaches the same asymptotic performance as the 90% initialization, it is the only policy we trained that intermittently violates the constraint. Because we initialize the learning problem at a best-effort of optimal tuning, the control performance improvements we observe are non-trivial to explain and arise due to complex interactions between multi-step adaptive-horizon MPC and the LQR control law. Fig. 3 shows the distribution of the prediction horizon and the steps between MPC computations chosen by the policies. The horizon policy has converged to only selecting the maximum horizon, while the MPC is mostly computed at every step (70% of steps) with some significantly longer streaks. Because of the finite-horizon nature of the OCP, the MPC will produce solutions of varying optimality based on the exact initial conditions it is computed from. The RL policy has seemingly learned to recognize a set of conditions for which the computed solutions are more optimal than neighboring conditions, and therefore not recomputing and employing a longer section of the more optimal input sequence will produce better control performance.

The control algorithm we propose adds some overhead compared to using MPC alone, i.e. evaluating the policies deciding if the MPC should be computed and with what horizon, and computing the LQR gain-matrix as necessary. The overhead is however small compared to the execution time of the MPC and the potential gains from computing it less often or with parameters that yield a simpler OCP. Therefore, our framework results in a 36% reduction in the total processing time of the control system compared to the best-performing MPC scheme. This frees up resources for other onboard tasks the controlled system might have or could be leveraged to increase the battery life of the system.

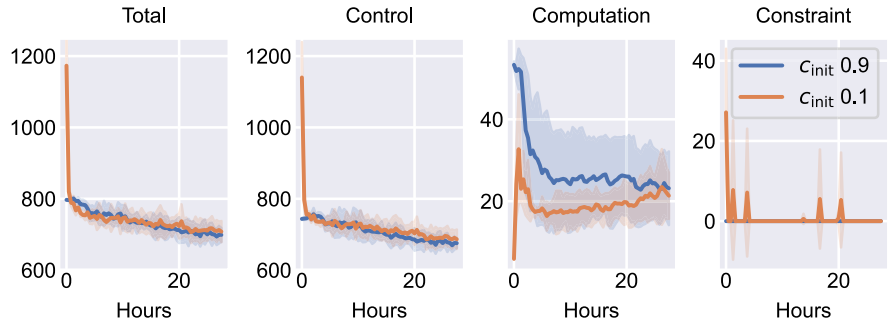
For this example it takes about 20 h of data of interaction with the system to reach convergence, corresponding to 700 thousand data samples. Fig. 4(a) shows that generally the performance monotonically improves with more data, we therefore do not view the data required as a major issue. One could run the data collection and parameter update calculation phase in a dedicated experiment, or since our tuning framework reduces the computational load of the control system itself, the tuning framework could in principle be run alongside the control system in normal operation to continuously improve performance with little impact on the total computational load of the embedded system. Further, parameter update calculations could even be offloaded to a remote computer as discussed in the introduction. Note that we did not spend significant time tuning the hyperparameters of the RL algorithm, and we favored consistency over a faster rate of improvement. Thus, the control algorithm's data requirement could potentially be improved with hyperparameter optimization of the RL algorithm.

One of the models we trained for the complete policy and one of the models for the isolated recomputation policy got stuck in the local minima of computing the MPC at every step, and since this did not give any interesting results (essentially maintaining the initial behavior and performance) we excluded them from Fig. 4 and the discussions, replacing them with new models with new seeds. Since policy gradient algorithms such as PPO are local search methods it is to be expected that it finds local minima, and random exploration can cause it to sometimes settle in sub-optimal local minima. This is also a question of how much data is used to generate the gradient.

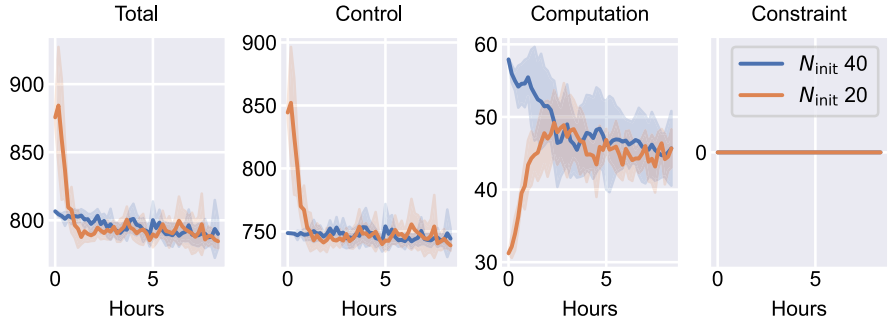
Multivariate optimization delivers additional improvements. These results support the conjecture stated in the introduction; by optimizing the different parameters of the control algorithm together, we are able to enhance the control algorithm in ways that we are not able to when optimizing the same parameters in isolation. The horizon policy tends to a mean horizon of 28–31 when optimized by itself (Fig. 4(c)), which is consistent with the best performing fixed horizon MPC as seen in Fig. 2. However, as Fig. 3 shows the complete RL policy favored higher horizons when optimizing all meta-parameters together in a multivariate fashion. As shown in Table 2 this improves performance significantly, where imposing a maximum horizon of 31 results in a 13.4% increase in cost. When solely tuning the LQR applied on an MPC computed on a fixed recomputation schedule, we were not able to achieve any consistent improvements. We hypothesize that this is due to the model mismatch (the MPC is overestimating the weight of the system by 36%) such that the computed LQR is not very well suited for the actual control problem (since it is computed from the erroneous MPC model), and therefore its gradients are flat and noisy. When combining LQR tuning with meta-parameter optimization we are able to



