Proceedings of Machine Learning Research vol vvv:1–32, 2024

MPC-Inspired Reinforcement Learning for Verifiable Model-Free Control

Yiwen Lu LUYW20@MAILS.TSINGHUA.EDU.CN
Zishuo Li LIZS19@MAILS.TSINGHUA.EDU.CN
Yihan Zhou ZHOUYH23@MAILS.TSINGHUA.EDU.CN
Na Li NALI@SEAS.HARVARD.EDU
Yilin Mo YLMO@TSINGHUA.EDU.CN

Abstract

In this paper, we introduce a new class of parameterized controllers, drawing inspiration from Model Predictive Control (MPC). The controller resembles a Quadratic Programming (QP) solver of a linear MPC problem, with the parameters of the controller being trained via Deep Reinforcement Learning (DRL) rather than derived from system models. This approach addresses the limitations of common controllers with Multi-Layer Perceptron (MLP) or other general neural network architecture used in DRL, in terms of verifiability and performance guarantees, and the learned controllers possess verifiable properties like persistent feasibility and asymptotic stability akin to MPC. On the other hand, numerical examples illustrate that the proposed controller empirically matches MPC and MLP controllers in terms of control performance and has superior robustness against modeling uncertainty and noises. Furthermore, the proposed controller is significantly more computationally efficient compared to MPC and requires fewer parameters to learn than MLP controllers. Real-world experiments on vehicle drift maneuvering task demonstrate the potential of these controllers for robotics and other demanding control tasks.

1. Introduction

Recent years have witnessed the development of Deep Reinforcement Learning (DRL) for control (Lillicrap et al., 2015; Duan et al., 2016; Haarnoja et al., 2018), with the locomotion of agile robots (Xie et al., 2018; Li et al., 2021; Margolis et al., 2022; Rudin et al., 2022) being a notable example. Many such applications use Multi-Layer Perceptron (MLP) as the entirety or a part of the control policy, which, despite their remarkable empirical performance, face limitations in terms of explainability (Agogino et al., 2019) and performance guarantees (Osinenko et al., 2022). Research efforts have been devoted to the stability verification of MLP controllers (Dai et al., 2021; Zhou et al., 2022), structured controller parameterizations (Srouji et al., 2018; Johannink et al., 2019; Sattar and Oymak, 2020; Ni et al., 2021), or a combination of both (Zinage and Bakolas, 2023), and the learning of explainable and verifiable controllers has remained an active topic.

On the other hand, Model Predictive Control (MPC) has been a prevalent choice for high performance controller, and its stability and safety has been well-studied (Morari and Lee, 1999; Qin and Badgwell, 2003; Schwenzer et al., 2021). Recently, there is a growing interest in augmenting MPC with learning, a large portion of which are focused on addressing the challenges in designing critical components of MPC like prediction models (Desaraju and Michael, 2016; Soloperto et al., 2018; Hewing et al., 2019), terminal costs and constraints (Brunner et al., 2015; Rosolia and Borrelli, 2017; Abdufattokhov et al., 2021), stage costs (Englert et al., 2017; Menner et al., 2019), or a combination of these components (Gros and Zanon, 2019); readers are referred to Hewing et al. (2020) for a comprehensive overview. Most of the methods follow a model-based framework – using data to estimate a model and then perform optimal control in a receding horizon fashion. These methods

still suffer various challenges, such as requiring intensive computation at each time horizon, and making myopic decisions that lead to infeasibility or inefficiency in the long run.

Motivated by the advantages of DRL and MPC, we propose an MPC-inspired yet model-free controller. Leveraging the fact that linear MPC solves a Quadratic Programming (QP) problem at each time step, we consider a parameterized class of controllers with QP structure similar to MPC. However, the key distinction lies in the approach to obtaining the QP problem parameters: instead of deriving them from a model, they are optimized via DRL. This approach ensures that the resulting controllers not only have theoretical guarantees akin to MPC, thanks to its QP structure, but also demonstrate competitive performance and computational efficiency when empirically compared to MPC and MLP controllers.

Contrasting with works from the learning community, such as Amos et al. (2018), which uses MPC as a module in a larger policy network, and Ha and Schmidhuber (2018); Hansen et al. (2023); LeCun (2022), which adapt MPC ideas by planning in latent spaces, our work retains the QP structure of linear MPC. While these learning-based approaches aim to be general and tackle more challenging tasks by integrating MPC concepts into comprehensive models, they often include black-box components without control-theoretic guarantees. In contrast, our approach specializes in control tasks, emphasizing performance guarantees and computational efficiency. However, empirical evidence on a real-world robotic system demonstrates that our controller may generalize beyond simple linear systems, as illustrated in an aggressive vehicle control setting.

Our Contribution. In this paper, we propose a new parameterized class of MPC-inspired controllers. Specifically, our controller resembles an unrolled QP solver, structured similarly to a Recurrent Neural Network (RNN), with its parameters learned rather than derived via a predictive model. To train the parameters of the controller, most of the existing DRL methods, such as PPO (Schulman et al., 2017), could be used. However, in contrast to most DRL-trained controllers, which often lack rigorous theoretical guarantees, our MPC-inspired controller is proven to enjoy verifiable properties like persistent feasibility and asymptotic stability. We also compare the proposed controller on benchmark tasks with other methods such as classical MPC and DRL-trained neural network controllers, showing that our proposed controller enjoys lighter computation and increased robustness. Lastly, though we only provide theoretical guarantees for controlling a linear system, the generalizability of the proposed constroller is empirically demonstrated via vehicle drift maneuvering, a challenging nonlinear robotics control task, indicating potential applications of our controller to real-world nonlinear robotic systems.

2. Problem Formulation and Preliminaries

Notations. Subscripts denote the time index, e.g., x_k stands for the system state at step k, and $x_{0:k}$ means the sequence $x_0, x_1, \ldots, x_k$. Superscripts denote the iteration index in an iterative algorithm, e.g. y^i stands for the variable y at the i-th iteration. Bracketed subscripts denote slicing operation on a vector, e.g., $v_{[1]}$ denotes the first element of the vector v, and $v_{[1:n]}$ denotes its first n elements. The set of positive definite $n \times n$ matrices is denoted as $\mathbb{S}^n_{++}$, and the nonnegative orthant of $\mathbb{R}^n$ is denoted as $\mathbb{R}^n_+$. The Kronecker product of two matrices A and B is denoted as $A \otimes B$. The block diagonal matrix with diagonal blocks $A_1, \ldots, A_n$ is denoted as $\mathrm{diag}(A_1, \ldots, A_n)$. The projection operator to a convex set C is denoted as $\Pi_C(\cdot)$.

2.1. Problem Formulation

In this paper, we consider the discrete-time infinite-horizon constrained linear-quadratic optimal control problem, formulated as follows:

Problem 1 (Infinite-horizon constrained linear-quadratic optimal control)

$$\underset{u_{0:\infty}}{\textit{minimize}} \quad \limsup_{N\to\infty} \frac{1}{N}\sum_{k=0}^{N-1}(x_{k+1}-r)^\top Q(x_{k+1}-r)+u_k^\top R u_k, \tag{1a}$$

$$\textit{subject to} \quad x_{k+1}=Ax_k+Bu_k, \tag{1b}$$

$$u_{\min}\le u_k\le u_{\max}, x_{\min}\le x_{k+1}\le x_{\max}, \tag{1c}$$

where $x_k\in\mathbb{R}^{n_{sys}}$ are state vectors, $u_k\in\mathbb{R}^{m_{sys}}$ are control input vectors, $r\in\mathbb{R}^{n_{sys}}$ is the reference signal, $A\in\mathbb{R}^{n_{sys}\times n_{sys}}$ and $B\in\mathbb{R}^{n_{sys}\times m_{sys}}$ are the system and input matrices, $Q\in\mathbb{S}_{++}^{n_{sys}}$ and $R\in\mathbb{S}_{++}^{m_{sys}}$ are the stage cost matrices, and $u_{\min},u_{\max}\in\mathbb{R}^{m_{sys}}$ and $x_{\min},x_{\max}\in\mathbb{R}^{n_{sys}}$ are bounds on control input and state respectively. It is assumed without loss of generality that (A,B) is controllable.

2.2. Linear MPC and its QP Representation

Problem 1 is typically computationally intractable due to infinite planning horizons and constraints. A commonly adopted approximation is to truncate it to finite horizon N, and solve the problem formulated in Problem 2 at each time step, with x_0 being the current state. The first control input u_0^* from the optimal solution is applied to the system in a receding horizon fashion.

Problem 2 (Linear MPC)

$$\underset{x_{1:N},u_{0:N-1}}{\textit{minimize}} \quad \sum_{k=0}^{N-1}(x_{k+1}-r)^\top Q(x_{k+1}-r)+u_k^\top R u_k, \tag{2a}$$

$$\textit{subject to} \quad x_{k+1}=Ax_k+Bu_k, \quad k=0,\cdots,N-1, \tag{2b}$$

$$u_{\min}\le u_k\le u_{\max}, x_{\min}\le x_{k+1}\le x_{\max}, \quad k=0,\cdots,N-1. \tag{2c}$$

The above MPC problem can be cast into a Quadratic Programming (QP) problem in the following standard form[1]:

Problem 3 (Standard-form QP)

$$\underset{y}{\textit{minimize}} \quad \frac{1}{2}y^\top Py+q^\top y, \quad \textit{subject to} \quad Hy+b\ge 0, \tag{3}$$

where $y\in\mathbb{R}^{n_{qp}}$, $P\in\mathbb{S}_{++}^{n_{qp}}$, $q\in\mathbb{R}^{n_{qp}}$, $H\in\mathbb{R}^{m_{qp}\times n_{qp}}$, and $b\in\mathbb{R}^{m_{qp}}$.

The translation from Problem 2 to Problem 3 can be performed by using the control sequence $y=\begin{bmatrix}u_0^\top & \cdots & u_{N-1}^\top\end{bmatrix}^\top$ as the decision variable, and eliminating the equality constraints (2b) by representing the trajectory $x_{1:N}$ using y. The resulting QP problem size and parameters are:

$$n_{qp}=Nm_{sys}, \quad m_{qp}=2N(m_{sys}+n_{sys}), \tag{4}$$

$$P=\boldsymbol{B}^\top\boldsymbol{Q}\boldsymbol{B}+\boldsymbol{R}, \quad q=2\boldsymbol{B}^\top\boldsymbol{Q}(\boldsymbol{A}x_0-\boldsymbol{r}), \quad H=-\boldsymbol{C}\boldsymbol{B}-\boldsymbol{D}, \quad b=\boldsymbol{e}-\boldsymbol{C}\boldsymbol{A}x_0, \tag{5}$$

where

$$\boldsymbol{A}=\begin{bmatrix}A\\A^2\\\vdots\\A^N\end{bmatrix}, \boldsymbol{B}=\begin{bmatrix}B & & & \\ AB & B & & \\ \vdots & & \ddots & \\ A^{N-1}B & \cdots & & B\end{bmatrix}, \quad \begin{aligned}&\boldsymbol{C}=I_N\otimes\mathrm{diag}(I_{n_{sys}},-I_{n_{sys}},\mathbf{0}_{2m_{sys}\times 2m_{sys}}),\\&\boldsymbol{D}=I_N\otimes\mathrm{diag}(\mathbf{0}_{2n_{sys}\times 2n_{sys}},I_{m_{sys}},-I_{m_{sys}}),\\&\boldsymbol{e}=I_N\otimes[x_{\max}^\top\ -x_{\min}^\top\ u_{\max}^\top\ -u_{\min}^\top]^\top,\\&\boldsymbol{r}=I_N\otimes r,\boldsymbol{Q}=I_N\otimes Q,\boldsymbol{R}=I_N\otimes R.\end{aligned} \tag{6}$$

1. Although a linear MPC without terminal costs or constraints is presented here for simplicity, one can derive a similar QP formulation for linear MPC with quadratic terminal cost and affine terminal constraint.

2.3. Algorithm for Solving QPs

A family of efficient methods for solving QPs is operator splitting algorithms (Ryu and Yin, 2022), which are adopted by existing solvers such as OSQP (Stellato et al., 2020). An iteration of an operator splitting algorithm for solving QPs can generally be represented as a combination of affine transformations and projections on the variable. For example, we derive a variant of the Primal-Dual Hybrid Gradient (PDHG) (Chambolle and Pock, 2011) algorithm, whose iteration can be expressed in the succinct form shown as follows: [2]

$$z^{i+1} = \Pi_{\mathbb{R}_+^{m_{qp}}} \left((I - 2\alpha F) z^i + \alpha (I - 2F) \lambda^i) - 2\alpha \mu \right), \quad \lambda^{i+1} = F(z^i + \lambda^i) + \mu, \tag{7}$$

where $z = Hy + b \in \mathbb{R}^{m_{qp}}$ is the primal variable of an equivalent form of the original problem (3), $\lambda \in \mathbb{R}^{m_{qp}}$ is a dual variable introduced by the same equivalent form, $\alpha > 0$ is the step size, and the parameters in the iteration are:

$$F = (I + HP^{-1}H^\top)^{-1}, \quad \mu = F(HP^{-1}q - b). \tag{8}$$

Once one obtains an approximate solution z^i, the original variable can be recovered from the equality-constrained QP problem $y^i \in \arg\min\{\frac{1}{2} y^\top P y + q^\top y \mid Hy + b = z^i\}$, where y^i can be explicitly represented as:

$$y^i = -P^{-1}q + P^{-1}H^\top (HP^{-1}H^\top)^\dagger (z^i - b + HP^{-1}q). \tag{9}$$

Theorem 1 *If $0 < \alpha < 1$ and the problem (3) is feasible, then the iterations (7) yields $y^i \to y^*$, where y^i is in (9) and y^* is the optimal solution of the original problem. Furthermore, the suboptimality gap satisfies:*

$$p^i - p^* \le \|\lambda^i\|_2 \|r^i_{prim}\|_2 + \|y^i - y^*\|_2 \|r^i_{dual}\|_2, \tag{10}$$

where p^i, p^* are the primal value at iteration i and the optimal primal value respectively, and r^i_{prim}, r^i_{dual} are the primal and dual residuals defined as follows:

$$r^i_{prim} = Hy^i + b - z^i, r^i_{dual} = Py^i + q + H^\top \lambda^i. \tag{11}$$

Proof Results similar to Theorem 1 has been derived in (Chambolle and Pock, 2011)(Boyd et al., 2011). Therefore, the full proof is presented in the extended version of the paper (Lu et al., 2023a, Appendix A) due to space limit. ■

The iteration (7) on the primal-dual variable pair (z, λ) can be implemented by interleaving an affine transformationwhose parameters (F, μ) depend on the problem parameters (P, q, H, b), and a projection of the z-part onto the positive orthant, which is equivalent to ReLU activation in neural networks. Therefore, the sequence of iterations for solving a QP problem resembles a single-layer Recurrent Neural Network (RNN) with weights dependent on the QP parameters (P, q, H, b), followed by ReLU activation. This resemblance facilitates the end-to-end policy gradient-based reinforcement learning of the QP parameters, as is discussed in Section 3.

2. The form (7) is slightly different from the original PDHG iteration (Ryu and Yin, 2022, p. 75) in that the primal and dual variables are updated synchronously, making it conceptually more straightforward to draw similarity between solver iterations and neural networks; see (Lu et al., 2023a, Appendix A) for the details.

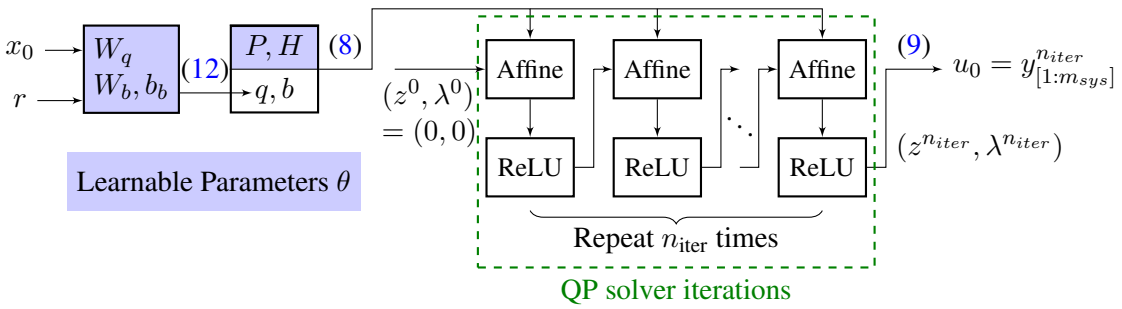


Figure 1: Proposed control policy architecture. The controller solves a QP problem in form (3), whose parameters P, H are shared across all initial state and reference (x_0, r), while q, b depend affinely on (x_0, r) with weights W_q, W_b and bias b_b (see (12)). An approximate solution to the QP problem, $y^{n_{iter}}$, is obtained by running a n_{iter} QP solver iterations (7) followed by a transform (9), whose first m_{sys} dimensions are used as the current control input u_0.

3. Learning Model-Free QP Controllers

This section introduces a reinforcement learning framework to tune the parameters of QP problem (3), instead of deriving them using the predictive models via conventional MPC. This could be beneficial to combat the short-sightedness (Erez et al., 2012) and the lack of robustness (Forbes et al., 2015), which may occur in MPC.

The policy architecture facilitating this learning process is shown in Figure 1, where the control policy, i.e., the mapping from x_0 and r to the control action u_0, is represented as a fixed number of PDHG iteration for solving a QP problem, or equivalently, an unrolled RNN, the parameters of which can then be trained using most of the existing policy-gradient based or actor-critic DRL method.

Here we highlight several critical design components of the policy architecture. The first two are regarding the parameterization of the policy, i.e., *what* to learn:

• **State-independent matrices P, H:** Note from (5) that for the MPC controller, the matrices P, H holds the same across different initial and reference state (x_0, r)'s. Motivated by this fact, the matrices P, H are state-independent in the proposed policy architecture, i.e., only one matrix P and one matrix H need to be learned for a specific system. Additionally, to ensure the positive definiteness of P, we use the factor L_P in the Cholesky decomposition $P = L_P L_P^\top$ instead of the matrix P itself as the learnable parameter, and force its diagonal elements to be positive via a softplus activation (Zheng et al., 2015), a commonly applied trick for learning positive definite matrices (Haarnoja et al., 2016; Lutter et al., 2019).

• **Affine transformations yielding vectors q, b:** Continuing with the inspirations from MPC (5), we restrict the vectors q to depend linearly on the current state x_0 and the reference state r, and the vector b to depend affinely on x_0:

$$q(x_0, r; W_q) = W_q[x_0^\top \; r^\top]^\top, \quad b(x_0, r; W_b, b_b) = W_b x_0 + b_b, \tag{12}$$

where W_q, W_b (resp. b_b) are learnable matrices (resp. vector) of proper dimensions.

The above-described parameterization strategy ensures that when the chosen problem dimensions n_{qp}, m_{qp} match the dimensions of the QP translated from MPC, then the MPC policy is within the family of parameterized policies defined by the proposed architecture. In other words, the proposed controller can be viewed as a generalization of MPC. While the state-independence

and state-affineness constraints in the parameterization limit the range of policies relative to those in MLP or MPC-like policies with generic function approximator components (Amos et al., 2018) , this reduced complexity is beneficial for deriving provable theoretical guarantees, as is discussed in Section 4. On the other hand, both numerical examples and real-world experiments show that the empirical performance of the proposed controller matches that of an MPC or MLP controller, and thus is not hindered by this restriction.

Another two design components determine *how* the parameters are learned:

• **Unrolling with a fixed number of iterations:** To solve the QP problem and differentiate the solution with respect to the problem parameters, we deploy a fixed number n_{iter} of QP solver iterations described in Section 2.3, and differentiate through the computational path of these iterations, a practice known as unrolling (Monga et al., 2021). Unlike implicit differentiation methods (Amos and Kolter, 2017; Amos et al., 2018; Agrawal et al., 2019), which differentiate through the optimality condition and hence requires the forward pass of the solver to reach the stationary point, our method directly differentiates the solution after n_{iter} iterations, and can obtain a correct gradient even if the stationary point is not reached within these iterations. According to our empirical results, a small number of iterations would suffice for good control performance (e.g., $n_{iter} = 10$), which mitigates the computational burden of the unrolling process. Intuitively, the sufficiency of a small n_{iter} can be accredited to the model-free nature of the proposed method, which, by discarding the restrictions imposed by model-based prediction (see (6)), gains the flexibility to learn a QP problem that not only optimizes the controller performance, but also is easy to solve.

• **Reinforcement learning with residual minimization:** The control policy described above, parameterized by $\theta = (L_P, H, W_q, W_b, b_b)$, can serve as a drop-in replacement for standard policy networks, and be optimized using various off-the-shelf policy-based or actor-critic RL algorithms, such as PPO (Schulman et al., 2017), SAC (Haarnoja et al., 2018) and DDPG (Lillicrap et al., 2015). However, apart from the standard RL loss, we also include a regularization term for minimizing the residuals given by the QP solver embedded in the policy. Given a dataset $\mathcal{D}$ of transition samples, it is defined as follows:

$$\ell_{res}(\theta; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{k=1}^{|\mathcal{D}|} \|H y_k^{n_{iter}} + b_k - z_k^{n_{iter}}\|_2^2 + \|P y_k^{n_{iter}} + q_k + H^\top \lambda_k^{n_{iter}}\|_2^2, \tag{13}$$

which, motivated by the result stated in Theorem 1 that small residuals are indicative of near-optimality, encourages the learned QP problems to be easy to solve. From above, the procedure of policy learning using an RL algorithm is shown in Algorithm 1.

Algorithm 1: Framework of Learning of QP Controllers

Input: Simulation environment Env with nominal dynamics (1b), RL algorithm RL, policy architecture π_θ shown in Fig. 1, regularization coefficient ρ_{res}

Output: Optimized policy parameters $\theta = (L_P, H, W_q, W_b, b_b)$

for *epoch* $= 1, 2, \ldots$ **do**

- Interact with Env using current policy π_θ to collect a dataset $\mathcal{D}$
- Compute RL loss, denoted by $\ell_{RL}(\theta; \mathcal{D})$
- Compute residual loss $\ell_{res}(\theta; \mathcal{D})$ using (13)
- Update θ according to the loss $\ell_{RL}(\theta; \mathcal{D}) + \rho_{res}\ell_{res}(\theta; \mathcal{D})$

As an additional note, the learned QP problem parameterized by (P, H, q, b) can be enforced to be feasible, which facilitates the theoretical analysis of the learned controller. Readers are referred to (Lu et al., 2023a, Appendix B) for the details.

4. Performance Guarantees of Learned QP Controller

In this section, we propose a method for establishing performance guarantees of a learned QP controller with the architecture described in Section 3. We provide sufficient conditions for persistent feasibility and asymptotic stability of the closed-loop system under a QP controller, which parallel the theoretical guarantees for linear MPC (Borrelli et al., 2017). For simplicity, we consider the stabilization around the origin, i.e., $r = 0$, but the method of analysis can be extended to the general case. Additionally, we assume throughout the section that the optimal solution of the learned QP problem is attained, which can be ensured by allowing the QP solver to run sufficient iterations until convergence when deploying.

Denote the property under consideration as $\mathcal{P}$. Suppose that a certificate to $\mathcal{P}$, given the initial state x_0 is in a polytopic set $\mathcal{X}_0$, can be written in the following form:

$$\min_{x_0 \in \mathcal{X}_0, u_0, \nu} \{f(x_0, u_0, \nu) | g(x_0, u_0, \nu) \leq 0, u_0 = \pi_\theta(x_0)\} \geq 0 \Rightarrow \mathcal{P} \text{ holds when } x_0 \in \mathcal{X}_0, \quad (14)$$

where π_θ denotes the θ-parameterized control policy described in Section 3, ν is an auxiliary variable, and f, g are quadratic (possibly nonconvex) functions. The optimization problem in the LHS of (14) can be expressed as a bilevel problem by explicitly expanding the control policy π_θ as:

$$\begin{aligned} &\pi_\theta(x_0) = y^*_{[1:m_{sys}]}, y^* \in \arg\min \left\{ (1/2) y^\top P y + q^\top y \mid Hy + b \geq 0 \right\}, \\ &\text{where } q = W_q x_0, b = W_b x_0 + b_b. \end{aligned} \quad (15)$$

Replacing the inner-level problem in (15) by its KKT condition, the verification problem in (14) can be cast into a nonconvex Quadratically Constrained Quadratic Program (QCQP) with variables x_0, ν, y, μ. Various computationally tractable methods for lower bounding the optimal value of a QCQP are available, such as Lagrangian relaxation (d'Aspremont and Boyd, 2003) and the method of moments (Lasserre, 2001), and once a nonnegative lower bound is obtained, the property $\mathcal{P}$ is verified.