(a) Optimizing both meta-parameters jointly. The MPC baseline represents the MPC scheme tuned as a function of fixed recomputation schedule and horizon.



(b) Optimizing the recomputation meta-parameter in isolation.



(c) Optimizing the prediction horizon meta-parameter in isolation.

Fig. 4. Training process for the proposed learned control algorithm, and for each meta-parameter in isolation. In the isolated cases, we show that the tuning process is capable of recovering from sub-optimal initializations.

tune the LQR to achieve meaningful improvements, with the tuned LQR improving by 1.95% over the initialization described in Section 2.1.3. This might be due to the recomputation policy generating trajectories for the LQR that are similar, thus yielding more consistent gradients. The last entry in Table 2 shows a scenario where the same amount of computation is expended uniformly in time rather than dynamically allocated by the recomputation policy, resulting in a 35.45% increase in cost.

5. Conclusion

This paper has introduced a novel control algorithm based around an event-triggered MPC that is automatically tuned using RL with a configurable objective. The main contribution of the method is tuning the control algorithm's meta-parameters: the prediction horizon and the recomputation (triggering) problem. In a numerical experiment with the classic inverted pendulum control problem, we demonstrate that the control performance and computational complexity of the MPC can be simultaneously improved by computing the MPC less often! We speculate that this control performance improvement stems from learning from which initial conditions the MPC produces the most optimal solutions, and subsequently intelligently selecting when to compute the MPC and with what parameters in a state-dependent manner.

Seeing as MPC is increasingly being considered for applications with fast dynamics or limited computational power and energy resources, our framework could be an important tool in enabling such applications to harness the good control performance and constraint satisfaction abilities of the MPC.

We found that with a model mismatch, tuning the dual mode MPC and LQR control law was difficult. Future work could evaluate if state-dependent $\mathbf{Q}$ and $\mathbf{R}$ matrices alleviate this issue, which would also provide a gain-scheduling property. Whether any stability guarantees and control satisfaction properties can be provided given the learned meta-parameter-deciding policies should be investigated.

CRediT authorship contribution statement

Eivind Bøhn: Conceptualization, Methodology, Software, Validation, Writing – original draft, Visualization. **Sebastien Gros:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Signe Moe:** Writing – review & editing, Supervision. **Tor Arne Johansen:** Writing – review & editing, Methodology, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Code and data are available at the URL provided in the article.

Acknowledgments

The authors would like to thank Filippo Remonato for helpful discussion on the sensitivities of the LQR. This research was funded by the Research Council of Norway through the Centres of Excellence funding scheme, grant number 223254 NTNU AMOS, and through PhD Scholarships at SINTEF, grant number 272402.

References

Albin, T., Ritter, D., Abel, D., Liberda, N., Quirynen, R., Diehl, M., 2015. Nonlinear MPC for a two-stage turbocharged gasoline engine airpath. In: 2015 54th IEEE Conference on Decision and Control. CDC, IEEE, pp. 849–856.

Allgöwer, F., Badgwell, T.A., Qin, J.S., Rawlings, J.B., Wright, S.J., 1999. Nonlinear predictive control and moving horizon estimation — An introductory overview. In: Frank, P.M. (Ed.), Advances in Control. Springer London, London, pp. 391–449.

Amos, B., Jimenez, I., Sacks, J., Boots, B., Kolter, J.Z., 2018. Differentiable mpc for end-to-end planning and control. Adv. Neural Inf. Process. Syst. 31.

Aswani, A., Gonzalez, H., Sastry, S.S., Tomlin, C., 2013. Provably safe and robust learning-based model predictive control. Automatica 49 (5), 1216–1226.

Bansal, S., Calandra, R., Xiao, T., Levine, S., Tomlin, C.J., 2017. Goal-driven dynamics learning via Bayesian optimization. In: 2017 IEEE 56th Annual Conference on Decision and Control. CDC, IEEE, pp. 5168–5173.