Verification of persistent feasibility and asymptotic stability both fall into the framework described above. The conclusions are stated as follows:

Theorem 2 (Certificate for Persistent Feasibility) *The control policy (15) if persistently feasible (i.e., gives a valid control input that keeps the next state inside the bounds at every step) for all initial states $x_0 \in \mathcal{X}_0 = \{x | Gx \leq c\}$, if the optimal value of the following nonconvex QCQP is nonnegative:*

$$\begin{aligned} \underset{x_0, \nu, y, \mu}{\textit{minimize}} \quad & -\nu^\top (G(Ax_0 + By_{[1:m_{sys}]}) - c), \\ \textit{subject to} \quad & Gx_0 \leq c, \nu \geq 0, \mathbf{1}^\top \nu = 1, \\ & Py + W_q x_0 - H^\top \mu = 0, Hy + W_b x_0 + b_b \geq 0, \mu \geq 0, \mu^\top (Hy + W_b x_0 + b_b) = 0. \end{aligned}$$

To certify asymptotic stability, we consider the Lyapunov function of a stabilizing baseline MPC, and attempt to show that the Lyapunov function decreases along all trajectories even if the learned QP controller is deployed instead of the baseline MPC. A similar technique has been applied to the stability analysis of approximate MPC (Schwan et al., 2023). To formalize this idea, we define

the following notations: $l(x,u) = x^\top Qx + u^\top Ru$ is the stage cost; the baseline MPC policy has horizon N, terminal constraint $x_N \in \mathcal{X}_f$ and terminal cost $V_f(x_N)$; the function $J(x_0, u_{0:N-1}) = \sum_{k=0}^{N-1} l(x_k, u_k) + V_N(x_N)$, where $x_{k+1} = Ax_k + Bu_k$, is the objective function of the baseline MPC. To ensure that the baseline MPC is stabilizing as long as it is feasible, one can choose $\mathcal{X}_f$ to be an invariant set under a stabilizing linear feedback controller $u = Kx$, and $V_f(x)$ to be the cost-to-go under $u = Kx$. Based on these notations, a certificate for asymptotic stability can be stated as follows:

Theorem 3 (Certificate for Asymptotic Stability) *Let $\mathcal{X}_0 = \{x | Gx \leq c\}$ be a set where the baseline MPC is well-defined and the policy (15) is persistently feasible. The closed-loop system under (15) is asymptotically stable on $\mathcal{X}_0$, if $b_b \geq 0$, and there exists $\epsilon > 0$ and $N \in \mathbb{N}^*$, such that the optimal value of the following problem is nonnegative:*

$$\begin{aligned} \underset{x_0, \bar{u}_{0:N}, y, \mu}{\textit{minimize}} \quad & J(x_0, \bar{u}_{0:N}) + l(x_0, y_{[1:m_{sys}]}) - J(x_0, (y_{[1:m_{sys}]}, \bar{u}_{1:N})) - \epsilon \|x_0\|^2, \\ \textit{subject to} \quad & Gx_0 \leq c, x_{N+1}(Ax_0 + By_{[1:m_{sys}]}, \bar{u}_{1:N}) \in \mathcal{X}_f, \\ & Py + W_q x_0 - H^\top \mu = 0, Hy + W_b x_0 + b_b \geq 0, \mu \geq 0, \mu^\top (Hy + W_b x_0 + b_b) = 0. \end{aligned}$$

Proofs of Theorems 2 and 3, as well as numerical examples showcasing the verification of a learned controller on a double integrator system, are provided in (Lu et al., 2023a, Appendix D).

5. Benchmarking Results

In our empirical evaluations, we aim to answer the following questions:

- How does the learned QP controller compare with common baselines (MPC, RL-trained MLP) on nominal linear systems?
- Can the learned QP controller handle modeling inaccuracies and disturbances?
- Does the method generalize to real-world robot systems with modeling inaccuracy and nonlinearity?

We only briefly describe the experimental setup and typical results in this section, with complete details on systems, setup, hyperparameters, baseline definitions, and additional results in (Lu et al., 2023a, Appendix E). Code is available at `https://github.com/yiwenlu66/learning-qp`.

5.1. Results on Nominal Systems

We compare the Learned QP (LQP) controller with MPC and MLP baselines on benchmark systems like the quadruple tank (Johansson, 2000) and cartpole (Geva and Sitte, 1993), generating random initial states and references across 10^4 trials. For MPC, we evaluate variants with and without manually tuned terminal costs over short (2 steps) and long (16 steps) horizons, all implemented using OSQP (Stellato et al., 2020), a solver known for its efficiency in MPC applications (Forgione et al., 2020), with default solver configurations. Both LQP and MLP are trained using PPO, maintaining consistent reward definitions and RL hyperparameters. We incrementally increase the MLP size until further increases yield negligible performance improvements, selecting this size for comparison. The LQP is assessed in both small ($n_{qp} = 4, m_{qp} = 24$) and large ($n_{qp} = 16, m_{qp} = 96$) configurations, approximately aligning with the QP problem sizes from short- and long-horizon MPC. All training (including simulation and policy update) are performed on a single NVIDIA RTX 4090 GPU, with the small and large configurations taking 1.2 hours and 2.7 hours respectively.

The results of the benchmarking experiments are summarized in Table 1. In terms of control performance, LQP demonstrates comparable effectiveness to both MPC and MLP baselines. A benefit of LQP is its independence from manual tuning of the terminal cost, which can be important

Table 1: Performance comparison on benchmark systems.

Method \ Metrics	Quadruple Tank					Cartpole Balancing				
	Fail%	Cost	P-Cost	FLOPs	#Params	Fail%	Cost	P-Cost	FLOPs	#Params
MPC(2)	16.59	236.1	275.7	$95K_{+1.2M}$	-	100.0	1.36	129	$67K_{+814K}$	-
MPC(16)	4.27	228.3	237.2	$22M_{+52M}$	-	46.86	0.34	8.39	$3.9M_{+47M}$	-
MPC-T(2)	4.23	239.6	248.5	$470K_{+779K}$	-	100.0	1.41	122	$89K_{+792K}$	-
MPC-T(16)	3.22	**224.8**	231.5	$26M_{+49M}$	-	4.74	**0.30**	**0.79**	$51M_{+50M}$	-
RL-MLP	**0.03**	266.7	266.7	23K	11K	**3.23**	0.57	0.91	87K	43K
LQP(4, 24)	0.18	272.5	272.8	**14K**	**0.3K**	3.49	0.76	1.12	**14K**	**0.2K**
LQP(16, 96)	0.13	227.3	**227.6**	208K	2.6K	4.11	0.44	0.87	208K	2.2K

Methods: MPC(N) = MPC (Problem 2) with horizon N without terminal cost; MPC-T(N) = MPC with horizon N and manually tuned terminal cost; RL-MLP = reinforcement learning controller with MLP policy; LQP(n_{qp}, m_{qp}) = proposed learned QP controller with problem dimensions (n_{qp}, m_{qp}). Metrics: Fail% = percentage of early-terminated trials due to constraint violation; Cost = average LQ cost until termination; P-Cost = average cost with penalty for constraint violation; FLOPs = floating point operations per control step (reported as $\text{median}_{+(\text{max}-\text{median})}$ for variable data); #Params = number of learnable policy parameters. Best is highlighted in **bold**, and second best is underlined.

for MPC methods. Regarding computational efficiency, LQP stands out for its minimal demand for achieving similar control performance. This efficiency stems from LQP's fixed number of unrolled QP solver iterations. While MPC's computation cost varies based on implementation, the light computation of LQP is still noteworthy, especially in scenarios with tight computational limits. For example, the LQP(4, 24) configuration, despite having lowest FLOPs among all methods, still manages acceptable control performance. Finally, in terms of the number of learnable policy parameters, LQP requires substantially fewer than the RL-MLP. This hints at LQP's suitability for memory-constrained embedded systems and applicability to online few-shot learning.

Results on additional systems, including a numerical example of verifying the stability of the learned controllers, are deferred to the supplementary materials due to space limit (Lu et al., 2023a).

5.2. Validation of Robustness

This subsection is concerned with the robustness of the learned QP controller against modeling inaccuracies and disturbances. Instead of the nominal dynamics (1b), we now consider the following perturbed dynamics:

$$x_{k+1} = (A + \Delta A)x_k + (B + \Delta B)u_k + w_k,$$

where $\Delta A, \Delta B$ are parametric uncertainties, and w_k is a disturbance. LQP and MLP are trained using domain randomization (Tobin et al., 2017; Mehta et al., 2020), where the simulator randomly sample these uncertain components during training. Robust MPC baselines, including tube MPC (Mayne et al., 2005) and scenario MPC (Bernardini and Bemporad, 2009) implemented by the do-mpc toolbox (Fiedler et al., 2023), are included for comparison.

Table 2: Performance comparison on quadruple tank system with process noise and parametric uncertainties.

Method	Fail%	Cost	P-Cost	Time(s)	#Params
MPC-T(16)	82.6	**216.8**	713.4	$0.25_{+0.56}$	-
Tube	81.9	233.3	597.9	2.22_{+44}	-
Scenario	16.4	236.9	273.2	5.21_{+18}	-
RL-MLP	**1.3**	238.9	**241.5**	1×10^{-3}	43K
LQP(4, 24)	1.5	256.7	261.8	$\mathbf{2\times10^{-4}}$	**0.3K**
LQP(16, 96)	1.4	240.6	243.4	2×10^{-3}	2.7K

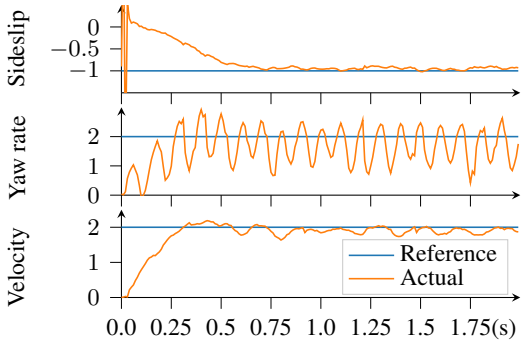
Notations are similar to those in the caption of Table 1. Computation time instead of FLOPs per control step is used as the metric for computational efficiency since it is difficult to obtain the exact FLOPs from the robust MPC baselines.

The results in Table 2 highlight LQP's robustness, as it achieves the success rate and constraint-violation-penalized cost comparable to MLP. Also, it requires significantly less online computation

compared to robust MPC methods, benefitting from domain randomization known for its effectiveness in empirical RL and robotics (Loquercio et al., 2019; Margolis et al., 2022).

5.3. Application Example on a Real-World System: Vehicle Drift Maneuvering

LQP is also evaluated on a challenging robotics control task, namely, the drift maneuvering of a 1/10 scale RC car, similar to the problem studied in Yang et al. (2022); Domberg et al. (2022); Lu et al. (2023b). The objective is to track the yaw rate, side slip angle, and velocity references, such that the car enters and maintains a drifting state. Despite the high nonlinearity of the system, the proposed controller formally introduced on linear systems successfully generalizes to this task. As shown in Fig. 2, the learned QP controller can track the references and maintain the drifting state, performing similarly to previous RL-trained MLP methods on this task (Domberg et al., 2022).



(*a*) Tracking performance

(*b*) Blended frames of experiment video

Figure 2: Result of deploying learned QP controller to the vehicle drift maneuvering task. Video available at: https://youtu.be/-XYtl2b4OVc.

6. Conclusion

This work presents a novel class of QP controllers inspired by MPC. The proposed controllers not only retain the theoretical guarantees akin to MPC, but also exhibit desirable empirical performance and computational efficiency. Benchmarks including applications in real-world scenarios like vehicle drift maneuvering, further validate the effectiveness and robustness of our approach.

Despite the promising results, several challenges remain: (i) the empirical results are not exhaustive nor conclusive, and further evidence is required to understand LQP's benefits and limitations in practice; (ii) the stability verification (Theorem 3) still relies on the Lyapunov function of MPC, urging the simultaneous learning of policy and certificate similar to Chang et al. (2019); (iii) the performance guarantees assume that the optimal QP solution is attained, a restriction that can potentially be lifted using techniques from Wu et al. (2022).

Additionally, the unrolled QP solver is structured similarly to a deep neural network, indicating the suitability of the proposed policy architecture as a drop-in replacement for standard policy networks in RL. This opens up possibilities for combining the architecture with various RL methods, such as meta-learning (Finn et al., 2017) and safety-constrained RL (Achiam et al., 2017; Yu et al., 2022), which are left for future investigation.

References

Shokhjakon Abdufattokhov, Mario Zanon, and Alberto Bemporad. Learning convex terminal costs for complexity reduction in mpc. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2163–2168. IEEE, 2021.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pages 22–31. PMLR, 2017.

Adrian Agogino, Ritchie Lee, and Dimitra Giannakopoulou. Challenges of explaining control. In *2nd ICAPS Workshop on Explainable Planning (XAIP'19)*, 2019.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

Mosek ApS. Mosek optimizer api for python. *Version*, 9(17):6–4, 2022.

MOSEK ApS. Mosek optimizer api for julia. 2023.

Daniele Bernardini and Alberto Bemporad. Scenario-based model predictive control of stochastic constrained linear systems. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6333–6338. IEEE, 2009.

Lorenz T Biegler and Victor M Zavala. Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3):575–582, 2009.

Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Florian D Brunner, Mircea Lazar, and Frank Allgöwer. Stabilizing model predictive control: On the enlargement of the terminal set. *International Journal of Robust and Nonlinear Control*, 25(15): 2646–2670, 2015.

Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40:120–145, 2011.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

Hongkai Dai, Benoit Landry, Lujie Yang, Marco Pavone, and Russ Tedrake. Lyapunov-stable neural-network control. *arXiv preprint arXiv:2109.14152*, 2021.

Alexandre d'Aspremont and Stephen Boyd. Relaxations and randomized methods for nonconvex qcqps. *EE392o Class Notes, Stanford University*, 1:1–16, 2003.

Vishnu R Desaraju and Nathan Michael. Experience-driven predictive control. *Robot Learning and Planning (RLP 2016)*, page 29, 2016.

Martin Doff-Sotta and Mark Cannon. Difference of convex functions in robust tube nonlinear mpc. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 3044–3050. IEEE, 2022.

Fabian Domberg, Carlos Castelar Wembers, Hiren Patel, and Georg Schildbach. Deep drifting: Autonomous drifting of arbitrary trajectories using deep reinforcement learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7753–7759. IEEE, 2022.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

Peter Englert, Ngo Anh Vien, and Marc Toussaint. Inverse kkt: Learning cost functions of manipulation tasks from demonstrations. *The International Journal of Robotics Research*, 36(13-14): 1474–1488, 2017.

Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. *Robotics: Science and systems VII*, page 73, 2012.

Felix Fiedler, Benjamin Karg, Lukas Lüken, Dean Brandner, Moritz Heinlein, Felix Brabender, and Sergio Lucia. do-mpc: Towards fair nonlinear and robust model predictive control. *Control Engineering Practice*, 140:105676, 2023.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

Michael G Forbes, Rohit S Patwardhan, Hamza Hamadah, and R Bhushan Gopaluni. Model predictive control in industry: Challenges and opportunities. *IFAC-PapersOnLine*, 48(8):531–538, 2015.

Marco Forgione, Dario Piga, and Alberto Bemporad. Efficient calibration of embedded MPC. In *Proc. of the 21st IFAC World Congress 2020, Berlin, Germany, July 12-17 2020*, 2020.

Shlomo Geva and Joaquin Sitte. A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems Magazine*, 13(5):40–51, 1993.

Sébastien Gros and Mario Zanon. Data-driven economic nmpc using reinforcement learning. *IEEE Transactions on Automatic Control*, 65(2):636–648, 2019.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. *Advances in neural information processing systems*, 29, 2016.

Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2019.

Lukas Hewing, Kim P Wabersich, Marcel Menner, and Melanie N Zeilinger. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:269–296, 2020.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE, 2019.

Karl Henrik Johansson. The quadruple-tank process: A multivariable laboratory process with an adjustable zero. *IEEE Transactions on control systems technology*, 8(3):456–465, 2000.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817, 2001.

Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022.

Benoît Legat, Chris Coey, Robin Deits, Joey Huchette, and Amelia Perry. Sum-of-squares optimization in Julia. In *The First Annual JuMP-dev Workshop*, 2017.

Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2811–2817. IEEE, 2021.

Zishuo Li, Bo Yang, Jiayun Li, Jiaqi Yan, and Yilin Mo. Linear model predictive control under continuous path constraints via parallelized primal-dual hybrid gradient algorithm. In *2023 62nd IEEE Conference on Decision and Control (CDC)*. IEEE, 2023.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

Yiwen Lu, Bo Yang, and Yilin Mo. Two-timescale mechanism-and-data-driven control for aggressive driving of autonomous cars. In *2021 China Automation Congress (CAC)*, pages 7874–7879. IEEE, 2021.

Yiwen Lu, Zishuo Li, Yihan Zhou, Na Li, and Yilin Mo. Bridging the gaps: Learning verifiable model-free quadratic programming controllers inspired by model predictive control. *arXiv preprint arXiv:2312.05332*, 2023a.

Yiwen Lu, Bo Yang, Jiayun Li, Yihan Zhou, Hongshuai Chen, and Yilin Mo. Consecutive inertia drift of autonomous rc car via primitive-based planning and data-driven control. In *2023 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2023b.

Michael Lutter, Kim Listmann, and Jan Peters. Deep lagrangian networks for end-to-end learning of energy-based control for under-actuated systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7718–7725. IEEE, 2019.

Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning. *arXiv preprint arXiv:2205.02824*, 2022.

David Q Mayne, María M Seron, and SV Raković. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41(2):219–224, 2005.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR, 2020.

Marcel Menner, Peter Worsnop, and Melanie N Zeilinger. Constrained inverse optimal control with application to a human manipulation task. *IEEE Transactions on Control Systems Technology*, 29(2):826–834, 2019.

Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

Manfred Morari and Jay H Lee. Model predictive control: past, present and future. *Computers & chemical engineering*, 23(4-5):667–682, 1999.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps. *arXiv preprint arXiv:2110.05038*, 2021.

Pavel Osinenko, Dmitrii Dobriborsci, and Wolfgang Aumer. Reinforcement learning with guarantees: a review. *IFAC-PapersOnLine*, 55(15):123–128, 2022.

S Joe Qin and Thomas A Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

Ugo Rosolia and Francesco Borrelli. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, 2017.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

Ernest K Ryu and Wotao Yin. *Large-scale convex optimization: algorithms & analyses via monotone operators*. Cambridge University Press, 2022.

Yahya Sattar and Samet Oymak. Quickly finding the best linear model in high dimensions via projected gradient descent. *IEEE Transactions on Signal Processing*, 68:818–829, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Roland Schwan, Colin N Jones, and Daniel Kuhn. Stability verification of neural network controllers using mixed-integer programming. *IEEE Transactions on Automatic Control*, 2023.

Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: An engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5-6):1327–1349, 2021.

Raffaele Soloperto, Matthias A Müller, Sebastian Trimpe, and Frank Allgöwer. Learning-based robust model predictive control with state-dependent uncertainty. *IFAC-PapersOnLine*, 51(20): 442–447, 2018.

Mario Srouji, Jian Zhang, and Ruslan Salakhutdinov. Structured control nets for deep reinforcement learning. In *International Conference on Machine Learning*, pages 4742–4751. PMLR, 2018.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12 (4):637–672, 2020.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

Tillmann Weisser, Benoît Legat, Chris Coey, Lea Kapelevich, and Juan Pablo Vielma. Polynomial and moment optimization in julia and jump. In *JuliaCon*, 2019. URL https://pretalx.com/juliacon2019/talk/QZBKAU/.

Wuwei Wu, Jianqi Chen, and Jie Chen. Stability analysis of systems with recurrent neural network controllers. *IFAC-PapersOnLine*, 55(12):170–175, 2022.

Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018.

Bo Yang, Yiwen Lu, Xu Yang, and Yilin Mo. A hierarchical control framework for drift maneuvering of autonomous vehicles. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1387–1393. IEEE, 2022.

Dongjie Yu, Haitong Ma, Shengbo Li, and Jianyu Chen. Reachability constrained reinforcement learning. In *International Conference on Machine Learning*, pages 25636–25655. PMLR, 2022.

Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, and Yanpeng Li. Improving deep neural networks using softplus units. In *2015 International joint conference on neural networks (IJCNN)*, pages 1–4. IEEE, 2015.

Ruikun Zhou, Thanin Quartz, Hans De Sterck, and Jun Liu. Neural lyapunov control of unknown nonlinear systems with stability guarantees. *Advances in Neural Information Processing Systems*, 35:29113–29125, 2022.

Vrushabh Zinage and Efstathios Bakolas. Neural koopman lyapunov control. *Neurocomputing*, 527:174–183, 2023.

Appendix A. PDHG iterations and convergence guarantee

We introduce the PDHG iterations to solve Problem 3 and analyze its iteration structure to facilitate further design of a QP-based controller. Define the following auxiliary objective functions

$$f(z) = \mathbb{I}\{z \in \mathbb{R}_+^{m_{qp}}\},\ g(z) = \min_y \left\{ \frac{1}{2} y^\top P y + q^\top y | Hy + b = z \right\}. \tag{16}$$

Then Problem 3 is equivalent to the following problem where $z \in \mathbb{R}^{m_{qp}}$:

$$\underset{z}{\text{minimize}}\ f(z) + g(z).$$

The corresponding dual problem is

$$\underset{\lambda}{\text{maximize}}\ -f^*(-\lambda) - g^*(\lambda)$$

with dual variable $\lambda \in \mathbb{R}^{m_{qp}}$, and f^*, g^* are the Fenchel conjugate function of f and g. The Lagrangian is

$$\boldsymbol{L}(z, \lambda) = f(z) + \lambda^\top z - g^*(\lambda). \tag{17}$$

Since Problem 3 has strongly convex objective due to $P \in \mathbb{S}_{++}^{n_{qp}}$ and affine constraints, strong duality holds.

Apply the variable metric proximal point method to the saddle subdifferential:

$$\partial \boldsymbol{L}(z, \lambda) = \begin{bmatrix} 0 & I^\top \\ -I & 0 \end{bmatrix} \begin{bmatrix} z \\ \lambda \end{bmatrix} + \begin{bmatrix} \partial f(z) \\ \partial g^*(\lambda) \end{bmatrix}, \text{ and define } M = \begin{bmatrix} (1/\alpha)I & I \\ I & (1/\beta)I \end{bmatrix}.$$

Then the fix point iteration operator $(M + \partial \boldsymbol{L})^{-1} M$ applied to iteration k is

$$\begin{bmatrix} (1/\alpha)I & 2I \\ 0 & (1/\beta)I \end{bmatrix} \begin{bmatrix} z^{k+1} \\ \lambda^{k+1} \end{bmatrix} + \begin{bmatrix} \partial f\left(z^{k+1}\right) \\ \partial g^*\left(\lambda^{k+1}\right) \end{bmatrix} \ni \begin{bmatrix} (1/\alpha)z^k + \lambda^k \\ z^k + (1/\beta)\lambda^k \end{bmatrix}.$$

The iteration can be equivalently written as:

$$\lambda^{k+1} = \text{Prox}_{\beta g^*}\left(\lambda^k + \beta z^k\right), \tag{18}$$

$$z^{k+1} = \text{Prox}_{\alpha f}\left(z^k - \alpha\left(2\lambda^{k+1} - \lambda^k\right)\right). \tag{19}$$

The proximal operator of indicator function f is expressed as $\text{Prox}_{\alpha f}(\cdot) = \text{Proj}_{\mathbb{R}_+^{m_{qp}}}(\cdot)$. We derive the explicit form of $\text{Prox}_{\beta g^*}(\cdot)$ in iteration (18). According to definition,

$$\text{Prox}_{g/\beta}(\psi) = \underset{z}{\text{argmin}} \left\{ \min_y \frac{1}{2} y^\top P y + q^\top y + \frac{\beta}{2}\|z - \psi\|_2^2 \ \text{ s.t. } Hy + b = z \right\}.$$

The KKT condition of the minimization problem associated with the proximal operator is

$$\begin{pmatrix} P & 0 & H^\top \\ 0 & \beta I & -I \\ H & -I & 0 \end{pmatrix} \begin{pmatrix} y \\ z \\ d \end{pmatrix} = \begin{pmatrix} -q \\ \beta\psi \\ -b \end{pmatrix}, \tag{20}$$

where d is the dual variable of equality constraint $Hy + b = z$. Since $P \succ 0, \beta > 0$, the square matrix on the LHS of (20) is non-singular and thus the solution y to KKT condition (20) can be explicitly written as

$$\text{Prox}_{g/\beta}(\psi) = \beta H(\beta H^\top H + P)^{-1} H^\top \psi - F(HP^{-1}q - b), \text{ with } F \triangleq (I + \beta H P^{-1} H^\top)^{-1}.$$

As a result, by Moreau's Identity $\phi = \mathrm{Prox}_{\beta g^*}(\phi) + \beta\,\mathrm{Prox}_{g/\beta}(\phi/\beta)$, the PDHG iteration (18) can be written as

$$\lambda^{k+1} = F\lambda^k + \beta F z^k + \beta F(HP^{-1}q - b), \tag{21}$$

which is a simple affine transformation of (z, λ). Plugging λ^{k+1} of (21) into (19), one obtains the following iteration

$$z^{k+1} = \mathrm{Proj}_{\mathbb{R}_+^{m_{qp}}}\left((I - 2\alpha\beta F)z^k + \alpha(I - 2F)\lambda^k - 2\alpha\beta F(HP^{-1}q - b)\right). \tag{22}$$

Due to the affine form of dual step (21), the λ^{k+1} in (22) can be expressed by affine transform of z^k and λ^k. As a result, there is no λ^{k+1} in update equation (22). Thus, the iterations (21)-(22) do not need to store history dual state λ^k and current dual state λ^{k+1} simultaneously for the calculation of z^{k+1}, which is an important advantage for efficient and memory-economic GPU tensor operations.

Since strong duality holds for Problem 3 and thus Lagrange (17), we have the following convergence theorem for PDHG iterations. Define the primal and dual optimal solution to Lagrange (17) as $z^\star, \lambda^\star$. The following result provides the convergence of PDHG (Ryu and Yin, 2022):

Theorem 4 *If $\alpha, \beta > 0$, and $\alpha\beta < 1$, then iterations (21)-(22) yield $z^k \to z^\star$ and $\lambda^k \to \lambda^\star$.*

For simplicity of calculation and notations, we fix $\beta = 1$ in our implementation (7).

Appendix B. Ensuring the Feasibility of the Learned QP Problem

A caveat of the policy architecture described in Section 3 is that the learned QP problem may not be feasible, since the sequence of PDHG iterations (i.e., recursively application of (7)) will return a vector $y^{n_{iter}}$, regardless of whether the QP problem parameterized by (P, H, q, b) it attempts to solve is feasible. Even though the control input extracted from such $y^{n_{iter}}$ may lead to good empirical performance, the use of an infeasible QP as the control law is less desirable, because i) the explainability is compromised compared to deriving the control input from the optimal solution of a feasible QP; ii) as shown in Section 4, a feasible QP can simplify the theoretical analysis of the controller.

To address this issue, we propose softening the learned constraints by introducing an additional slack variable ϵ. Specifically, given learned parameters (P, H, q, b), consider the following QP problem as a replacement for the original problem in (3):

$$\begin{aligned} \underset{y\in\mathbb{R}^{n_{qp}},\epsilon\in\mathbb{R}}{\text{minimize}} \quad & \frac{1}{2}y^\top P y + q^\top y + \rho_\epsilon \epsilon^2, \\ \text{subject to} \quad & Hy + b + \epsilon\mathbf{1} \geq 0, \epsilon \geq 0, \end{aligned} \tag{23}$$

where $\rho_\epsilon > 0$ is a constant coefficient for penalizing a large slack variable. Note that the new problem (23) is always feasible regardless of the choice of (P, H, q, b), since for any (P, H, q, b) and any y, one can always choose a sufficiently large ϵ such that (y, ϵ) is a feasible solution to (23). With $\tilde{y} := [y^\top \ \epsilon^\top]^\top$ being the new decision variable, one can rearrange (23) into the standard form (3), with the parameters being:

$$\tilde{P} = \begin{bmatrix} P & 0 \\ 0 & 2\rho_\epsilon \end{bmatrix}, \quad \tilde{q} = \begin{bmatrix} q \\ 0 \end{bmatrix}, \quad \tilde{H} = \begin{bmatrix} H & \mathbf{1} \\ 0 & 1 \end{bmatrix}, \quad \tilde{b} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{24}$$

Hereafter, with a slight abuse of notation, we always assume that the control policy solves a QP problem parameterized by $(\tilde{P}, \tilde{H}, \tilde{q}, \tilde{b})$ even when we denote the parameters as (P, H, q, b).