Bemporad, A., Morari, M., 1999. Robust model predictive control: A survey. In: Robustness in Identification and Control. Springer, pp. 207–226.

Bemporad, A., Morari, M., Dua, V., Pistikopoulos, E.N., 2002. The explicit linear quadratic regulator for constrained systems. Automatica 38 (1), 3–20. http://dx.doi.org/10.1016/S0005-1098(01)00174-1.

Berglind, J.B., Gommans, T., Heemels, W., 2012. Self-triggered MPC for constrained linear systems and quadratic costs. IFAC Proc. Vol. 45 (17), 342–348. http://dx.doi.org/10.3182/20120823-5-NL-3013.00058, 4th IFAC Conference on Nonlinear Model Predictive Control.

Bertsekas, D.P., 1995. Dynamic Programming and Optimal Control, Vol. 1. Athena scientific Belmont, MA.

Bøhn, E., 2021. Project code-base and models. GitHub. https://github.com/eivindeb/rlmpcopt.

Bøhn, E., Gros, S., Moe, S., Johansen, T.A., 2021a. Optimization of the model predictive control update interval using reinforcement learning. IFAC-PapersOnLine 54 (14), 257–262. http://dx.doi.org/10.1016/j.ifacol.2021.10.362, 3rd IFAC Conference on Modelling, Identification and Control of Nonlinear Systems MICNON 2021.

Bøhn, E., Gros, S., Moe, S., Johansen, T.A., 2021b. Reinforcement learning of the prediction horizon in model predictive control. IFAC-PapersOnLine 54 (6), 314–320. http://dx.doi.org/10.1016/j.ifacol.2021.08.563, 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021.

Demirtas, H., 2017. On accurate and precise generation of generalized Poisson variates. Comm. Statist. Simulation Comput. 46 (1), 489–499. http://dx.doi.org/10.1080/03610918.2014.968725.

Edwards, W., Tang, G., Mamakoukas, G., Murphey, T., Hauser, K., 2021. Automatic tuning for data-driven model predictive control. In: International Conference on Robotics and Automation. ICRA.

Feng, L., Gutvik, C., Johansen, T., Sui, D., Brubakk, A., 2012. Approximate explicit nonlinear receding horizon control for decompression of divers. IEEE Trans. Control Syst. Technol. 20, 1275–1284. http://dx.doi.org/10.1109/TCST.2011.2162516.

Fisac, J.F., Akametalu, A.K., Zeilinger, M.N., Kaynama, S., Gillula, J., Tomlin, C.J., 2018. A general safety framework for learning-based control in uncertain robotic systems. IEEE Trans. Automat. Control 64 (7), 2737–2752.

Friedman, J., Hastie, T., Tibshirani, R., et al., 2001. The Elements of Statistical Learning. In: Springer Series in Statistics New York, vol. 1, no. 10.

Gardezi, M.S.M., Hasan, A., 2018. Machine learning based adaptive prediction horizon in finite control set model predictive control. IEEE Access 6, 32392–32400. http://dx.doi.org/10.1109/ACCESS.2018.2839519.

Goebel, G., Allgöwer, F., 2015. A simple semi-explicit MPC algorithm. IFAC-PapersOnLine 48 (23), 489–494. http://dx.doi.org/10.1016/j.ifacol.2015.11.326, 5th IFAC Conference on Nonlinear Model Predictive Control NMPC 2015.

Gondhalekar, R., Dassau, E., Doyle, III, F.J., 2015. Tackling problem nonlinearities & delays via asymmetric, state-dependent objective costs in MPC of an artificial pancreas. IFAC-PapersOnLine 48 (23), 154–159. http://dx.doi.org/10.1016/j.ifacol.2015.11.276, 5th IFAC Conference on Nonlinear Model Predictive Control NMPC 2015.

Gros, S., Quirynen, R., Diehl, M., 2012. Aircraft control based on fast non-linear MPC & multiple-shooting. In: 2012 IEEE 51st IEEE Conference on Decision and Control. CDC, IEEE, pp. 1142–1147.

Gros, S., Zanon, M., 2019. Data-driven economic NMPC using reinforcement learning. IEEE Trans. Automat. Control 65 (2), 636–648.

Gros, S., Zanon, M., 2021. Reinforcement learning based on MPC and the stochastic policy gradient method. In: 2021 American Control Conference. ACC, pp. 1947–1952. http://dx.doi.org/10.23919/ACC50511.2021.9482765.

Gros, S., Zanon, M., Bemporad, A., 2020. Safe reinforcement learning via projection on a safe set: how to achieve optimality? In: IFAC 2020.