Appendix C. Intuition Behind Learned QP Controller: a Comparison with MPC

In this appendix, we exemplify why the learned controller can overcome the limitations of MPC via a toy example.

Example 1 (Double integrator, variant of (Borrelli et al., 2017, p. 246, Example 12.1)) *Consider Problem 1 with* $n_{sys} = 2, m_{sys} = 1$*, where*

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, Q = I, R = 100, u_{\min} = -0.5, u_{\max} = 0.5, x_{\min} = \begin{bmatrix} -5 \\ -5 \end{bmatrix}, x_{\max} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}. \tag{25}$$

The task is to stabilize the system around the origin, i.e., $r = 0$.

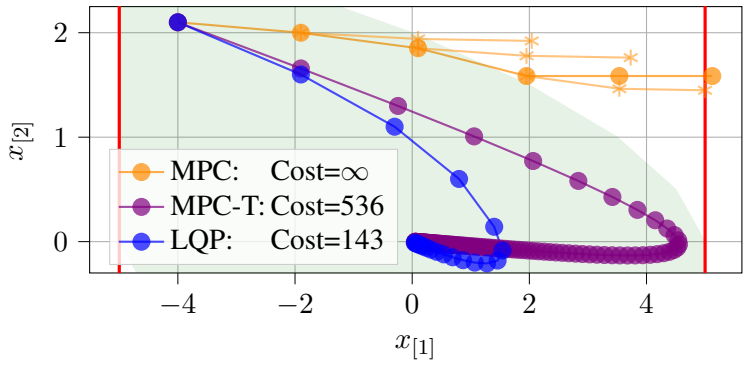


Figure 3: Comparison of trajectories under different controllers on the double integrator example, starting from the common initial state $x_0 = (-4, 2.1)$. "MPC" stands for the truncated MPC, "MPC-T" stands for MPC with a manually crafted terminal cost, and "LQP" stands for the learned QP controller. Solid dots represent realized trajectories, while asterisks stand for predicted trajectories. The green shadow stands for the maximal control invariant set (Borrelli et al., 2017, p. 192, Definition 10.10), i.e., the largest set over which one can expect *any* controller to work. The thick red lines stand for the bounds on the state.

We consider MPC with horizon $N = 3$. We also train a QP controller that solves a problem of the same dimension, i.e., $n_{qp} = Nm_{sys} = 3, m_{qp} = N(m_{sys} + n_{sys}) = 9$, for sake of fair comparison.

A notable feature of this example is the high cost of control, from which one can expect the MPC to exert a weak control effort, as long as the state does not go out of bounds within its prediction horizon. This is the case with the orange trajectory in Fig. 3: its "flatness" indicates a small local cost if we focus our attention on a N-long segment of the trajectory excluding the last step (since variations along the $x_{[2]}$-axis would require costly control inputs), but the trajectory as a whole is unacceptable because it violates the state constraints. The myopia of optimizing the cost over such a short horizon can be mitigated by introducing a terminal cost, as shown by the purple trajectory in Fig. 3, which incorporates a terminal cost $50x_{[2]}^2$. However, manually crafting of a terminal cost is nontrivial, and it is unclear how to optimize this design against the true objective, i.e., the cumulative cost along the entire trajectory before the state is brought to the origin. By comparison, the learned QP controller is free of myopia or manual crafting, since the RL algorithm naturally considers entire trajectories, and finds a QP-based control policy within the class described in Section 3 that can generate the trajectories with lowest expected total cost.

Appendix D. Proofs of Performance Guarantees and Numerical Examples

D.1. Proof of Theorem 2

Proof According to Appendix B, the optimization problem in (15) is always feasible. Therefore, we only need to prove that the control input $\pi_\theta(x)$ provided by (15) always keeps the system state in a feasible region, a sufficient condition for which is:

$$G(Ax + B\pi_\theta(x)) \leq c \text{ whenever } Gx \leq c. \tag{26}$$

The condition (26) is equivalent to the following condition:

$$\max_{x:Gx\leq c} \min_{\epsilon\in\mathbb{R}} \{\epsilon \mid G(Ax + B\pi_\theta(x)) - c \leq \epsilon\} \leq 0. \tag{27}$$

According to the strong duality of linear programming,

$$\min_{\epsilon\in\mathbb{R}} \{\epsilon \mid G(Ax + B\pi_\theta(x)) - c \leq \epsilon\} = \max_{\nu\in\mathbb{R}^p} \left\{\nu^\top (G(Ax + B\pi_\theta(x)) - c) \mid \nu \geq 0, \mathbf{1}^\top \nu = 1\right\}, \tag{28}$$

where p is the dimension of c. Substituting (28) into (27), and flipping maximization to minimization, we obtain that the condition (26) is equivalent to the optimal solution of the following problem being nonnegative:

$$\underset{x,\nu}{\text{minimize}} \quad -\nu^\top (G(Ax + B\pi_\theta(x)) - c), \tag{29a}$$

$$\text{subject to} \quad \nu \geq 0, \mathbf{1}^\top \nu = 1, \tag{29b}$$

Expanding $\pi_\theta(x)$ using (15), we obtain the following equivalent problem:

$$\underset{x,\nu}{\text{minimize}} \quad -\nu^\top (G(Ax + By_{[1:m_{sys}]}) - c), \tag{30a}$$

$$\text{subject to} \quad \nu \geq 0, \mathbf{1}^\top \nu = 1, \tag{30b}$$

$$y^* \in \arg\min \left\{(1/2) y^\top P y + q^\top y \mid Hy + b \geq 0\right\}. \tag{30c}$$

According to Appendix B, the inner problem (30c) is feasible and satisfies Slater's condition. Therefore, (30c) can be equivalently replaced by its KKT condition:

$$Py + W_q x_0 - H^\top \mu = 0, Hy + W_b x_0 + b_b \geq 0, \mu \geq 0, \mu^\top (Hy + W_b x_0 + b_b) = 0, \tag{31}$$

where μ is the dual variable of proper dimensions associated with the inequality constraint in (30c). Substituting (31) into (30c), we obtain the problem in the theorem statement, whose optimal solution being nonnegative is sufficient for persistent feasibility. ■

D.2. Proof of Theorem 3

Here we formulate the baseline MPC required for Theorem 3:

Problem 4 (Stabilizing Baseline MPC with horizon N)

$$\underset{x_{1:N}, u_{0:N-1}}{\textit{minimize}} \quad J(x_0, u_{0:N-1}) = \sum_{k=0}^{N-1} l(x_k, u_k) + V_f(x_N), \tag{32a}$$

$$\textit{subject to} \quad x_{k+1} = Ax_k + Bu_k, \quad k = 0, \cdots, N-1, \tag{32b}$$

$$x_N \in \mathcal{X}_f, \tag{32c}$$

where:

- $l(x,u) = x^\top Q x + u^\top R u$ *is the stage cost;*
- $V_f(x) = x^\top P_f x$ *is the quadratic cost-to-go function under a stabilizing linear feedback controller* $u = Kx$*;*
- $\mathcal{X}_f$ *is a positive invariant set under the same linear feedback controller* $u = Kx$*.*

Here K *can be any stabilizing linear feedback gain, e.g., it can be chosen as the LQR gain determined by* (A, B, Q, R)*.*

Theorem 3 can be proved by comparing the Lyapunov decrease of the learned controller and the above defined stabilizing baseline MPC:

Proof Consider the following candidate Lyapunov function:

$$V(x_0) = \min_{\bar{u}_{0:N}} \{J(x_0, \bar{u}_{0:N}) \mid x_{N+1}(Ax_0 + Bu_0, \bar{u}_{1:N}) \in \mathcal{X}_f\}, \tag{33}$$

where

$$J(x_0, \bar{u}_{0:N}) = \sum_{i=0}^{N} l(x_i, \bar{u}_i) + V_f(x_{N+1}) \text{ when } x_{0:N+1} \text{ is the state trajectory under } \bar{u}_{0:N}, \tag{34}$$

and $u_0 = y_{[1:m_{sys}]}$ stands for the control input given by the learned policy (15) when the state is x_0, and $x_{N+1}(Ax_0+Bu_0, \bar{u}_{1:N})$ stands for the state at time step $N+1$ if the control input sequence $\bar{u}_{1:N}$ is applied to the system starting from $x_1 = Ax_0 + Bu_0$. Since the baseline MPC is well-defined, $V(x)$ takes finite values over $\mathcal{X}_0$. One can verify that i) $u_0 = \min\{\frac{1}{2}y^\top P y | Hy+b_b \geq 0\}_{[1:m_{sys}]} = 0$ when $x_0 = 0$ and hence $V(0) = J(0,0) = 0$; ii) $V(x_0) \geq l(x_0, \bar{u}_0^*) \geq x_0^\top Q x_0 > 0$ for all $x_0 \neq 0$, where $\bar{u}_0^*$ is first control input from the optimal solution to the problem in (33). Therefore, we only need to verify that $V(Ax_0 + Bu_0) - V(x_0) < 0$ for all $x_0 \in \mathcal{X}_0 \backslash \{0\}$.

Consider the following problem:

$$\underset{x_0, \bar{u}_{0:N}, u_0, y}{\text{minimize}} \quad J(x_0, \bar{u}_{0:N}) + l(x_0, u_0) - J(x_0, (u_0, \bar{u}_{1:N})) - \epsilon\|x_0\|^2, \tag{35a}$$

$$\text{subject to} \quad Gx_0 \leq c, x_{N+1}(Ax_0 + Bu_0, \bar{u}_{1:N}) \in \mathcal{X}_f, \tag{35b}$$

$$u_0 = y_{[1:m_{sys}]}, \quad y \in \arg\min \left\{(1/2)y^\top P y + q^\top y \mid Hy + b \geq 0\right\}, \tag{35c}$$

According to Appendix B, the inner problem (35c) is feasible and satisfies Slater's condition. Therefore, (35c) can be equivalently replaced by its KKT condition, and the bilevel problem (35a)-(35c) is equivalent to the one in the theorem statement. Since the optimal value of this problem is non-negative, it holds for any $x_0 \in \mathcal{X}_0$ that:

$$\begin{aligned} 0 &\leq J(x_0, \bar{u}_{0:N}^*(x_0)) + l(x_0, u_0) - J(x_0, (u_0, \bar{u}_{1:N}^*(x_0))) - \epsilon\|x_0\|^2 \\ &= V(x_0) + l(x_0, u_0) - J(x_0, (u_0, \bar{u}_{1:N}^*(x_0))) - \epsilon\|x_0\|^2, \end{aligned} \tag{36}$$

where $\bar{u}^*(x_0)$ stands for the state sequence that achieves the optimal value of the problem in (33) when the initial state is x_0.

Now assume that the system state starts from an arbitrary $x_0 \in \mathcal{X}_0$, the control input u_0 given by the learned controller is applied for the first step. Further assume that the control input sequence $\bar{u}_{1:N}^*$ is applied for the subsequent N steps, i.e., $u_{1:N} = \bar{u}_{1:N}^*$, and that at step $N+1$, the control input

$u_{N+1} = Kx_{N+1}$ is applied, where K is the stabilizing linear feedback gain defined in Problem 4 [3]. Since $x_{N+1} \in \mathcal{X}_f$ by definition of $\bar{u}^*_{1:N}$, and $\mathcal{X}_f$ is invariant under the controller $u = Kx$, we have $x_{N+2}(x_0, u_{0:N+1}) \in \mathcal{X}_f$. Hence,

$$
\begin{aligned}
V(x_1) &\leq \sum_{i=1}^{N+1} l(x_i, u_i) + V_f(x_{N+2}) && \text{(Optimality in definition of } V\text{)} \\
&= \sum_{i=0}^{N} l(x_i, u_i) - l(x_0, u_0) + l(x_{N+1}, u_{N+1}) + V_f(x_{N+2}) && \text{(Shifting summation index)} \\
&= \sum_{i=0}^{N} l(x_i, u_i) - l(x_0, u_0) + V_f(x_{N+1}) && \text{(Definition of } V_f\text{)} \\
&= J(x_0, u_{0:N}) - l(x_0, u_0) && \text{(Definition of } J \text{ (34))} \\
&\leq V(x_0) - \epsilon \|x_0\|^2, && \text{(Inequality (36)).}
\end{aligned}
$$

Hence, we have $V(x_1) - V(x_0) < 0$ for every $x_0 \in \mathcal{X}_0 \backslash \{0\}$, which verifies the stability of the closed-loop system under the learned controller. ■

D.3. A Numerical Example

Consider the double integrator example in Appendix C. Due to the innate symmetry of the system, we consider a symmetrized parameterization of the learned QP controller (15):

$$
\begin{aligned}
&\pi_\theta(x_0) = y^*_{[1:m_{sys}]}, y^* \in \arg\min \left\{ (1/2) y^\top P y + q^\top y \mid -1 \leq \hat{H} y \leq 1 \right\}, \\
&\text{where } q = W_q x_0.
\end{aligned} \tag{37}
$$

Furthermore, the technique introduced in Appendix B is applied to (P, q, H) with penalty coefficient $\rho_\epsilon = 10$ to ensure the feasibility of the learned QP problem. The overall problem size is chosen to be $n_{qp} = 4, m_{qp} = 20$ (such that (37) has 10 pairs of symmetrized inequality constraints). A set of parameters resulting from training the controller using the PPO algorithm (Schulman et al., 2017) is:

$$
P = \begin{bmatrix} 71.693 & -59.678 & -6.376 & 0.0 \\ -59.678 & 50.029 & 5.334 & 0.0 \\ -6.376 & 5.334 & 0.695 & 0.0 \\ 0.0 & 0.0 & 0.0 & 10.0 \end{bmatrix}, W_q = \begin{bmatrix} 0.416 & 0.159 \\ -0.265 & 0.408 \\ -0.020 & 0.023 \\ 0.0 & 0.0 \end{bmatrix}, \tag{38}
$$

$$
\hat{H} = \begin{bmatrix} 0.183 & 0.568 & -0.109 & -0.839 & 0.377 & 0.285 & 0.174 & 0.356 & 0.632 & 0.0 \\ 0.034 & -0.116 & 0.149 & 0.371 & -0.858 & -0.265 & -0.427 & -0.408 & -0.657 & 0.0 \\ -1.518 & -0.686 & 0.925 & -1.335 & -0.652 & -0.643 & -1.839 & -1.337 & -1.630 & 0.0 \\ 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}^\top. \tag{39}
$$

Casting the learned parameters back to the original parameterization (15), the controller parameters P, W_q are the same as (38), while H, W_b, b_b can be obtained as follows:

$$
H = \begin{bmatrix} \hat{H}^\top & -\hat{H}^\top \end{bmatrix}^\top, W_b = 0, b_b = \mathbf{1}_{m_{qp}}. \tag{40}
$$

3. Note that $u_{1:N+1}$ are only defined here for deriving the Lyapunov decrease, but they will not be actually applied to the system, since u_0 given by the learned problem is updated in a receding horizon manner at every step.

We first verify the persistent feasibility of the learned controller on some initial set. To guess a suitable initial set, we first estimate the maximal control invariant set of the system by recursively computing the backward reachable set starting from the original. The resulting estimate of maximal control invariant set for the system (25) is $\{x|\hat{G}x \leq \hat{c}\}$, where:

$$\begin{bmatrix}\hat{G}^\top \\ \hat{c}^\top\end{bmatrix} = \begin{bmatrix} 0 & 1 & 0.71 & -1 & -0.71 & 0 & 0.45 & -0.24 & -0.45 & -0.32 & 0.2 & 0.16 & 0.24 & 0.32 & -0.16 & -0.2 \\ -1 & 0 & 0.71 & 0 & -0.71 & 1 & 0.89 & -0.97 & -0.89 & -0.95 & 0.98 & 0.99 & 0.97 & 0.95 & -0.99 & -0.98 \\ 2.8 & 5 & 3.5 & 5 & 3.5 & 2.8 & 2.5 & 2.0 & 2.5 & 2.1 & 2.0 & 2.1 & 1.9 & 2.1 & 2.1 & 2.0 \end{bmatrix}. \tag{41}$$

A guess of the initial set can be obtained by shrinking the estimated maximal control invariant set slightly. In particular, we guess the initial set $\{x|Gx \leq c\}$, where:

$$G = \hat{G}, c = \hat{c} - 0.2, \tag{42}$$

where "$-$" stands for element-wise subtraction.

With parameters defined in (38), (40) and (42), the feasibility verification problem in Theorem 2 is fully instantiated. In what follows, we find the upper and lower bounds to this problem, with all computation done on a workstation with 16-Core Intel i7-13700KF CPU. Solving the problem using the local nonlinear programming solver IPOPT (Biegler and Zavala, 2009), which finishes in 0.4 seconds, we quickly obtain an *upper bound* to the optimal value: $p^* \leq 0.0828$. Note that this is only a hint that the controller may be persistently feasible on the initial set $\{x|Gx \leq c\}$, but not a certificate, since the upper bound provided by a local solver may not be tight. To obtain a certificate, we use the `SumOfSquares.jl` toolbox in Julia (Legat et al., 2017; Weisser et al., 2019) to relax this nonconvex QCQP problem to a SDP problem, and solve the SDP problem using the Mosek solver (ApS, 2023). The solver finishes in 216 seconds, returning a *lower bound* to the optimal value of the original problem: $p^* \geq 0.0803$. This nonnegative lower bound certifies that the learned controller is indeed persistently feasible on the initial set $\{x|Gx \leq c\}$, and the lower and upper bounds are quite close in this example.

We next verify the asymptotic stability of the learned controller on the same initial set $\{x|Gx \leq c\}$. For the current problem, it suffices to choose baseline MPC horizon $N = 0$, which reduces the verification problem in Theorem 3 to finding a quadratic Lyapunov function $V_f(x) = x^\top P_f x$ such that the following problem has nonnegative optimal value:

$$\underset{x_0,y,\mu}{\text{minimize}} \quad V_f(x_0) - V_f(Ax_0 + By_{[1:m_{sys}]}) - \epsilon\|x_0\|^2, \tag{43a}$$

$$\text{subject to} \quad Gx_0 \leq c, \tag{43b}$$

$$Py + W_q x_0 - H^\top \mu = 0, Hy + W_b x_0 + b_b \geq 0, \mu \geq 0, \mu^\top (Hy + W_b x_0 + b_b) = 0. \tag{43c}$$

To find such a Lyapunov function, we first approximate the control policy $\pi(x)$ defined in (37) by a linear feedback controller $u = \hat{K}x$ near the origin: by choosing $x_1 = \begin{bmatrix}0.1 & 0\end{bmatrix}^\top$ and $x_2 = \begin{bmatrix}0 & 0.1\end{bmatrix}^\top$, we obtain:

$$\hat{K} = \begin{bmatrix}\pi(x_1) & \pi(x_2)\end{bmatrix}\begin{bmatrix}x_1 & x_2\end{bmatrix}^{-1} = \begin{bmatrix}-0.20 & -1.28\end{bmatrix}, \tag{44}$$

and setting P_f to be the solution to the Lyapunov equation $(A+B\hat{K})P_f(A+B\hat{K})^\top + Q + \hat{K}^\top R\hat{K} = 0$, we obtain:

$$P_f = \begin{bmatrix}5.64 & 12.59 \\ 12.59 & 58.40\end{bmatrix}. \tag{45}$$

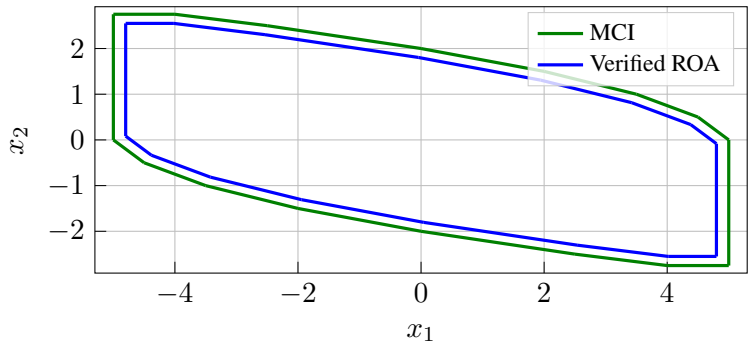


Figure 4: Comparison of boundaries of the Maximal Control Invariant (MCI) set and verified Region Of Attraction (ROA) for the double integrator example.

Plugging (45) into the problem (43a)-(43c), we obtain a fully defined stability verification problem. Similarly to the feasibility verification problem, we try to solve the stability verification using both the local solve IPOPT and the global solver based on `SumOfSquares.jl` and Mosek. The local solver finishes in 0.007 seconds, which the global solver finishes in 1.4 seconds, both returning the optimal value $p^* = 0$ up to numerical accurary 10^{-5} (the optimal value is attained at $x_0 = 0$). This verifies the asymptotic stability of the learned controller on the initial set $\{x|Gx \leq c\}$.

The maximal control invariant set $\{x|\hat{G}x \leq \hat{c}\}$ and verified region of attraction $\{x|Gx \leq c\}$ for this example are visualized in Fig. 4. The verified region of attraction is only slightly smaller than the maximal control invariant set (accounts for 92% of the area of the maximal control invariant set), which demonstrates the effectiveness of our proposed verification method.

Appendix E. Benchmarking Setup and Additional Experiment Results

E.1. Environment Definition for RL Training

All learned QP controllers are trained using Reinforcement Learning (RL). To simulate the constrained optimal control problem described in Problem 1, the RL environment consists of components described as follows:

- **Observation**: the observation variable is $x = [x_0^\top, r^\top]^\top$ when training policies for tracking a reference r, and $x = x_0$ when training policies for stabilizing the system around the origin.
- **Action Preprocessing**: given policy output u_0, the actual action applied to the system is chosen to be $u = \text{clip}(u_0, u_{\min}, u_{\max})$, where $\text{clip}(x, a, b) = \min(\max(x, a), b)$ element-wise. In other words, the action is clipped to the bounds on the control input, even if the policy output is outside of the bounds.
- **Reward**: let (x, u, x') be a state, action, and next state triple. Recall from (1a) that the LQ cost is defined as $l(x', u) = (x' - r)^\top Q(x' - r) + u^\top Ru$. Taking into account the original LQ cost,

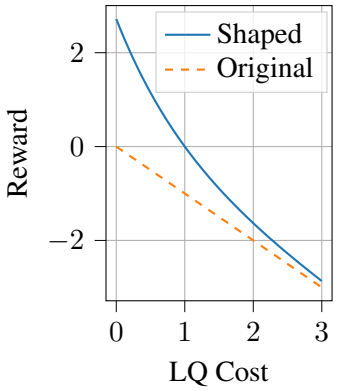


Figure 5: Visualization of the reward shaping scheme used (with hyperparameters $\rho_{sta} = 1, c_1 = 1, c_2 = 1$): a positive reward is added to small LQ cost, in order to encourage the controller to minimize the steady-state cost.

the state constraint, and a reward shaping term for minimizing the steady-state cost, the overall reward function for RL training is chosen as:

$$\begin{aligned}\text{Reward}(x,u,x') = &-l(x',u) && \text{(Original LQ cost)}\\ &-\rho_{pen}\left(1-\mathbf{1}_{\{x_{\min}\le x'\le x_{\max}\}}\right) && \text{(Penalty on constraint violation)}\\ &-\rho_{sta}\exp(-c_1(l(x',u)-c_2)) && \text{(Reward shaping term),}\end{aligned} \tag{46}$$

where $\rho_{sta}, c_1 > 0, c_2$ are hyperparameters for reward shaping. See Fig. 5 for a visualization of the reward shaping scheme.

- **Episode Definition**: at the start of an episode, both the initial state and the reference are randomly sampled according to a task-specific distribution (see "Initial State" and "Reference" in Table 3), and an episode is terminated on either of the following conditions:
  - The state constraint is violated;
  - The episode length exceeds a predefined threshold (see "Episode length" in Table 3).
- **Domain randomization**: when training the controllers for the nominal systems (subsection 5.1), no domain randomization is used, i.e., the environment simulates the deterministic dynamics (1b). On the other hand, when training the robust controllers for uncertain systems (subsection 5.2), the environment simulates the following perturbed dynamics:

$$x_{k+1} = (A+\Delta A)x_k + (B+\Delta B)u_k + w_k, \tag{47}$$

where two sources of randomness are incorporated into the dynamics:

  - Perturbation in the system parameters: $\Delta A, \Delta B$ are randomly sampled from a task-specific distribution at the start of each episode (see "System Perturbation" in Table 3);
  - Process noise: w_k is sampled from a task-dependent distribution at each time step (see "Process Noise" in Table 3).

E.2. Task Definitions

The details on the benchmarking tasks are summarized in Table 3.

Table 3: Task definitions for benchmarking.

	Double Integrator	**Quadruple Tank**	**Cartpole Balance**	**Drift Control**
Source	(Borrelli et al., 2017, p. 246, Example 12.1)	Johansson (2000) (original source); Li et al. (2023) (linearization)	Geva and Sitte (1993)	Lu et al. (2023b)
Remark	Classical example where improperly designed MPC may fail.	Stable linearized system with adjustable reference.	Unstable system with slight nonlinearity.	Highly nonlinear robotics control task.