Grüne, L., Pannek, J., 2011. Nonlinear Model Predictive Control. Springer-Verlag, London.

Johansen, T.A., 2017. Toward dependable embedded model predictive control. IEEE Syst. J. 11, 1208–1219.

Kalyanakrishnan, S., Aravindan, S., Bagdawat, V., Bhatt, V., Goka, H., Gupta, A., Krishna, K., Piratla, V., 2021. An analysis of frame-skipping in reinforcement learning. arXiv preprint arXiv:2102.03718.

Krener, A.J., 2018. Adaptive horizon model predictive control. IFAC-PapersOnLine 51 (13), 31–36. http://dx.doi.org/10.1016/j.ifacol.2018.07.250.

Lau, M., Yue, S., Ling, K., Maciejowski, J., 2015. A comparison of interior point and active set methods for FPGA implementation of model predictive control. In: Proc. European Control Conference.

Laub, A., 1979. A Schur method for solving algebraic Riccati equations. IEEE Trans. Automat. Control 24 (6), 913–921.

Li, H., Shi, Y., 2014. Event-triggered robust model predictive control of continuous-time nonlinear systems. Automatica 50 (5), 1507–1513. http://dx.doi.org/10.1016/j.automatica.2014.03.015.

Lowrey, K., Rajeswaran, A., Kakade, S., Todorov, E., Mordatch, I., 2018. Plan online, learn offline: Efficient learning and exploration via model-based control. arXiv preprint arXiv:1811.01848.

Makrygiorgos, G., Bonzanini, A.D., Miller, V., Mesbah, A., 2022. Performance-oriented model learning for control via multi-objective Bayesian optimization. Comput. Chem. Eng. 162, 107770.

Mayne, D., Rawlings, J., Rao, C., Scokaert, P., 2000. Constrained model predictive control: Stability and optimality. Automatica 36 (6), 789–814. http://dx.doi.org/10.1016/S0005-1098(99)00214-9.

Mayne, D.Q., Seron, M.M., Raković, S., 2005. Robust model predictive control of constrained linear systems with bounded disturbances. Automatica 41 (2), 219–224.

Mesbah, A., Wabersich, K.P., Schoellig, A.P., Zeilinger, M.N., Lucia, S., Badgwell, T.A., Paulson, J.A., 2022. Fusion of machine learning and MPC under uncertainty: What advances are on the horizon? In: 2022 American Control Conference. ACC, IEEE, pp. 342–357.

Michalska, H., Mayne, D.Q., 1993. Robust receding horizon control of constrained nonlinear systems. IEEE Trans. Automat. Control 38 (11), 1623–1633. http://dx.doi.org/10.1109/9.262032.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.

Nejatbakhsh Esfahani, H., Bahari Kordabad, A., Gros, S., 2021. Reinforcement learning based on MPC/MHE for unmodeled and partially observable dynamics. pp. 2121–2126. http://dx.doi.org/10.23919/ACC50511.2021.9483399.

Piga, D., Forgione, M., Formentin, S., Bemporad, A., 2019. Performance-oriented model learning for data-driven MPC design. IEEE Control Syst. Lett. 3 (3), 577–582. http://dx.doi.org/10.1109/LCSYS.2019.2913347.

Rao, C.V., Wright, S.J., Rawlings, J.B., 1998. Application of interior-point methods to model predictive control. J. Optim. Theory Appl. 99 (3), 723–757.

Rawlings, J.B., Mayne, D.Q., Diehl, M., 2017. Model Predictive Control: Theory, Computation, and Design, Vol. 2. Nob Hill Publishing Madison, WI.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Scokaert, P.O.M., Mayne, D.Q., 1998. Min-max feedback model predictive control for constrained linear systems. IEEE Trans. Automat. Control 43 (8), 1136–1142. http://dx.doi.org/10.1109/9.704989.

Scokaert, P.O.M., Mayne, D.Q., Rawlings, J.B., 1999. Suboptimal model predictive control (feasibility implies stability). IEEE Trans. Automat. Control 44 (3), 648–654. http://dx.doi.org/10.1109/9.751369.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA, http://dx.doi.org/10.5555/3312046.

Wang, W., Famoye, F., 1997. Modeling household fertility decisions with generalized Poisson regression. J. Popul. Econ. 10 (3), 273–283.

Yoo, J., Johansson, K.H., 2019. Event-triggered model predictive control with a statistical learning. IEEE Trans. Syst. Man Cybern. 1–11. http://dx.doi.org/10.1109/TSMC.2019.2916626.

Zhong, M., Johnson, M., Tassa, Y., Erez, T., Todorov, E., 2013. Value function approximation and model predictive control. In: 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. ADPRL, pp. 100–107. http://dx.doi.org/10.1109/ADPRL.2013.6614995.