	Double Integrator	**Quadruple Tank**	**Cartpole Balance**	**Drift Control**
Nominal dynamics	$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.	$A = \begin{bmatrix} 0.98 & 0 & 0.04 & 0 \\ 0 & 0.99 & 0 & 0.03 \\ 0 & 0 & 0.96 & 0 \\ 0 & 0 & 0 & 0.97 \end{bmatrix}$, $B = \begin{bmatrix} 0.83 & 0 \\ 0 & 0.62 \\ 0 & 0.47 \\ 0.3 & 0 \end{bmatrix}$.	$\begin{bmatrix} m_c + m_p & m_p l c_\theta \\ m_p l c_\theta & m_p l^2 \end{bmatrix} \begin{bmatrix} \ddot{p}_x \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} u + m_p l s_\theta \dot{\theta}^2 \\ m_p g l s_\theta \end{bmatrix}$, where state variable is $x = [p_x, \dot{p}_x, \theta, \dot{\theta}]^\top$, and nominal parameters are $m_c = 1, m_p = 0.1, l = 0.55$. Discretized with time step 0.1s. Simulation uses nonlinear dynamics, while baseline MPC uses linearized dynamics.	(Lu et al., 2023b, Eq. (5)), with intermediate variables determined by (Lu et al., 2021, Eq. (6)). State variable is $x = [r, \beta, V]^\top$, input variable is $u = [\delta, \omega]^\top$ and nominal parameters are $B = 0.9, C = 2.13, D = 0.382$. Discretized with time step 0.1s. Nonlinear dynamics is used.
Constraints	$-5 \le x \le 5$, $-0.5 \le u \le 0.5$.	$0 \le x \le 20$, $0 \le u \le 8$.	$-2 \le p_x \le 2$, $-0.5 \le \theta \le 0.5$, $-10 \le u \le 10$.	$-7.6 \le r \le 7.6$, $-3.8 \le \beta \le 3.8$, $0 \le V \le 4$, $-0.6 \le \delta \le 0.6$, $1 \le \omega \le 7$.
Cost matrices	$Q = I, R = 100I$.	$Q = I, R = 0.1I$.	$Q = \text{diag}([1, 10^{-4}, 1, 10^{-4}])$, $R = 10^{-4}$.	$Q = I$, $R = \text{diag}([1, 0.01])$.
Initial state	Uniformly sampled over state space, then projected to maximal control invariant set.	Uniform over $[0, 16]^4$ (ensures that initial state is within maximal control invariant set).	$p_x \sim \mathcal{U}[-1.8, 1.8]$, $\dot{p}_x \sim \mathcal{U}[-1, 1]$, $\theta \sim \mathcal{U}[-0.1, 0.1]$, $\dot{\theta} \sim \mathcal{U}[-0.1, 0.1]$.	$r = \beta = V = 0$ (starts from still).
Reference	$x = 0$ (stabilizing around the origin).	Uniform over state space.	$p_x \sim \mathcal{U}[-2, 2]$, $\dot{p}_x = \theta = \dot{\theta} = 0$ (balancing pole at random position).	$r \sim s \times \mathcal{U}[1.5, 2.5]$, $\beta \sim -s \times \mathcal{U}[0.8, 1.2]$, $V \sim \mathcal{U}[1.5, 2.5]$, $s \sim \mathcal{U}\{-1, 1\}$ (Circular drifting path in either CW or CCW direction).
Episode length	100	500	100	1500
System perturbation	$\Delta A = \begin{bmatrix} \delta_1 & \delta_2 \\ 0 & \delta_3 \end{bmatrix}$, $\Delta B = \begin{bmatrix} 0 \\ \delta_4 \end{bmatrix}$, $\delta_i \overset{i.i.d.}{\sim} \mathcal{U}[-0.05, 0.05]$	$\Delta A = \begin{bmatrix} \delta_1 & 0 & \delta_3 & 0 \\ 0 & \delta_2 & 0 & \delta_4 \\ 0 & 0 & -\delta_1 & 0 \\ 0 & 0 & 0 & -\delta_2 \end{bmatrix}$, $\Delta B = \begin{bmatrix} 8.3\delta_5 & 0 \\ 0 & 6.2\delta_6 \\ 0 & 6.2\delta_6 \\ 3\delta_5 & 0 \end{bmatrix}$, $\delta_i \overset{i.i.d.}{\sim} 0.002 \times \mathcal{U}[-1, 1]$.	$m_c \sim \mathcal{U}[0.7, 1.3]$, $m_p \sim \mathcal{U}[0.07, 0.13]$, $l \sim \mathcal{U}[0.4, 0.7]$.	$B \sim \mathcal{U}[0.8, 1]$, $C \sim \mathcal{U}[2, 2.5]$, $D \sim \mathcal{U}[0.3, 0.4]$.
Process noise	$w_k \overset{i.i.d.}{\sim} \mathcal{N}(0, 0.05I)$.	$w_k \overset{i.i.d.}{\sim} \mathcal{N}(0, 0.1I)$.	$w_k \overset{i.i.d.}{\sim} \mathcal{N}(0, \text{diag}([0, 0.1, 0, 0.1]))$	Disturbance on f_{ij} defined in (Lu et al., 2023b, Eq. (5)), sampled from AR(1) process with parameter 0.97 and noise standard deviation 0.03.

E.3. Training Details

All the training (MLP and LQP) is done using the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017). The same set of hyperparameters is used for all the experiments, except for the size of the function approximators (number of neurons for MLP, and number of variables/constraints for LQP), since multiple sizes are tested for each task. The hyperparameters used for training are summarized in Table 4.

Table 4: Hyperparameters used for training.

Parameter	**Value**
Common	
Number of epochs	5000
Optimizer	Adam (Kingma and Ba, 2014)
Critic learning rate	Linear annealing 1e-3 $\rightarrow$ 2e-6
Actor learning rate	Linear annealing 5e-4 $\rightarrow$ 1e-6
Entropy coefficient	0
Reward discount factor (γ)	0.99
Target smoothing parameter (τ)	0.95
Batch size	100000
Horizon length	20
PPO mini-epochs	1
PPO clip	0.2
Penalty on constraint violation (ρ_{pen}, see (46))	100000
Reward shaping (see (46))	$\rho_{sta} = 50, c_1 = 0.05, c_2 = 2$
MLP	
Number of hidden layers	3
Number of neurons in hidden layers	$[4n, 2n, n]$, where $n \in \{8, 16, 32, 64\}$
Activation function	ELU
LQP	
Number of variables/constraints (n_{qp}, m_{qp})	$(n, 6n)$, where $n \in \{4, 8, 16\}$
Number of unrolled iterations (n_{iter})	10
PDHG step size (α)	1
Residual regularization coefficient (ρ_{res})	1e-3
Penalty for infeasibility (ρ_ϵ in Appendix B)	10

E.4. Definition of MPC Baselines

In this subsection, we provide the details on the MPC baselines used in the benchmarking experiments. For each MPC baseline appearing in Tables 1 and 2, the problem formulation (either in explicit formulas or provided by an existing software package) and the solver used for solving the MPC problem are summarized in Table 5.

Table 5: Definition of MPC baselines.

Name	Formulation	Solver
MPC(N)	Problem 2, translated to QP via (5)-(6).	OSQP (Stellato et al., 2020)
MPC-T(N)	Problem 2 with terminal cost $\rho\|x_N\|^2$ added to the objective, translated to QP similarly to (5)-(6). Weight ρ grid-searched over $\{1, 10, 100\}$, picking the best-performing one.	OSQP (Stellato et al., 2020)
Tube	Tube MPC implemented by open-sourced code of Doff-Sotta and Cannon (2022), simplified for linear systems. Horizon $N = 16$ and terminal weight $\rho = 10$. Tube size grid-searched over $\{0.05, 0.1, 0.2, 0.3\}$, since using the actual upper bound on disturbance magnitude may result in infeasibility.	Mosek (ApS, 2022)
Scenario	Scenario-based MPC implemented by the do-mpc toolbox (Fiedler et al., 2023). Horizon $N = 16$ and terminal weight $\rho = 10$. Three scenarios are generated for each of A, B, w.	IPOPT (Biegler and Zavala, 2009)

E.5. Evaluation Metrics

In this subsection, we provide the details on the evaluation metrics used in the benchmarking experiments. For each controller on each task, we run $N_t = 10000$ independent trials/episodes. For each trial $i = 1, \dots, N_t$, we denote the episode length as $T^{(i)}$, which can either be the maximum episode length listed in Table 3 or smaller (due to early termination caused by constraint violation). We denote the state and input trajectories of the i-th trial as $\{x_k^{(i)}\}_{k=1}^{T^{(i)}}$ and $\{u_k^{(i)}\}_{k=1}^{T^{(i)}}$, respectively. The random generators for initial state, reference, system perturbation and process noise are seeded separately, such that for different controllers on the same task, the i-th trial always has the same initial state, reference, system perturbation and realization of process noise. The metrics appearing in Tables 1 and 2 are defined as follows:

- **Fail%**: percentage of trials that are early-terminated due to constraint violation, i.e.,

$$\text{Fail\%} = \frac{1}{N_t} \sum_{i=1}^{N_t} \mathbf{1}_{\{T^{(i)} < T_{\max}\}} \times 100, \tag{48}$$

  where $T_{\max}$ is the task-dependent maximum episode length listed in Table 3.

- **Cost**: average LQ cost per time step, i.e.,

$$\text{Cost} = \frac{1}{\sum_{i=1}^{N_t} T^{(i)}} \sum_{i=1}^{N_t} \sum_{k=1}^{T^{(i)}} l(x_k^{(i)}, u_k^{(i)}), \tag{49}$$

  where $l(x, u)$ is the LQ cost defined in (1a).

- **P-Cost**: average LQ cost per time step, including the penalty for constraint violation, i.e.,

$$\text{P-Cost} = \frac{1}{\sum_{i=1}^{N_t} T^{(i)}} \sum_{i=1}^{N_t} \sum_{k=1}^{T^{(i)}} l(x_k^{(i)}, u_k^{(i)}) + \rho_{pen} \left(1 - \mathbf{1}_{\left\{x_{\min} \le x_k^{(i)} \le x_{\max}\right\}}\right), \tag{50}$$

  where ρ_{pen} is the penalty for constraint violation defined in Table 4.

- **FLOPs**: number of Floating Point Operations (FLOPs) required for each time step k. For MLP and LQP, this value is constant since the path of forward propagations is fixed. For MPC, this value is variable since the number of iterations required for solving the MPC problem to the desired accuracy at each step is variable, and is reported in the format $\text{median}_{+(\text{max}-\text{median})}$.

- **Time**: computation time per step. For MLP and LQP, the time is measured by the time of batch operation on an NVIDIA RTX 4090 GPU. For MPC and robust MPC methods, the time is measured by solving a single instance of the MPC problem on a single core of an Intel i7-13700KF CPU. Overhead may apply since the testing program is parallelized. The time is reported in the format $\text{median}_{+(\text{max}-\text{median})}$ (subscript omitted when fluctuations are negligible).

- **#Params**: number of learnable policy parameters, i.e., the total size of weight matrices and bias vectors in the MLP policy, or the total size of W_q, W_b, b_b defined in (8) in the LQP policy. Note that critic parameters are excluded since the comparison is mainly concerned with the *deployment* of learned policies.

E.6. Complete Benchmarking Results

In this subsection, we present extended versions of Tables 1 and 2, which include additional configurations of the methods, as well as additional tasks.

The benchmarking results for nominal systems are presented in Table 6, which extends Table 1 by including results for different weights of terminal cost in MPC, for MLPs with different sizes, and for an additional size of LQP. It also includes complete benchmarking results for the double integrator example. One can observe from the table that the proposed LQP method consistently achieves control performance similar to MPC with sufficiently long horizon and proper terminal cost, and MLP with sufficiently large size, while being more efficient in terms of FLOPs and number of learnable policy parameters.

The benchmarking results for perturbed systems are presented in Table 7, which extends Table 2 by including more systems as well as baselines. On one hand, under the magnitude of perturbation considered in the experiments, it is shown that the proposed LQP method is more effective and computationally lighter than robust MPC baselines. Although it might be possible to design ad-hoc robust MPC methods tailored for improved performance and efficiency, LQP works off-the-shelf, in the sense that it requires minimal task-specific knowledge, except for a simulator taking the disturbances into account. On the other hand, LQP is even marginally better than MLP controllers, with much fewer learnable policy parameters, which hints at the suitability of the particular parameterized policy class for control tasks.

Table 6: Performance comparison on nominal systems (extended version).

Method \ Metrics	Quadruple Tank					Cartpole Balancing				
	Fail%	Cost	P-Cost	FLOPs	#Params	Fail%	Cost	P-Cost	FLOPs	#Params
MPC(2)	16.59	236.1	275.7	$95K_{+1.15M}$	-	100.0	1.36	129	$67K_{+814K}$	-
MPC(16)	4.27	228.3	237.2	$75M_{+0}$	-	46.86	0.34	8.39	$3.9M_{+47M}$	-
MPC-T(2, 1)	15.75	235.0	272.2	$126K_{+1.12M}$	-	100.0	1.39	122	$89K_{+792K}$	-
MPC-T(2, 10)	10.53	226.7	250.2	$220K_{+1.03M}$	-	100.0	1.41	122	$89K_{+792K}$	-
MPC-T(2, 100)	4.23	239.6	248.5	$470K_{+780K}$	-	100.0	1.41	122	$89K_{+792K}$	-
MPC-T(16, 1)	4.02	228.0	236.4	$24M_{+50M}$	-	33.06	0.33	4.96	$3.9M_{+48M}$	-
MPC-T(16, 10)	3.22	224.8	231.5	$26M_{+48M}$	-	4.74	0.30	0.79	$52M_{+0}$	-
MPC-T(16, 100)	0.37	268.5	269.2	$75M_{+0}$	-	5.01	0.34	0.87	$52M_{+0}$	-
RL-MLP(8)	0.08	315.4	315.6	2K	952	4.52	0.94	1.41	2K	856
RL-MLP(16)	0.00	313.3	313.3	6K	3.2K	4.67	0.79	1.28	6K	3K
RL-MLP(32)	0.03	266.7	266.7	23K	11.5K	4.65	0.83	1.32	23K	11K
RL-MLP(64)	0.03	291.3	291.4	87K	43K	3.23	0.57	0.91	87K	43K
LQP(4, 24)	0.18	272.5	272.8	14K	354	3.49	0.76	1.12	14K	246
LQP(8, 48)	0.22	230.7	231.1	53K	916	3.33	0.64	0.99	53K	700
LQP(16, 96)	0.13	227.4	227.6	208K	2.7K	4.11	0.44	0.87	208K	2.2K

Method \ Metrics	Double Integrator				
	Fail%	Cost	P-Cost	FLOPs	#Params
MPC(2)	83.80	1.78	945.06	$32.8K_{+218K}$	-
MPC(16)	55.55	0.42	246.11	$1.94M_{+72.6M}$	-
MPC-T(2, 1)	80.10	1.16	755.63	$32.8K_{+218K}$	-
MPC-T(2, 10)	62.52	0.69	328.06	$32.8K_{+312K}$	-
MPC-T(2, 100)	55.43	0.82	245.31	$32.8K_{+468K}$	-
MPC-T(16, 1)	55.55	0.42	246.11	$1.94M_{+72.6M}$	-
MPC-T(16, 10)	55.55	0.44	246.12	$1.94M_{+72.6M}$	-
MPC-T(16, 100)	55.55	0.88	244.77	$1.94M_{+72.6M}$	-
RL-MLP(8)	64.01	1.32	352.00	1.61K	760
RL-MLP(16)	0.0	21.00	21.00	5.78K	2.8K
RL-MLP(32)	0.0	0.55	0.55	21.8K	10.7K
RL-MLP(64)	0.0	1.02	1.02	84.5K	41.9K
LQP(4, 24)	0.0	0.56	0.56	13.8K	186
LQP(8, 48)	0.0	0.54	0.54	53.0K	580
LQP(16, 96)	0.0	0.52	0.52	207K	2.0K

Methods: MPC(N) = naive MPC (Problem 2) with horizon N; MPC-T(N, ρ) = MPC with horizon N and terminal cost $\rho\|x_N\|^2$; RL-MLP(n) = reinforcement learning controller with MLP policy with hidden layer sizes $[4n, 2n, n]$; LQP(n_{qp}, m_{qp}) = proposed learned QP controller with problem dimensions (n_{qp}, m_{qp}). Metrics: see Appendix E.5.

Table 7: Performance comparison on perturbed systems (extended version).

Method \ Metrics	Quadruple Tank					Cartpole Balancing				
	Fail%	Cost	P-Cost	Time(s)	#Params	Fail%	Cost	P-Cost	Time(s)	#Params
MPC	82.6	216.8	713.4	$0.25_{+0.56}$	-	19.1	0.08	0.57	0.06_{+0}	-
Scenario	16.4	236.9	273.2	5.21_{+18}	-	100.0	2.19	100.88	$6.5_{+1.9}$	-
Tube(0.05)	81.9	233.3	696.4	2.17_{+39}	-	100.0	2.41	112.43	$3.3_{+82.5}$	-
Tube(0.1)	82.1	218.1	675.3	2.22_{+44}	-	100.0	2.41	112.43	$3.3_{+78.8}$	-
Tube(0.2)	82.5	233.3	597.9	1.98_{+38}	-	100.0	2.41	112.43	$2.8_{+59.7}$	-
Tube(0.3)	100.0	255.2	1322	1.60_{+31}	-	100.0	2.41	112.43	$3.5_{+89.5}$	-
RL-MLP(8)	2.8	249.0	254.7	2×10^{-5}	1.0K	4.7	0.63	0.73	2K	856
RL-MLP(16)	2.0	241.5	245.6	8×10^{-5}	3.2K	4.8	0.40	0.50	6K	3K
RL-MLP(32)	1.5	239.2	242.3	3×10^{-4}	11.5K	3.6	0.32	0.39	23K	11K
RL-MLP(64)	1.3	238.9	241.5	1×10^{-3}	43K	3.2	0.21	0.28	87K	43K
LQP(4, 24)	2.5	256.7	261.8	2×10^{-4}	354	3.9	0.12	0.20	14K	246
LQP(8, 48)	1.7	240.4	243.8	7×10^{-4}	916	4.3	0.11	0.20	53K	700
LQP(16, 96)	1.4	240.6	243.4	2×10^{-3}	2.7K	4.0	0.18	0.27	208K	2.2K

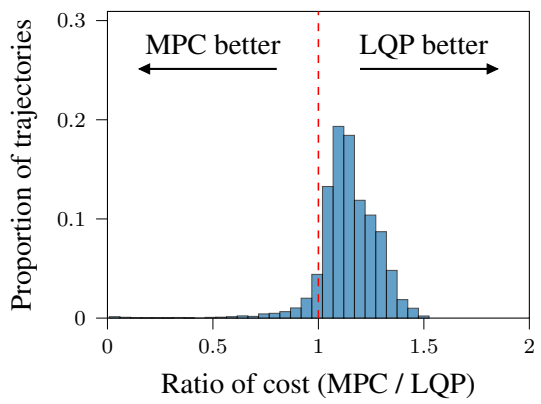
Method \ Metrics	Double Integrator				
	Fail%	Cost	P-Cost	Time(s)	#Params
MPC	36.4	0.60	114.51	$0.002_{+0.02}$	-
Scenario	30.2	0.86	87.01	$5.0_{6.1}$	-
Tube(0.05)	100.0	10.58	12105	$0.43_{+4.66}$	-
Tube(0.1)	100.0	10.58	12105	$0.46_{+3.96}$	-
Tube(0.2)	100.0	10.58	12105	$0.46_{+4.20}$	-
Tube(0.3)	100.0	10.58	12105	$0.44_{+4.38}$	-
RL-MLP(8)	30.2	1.18	87.33	2×10^{-5}	760
RL-MLP(16)	30.2	0.90	87.05	8×10^{-5}	2.8K
RL-MLP(32)	31.6	0.80	92.75	3×10^{-4}	10.7K
RL-MLP(64)	31.4	0.76	87.73	1×10^{-3}	41.9K
LQP(4, 24)	27.5	1.03	76.58	2×10^{-4}	186
LQP(8, 48)	27.5	0.84	76.38	7×10^{-4}	580
LQP(16, 96)	27.4	0.86	76.03	2×10^{-3}	2.0K

Methods: MPC = MPC (Problem 2) with horizon $N = 16$ and terminal cost $10\|x_N\|^2$; Scenario = scenario-based MPC (see Table 5); Tube(s) = tube MPC with tube size s (see Table 5); RL-MLP(n) = reinforcement learning controller with MLP policy with hidden layer sizes $[4n, 2n, n]$; LQP(n_{qp}, m_{qp}) = proposed learned QP controller with problem dimensions (n_{qp}, m_{qp}). Metrics: see Appendix E.5.

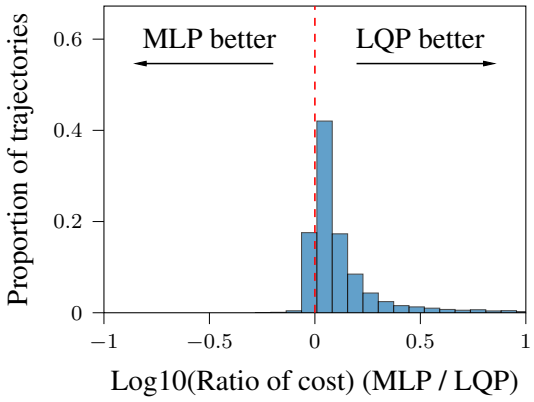
E.7. Trajectory-wise Comparisons

In this subsection, we provide trajectory-wise comparisons of the proposed LQP controllers with MPC and MLP controllers. For each task, we use the nominal system, and randomize the initialization for 10,000 times. Under each initialization, we run the controllers LQP(16, 96), MPC(16, 10) and RL-MLP(32) in Table 6. We do pairwise comparisons of (MPC, LQP) and (MLP, LQP) in terms of the cumulative cost along the trajectories: for each initialization where both controllers can run until the maximum number of steps without violating the constraints, we record the ratios of cumulative costs along the trajectories, i.e., (Cost of MPC / Cost of LQP) or (Cost of MLP / Cost of LQP). The histograms of these ratios are plotted in Figure 6. The following observations can be made from the histograms:

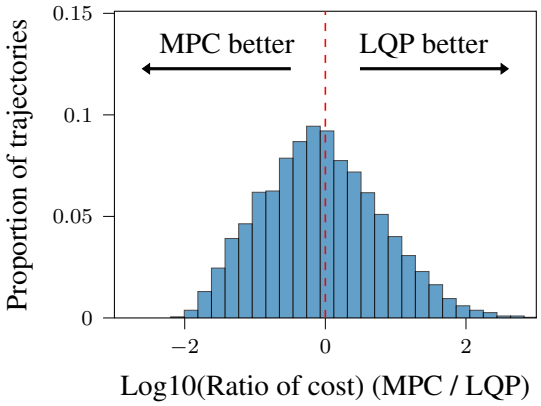
- LQP is overall competitive with MPC: it slightly outperforms MPC in the quadruple tank task, is approximately on par with MPC in the cartpole balancing task, and is slightly worse in the double integrator task. The last is reasonable, since it is evident from Table 6 that MPC fails a significant proportion of the trials while LQP does not, and the cost of the failed trials is not included in the histogram. That said, the cost ratio is still close to 1, indicating the competitiveness of LQP. The slight suboptimality of LQP can be attributed to the RL algorithm used for training, the improvement of which is a potential future work.
- LQP can act as a drop-in replacement for MLP in control tasks: LQP can slightly outperform MLP on average, and furthermore, in all of the three tasks considered, the cost ratios of (MLP/LQP) are *heavy-tailed* and inclined towards the right, showing that LQP can effectively mitigate the extreme cases where MLP performs poorly. This is an empirical counterpart of the verifiability of LQP brought by its structure: it can potentially improve the reliability of the learned controllers.
- LQP can display distinct behavior compared to MPC and MLP: it is evident that LQP neither dominates over MPC or MLP, nor vice versa. Especially, in the cartpole task, the cost of LQP and be either 100 times higher than that of MPC/MLP, or 100 times lower. This is an interesting phenomenon indicating that LQP can potentially provide a different perspective on the control problem, though further investigation is needed to understand the benefits and/or drawbacks of this distinct behavior.



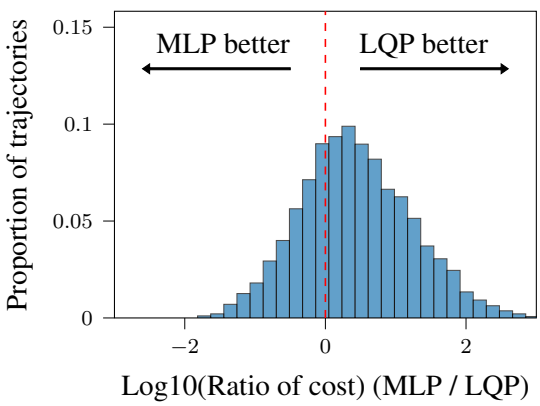
(*a*) Quadruple tank, MPC vs. LQP



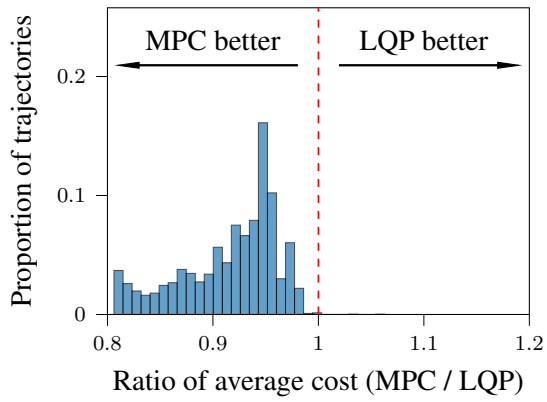
(*b*) Quadruple tank, MLP vs. LQP



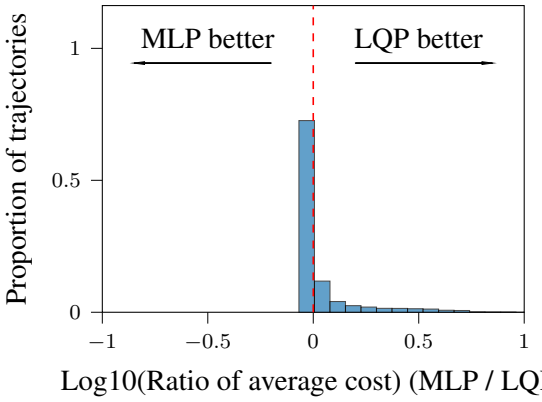
(*c*) Cartpole balancing, MPC vs. LQP



(*d*) Cartpole balancing, MLP vs. LQP



(*e*) Double integrator, MPC vs. LQP



(*f*) Double integrator, MLP vs. LQP

Figure 6: Histogram of the ratio of MPC/MLP cost to LQP cost. For each task, each pair of controllers are run on 10,000 different trials. In each trial, the same initialization is used for both controllers. The ratios of cumulative costs among trials completed by both controllers are plotted. In some subfigures, the x-axis is in log scale for the ease of observation.