Rollout Then Optimize: A One-Step Newton Refinement of Learned Policies for Nonlinear Model Predictive Control

Andrea Ghezzi[1], Rudolf Reiter[2], Katrin Baumgärtner[1], Alberto Bemporad[3], Moritz Diehl[1,4]

***Abstract*— We propose a computationally efficient rollout-then-optimize method to improve a learned control policy at deployment time. A learned policy provides a nominal trajectory, which is refined online by a single Newton step implemented via a Riccati recursion within a model predictive control (MPC) scheme. This refinement combines model knowledge with the learned policy at minimal additional computational cost. We establish bounds on the approximation error of the learned policy relative to the MPC policy and show that one Newton step reduces the suboptimality of the learned rollout quadratically in the policy approximation error. The proposed controller is validated in simulation on a constrained trajectory-tracking task for a quadcopter with nonlinear dynamics. Results highlight that the Newton step significantly improves the learned policy, achieving performance close to a fully converged MPC solution while requiring roughly half of the computational time. The code is available at `https://github.com/aghezz1/rl-riccati`.**

I. Introduction

Model predictive control (MPC) and reinforcement learning (RL) are two prominent approaches for approximately solving infinite-horizon optimal control problems [1]. Both provide feedback policies that minimize long-term cost while respecting system dynamics and constraints. However, they differ significantly in how this objective is achieved and in their computational trade-offs. MPC repeatedly solves finite-horizon optimal control problems online and can explicitly enforce constraints. To achieve high performance, long prediction horizons are often required, which leads to substantial computational effort at deployment. In contrast, RL shifts most of the computational burden offline: once training is complete, the learned policy can be evaluated very efficiently. However, obtaining high-performance and constraint-satisfying policies typically requires extensive training and large amounts of data, especially for nonlinear systems. The motivation of this work is to bridge this gap. We aim to retain the fast online evaluation of learned policies while recovering part of the performance and constraint-handling capabilities of MPC without solving a nonlinear program (NLP) at each control step.

We consider the control of nonlinear dynamical systems $s_{t+1} = f(s_t, u_t)$, with state $s_t \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$, control $u_t \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$, $f : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$, and $t \in \mathbb{N}$. The optimal feedback law $\pi^* : \mathcal{S} \to \mathcal{U}$ is defined as the solution of the infinite-horizon optimal control problem (OCP)

$$\begin{aligned} J^*(s) \coloneqq \min_{\pi} \quad & \sum_{t=0}^{\infty} \ell(s_t, u_t) \\ \text{s.t.} \quad & s_0 = s, \\ & s_{t+1} = f(s_t, u_t),\ t \in \mathbb{N}, \\ & u_t = \pi(s_t), \qquad t \in \mathbb{N}, \\ & g(s_t, u_t) \leq 0, \qquad t \in \mathbb{N}, \end{aligned} \tag{1}$$

where the stage cost $\ell : \mathcal{S} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ and constraints $g : \mathcal{S} \times \mathcal{U} \to \mathbb{R}^{n_g}$ are continuous functions.

MPC approximately solves (1) by minimizing the cost over a finite-horizon under an open-loop policy [2]. In every control step $t \in \mathbb{N}$, MPC solves the NLP

$$V_N^0(s_t) \coloneqq \min_{\mathbf{x},\, \mathbf{u}} \quad V_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k) \tag{2a}$$

$$\text{s.t.} \quad x_0 = s_t, \tag{2b}$$

$$x_{k+1} = f(x_k, u_k),\ k \in \mathbb{Z}_0^{N-1}, \tag{2c}$$

$$g(x_k, u_k) \leq 0, \qquad k \in \mathbb{Z}_0^{N-1}, \tag{2d}$$

with $\mathbb{Z}_a^b \coloneqq \{a, a+1, \dots, b\}$ and decision variables $\mathbf{x} \coloneqq (x_0, \dots, x_N)$, $\mathbf{u} \coloneqq (u_0, \dots, u_{N-1})$. The optimal solution of (2) is denoted by $\mathbf{x}^*(s_t), \mathbf{u}^*(s_t)$, the MPC policy is denoted by $\pi_{\mathrm{MPC}}(s_t)$ and corresponds to the value of the first optimal control $u_0^*(s_t)$.

Conversely, RL seeks to directly optimize a parameterized policy π_θ, typically represented by a neural network [3]. This is done through iterative updates based on observed costs and transitions, making it specifically suited for problems where myriads of transition samples can be obtained, e.g., via simulation. Both value-based and policy-gradient methods can be employed, as well as actor-critic architectures such as PPO [4].

Several approaches aim at reducing the online computational burden of MPC while retaining high performance. One direction consists in learning a policy offline, either via reinforcement learning or imitation of an expert controller, and deploying it directly at runtime [5], [6]. While this enables fast feedback evaluation, constraint satisfaction and performance guarantees are typically difficult to obtain [7]. A different option for fast yet well-performing MPC is to reduce the horizon while improving the terminal cost function V_f. Ideally, V_f should reflect the optimal cost-to-go

[1] Department of Microsystems Engineering (IMTEK), University of Freiburg, 79110 Freiburg, Germany

[2] Robotics and Perception Group, University of Zurich, Switzerland

[3] IMT School of Advanced Studies, Lucca, Italy

[4] Department of Mathematics, University of Freiburg

Correspondent: `andrea.ghezzi@imtek.uni-freiburg.de`

This research was supported by DFG via projects 504452366 (SPP 2364) and 525018088, by BMWK via 03EI4057A and 03EN3054B, and by the EU via ELO-X 953348. This work was also supported by the European Research Council (ERC), Advanced Research Grant COMPACT (Grant Agreement No. 101141351).

beyond the finite horizon, reducing the impact of the horizon truncation on policy performance. This approximation becomes increasingly important as the horizon length decreases. Approaches include offline RL to learn V_f [8], using RL to directly tune a parameterized and structured V_f [9], using supervised learning to construct a convex quadratic V_f based on optimal state-control trajectories obtained from the solution of long horizon MPC [10]. When a suboptimal policy is available, its rollout can be used to approximate the value function, an idea known as closed-loop costing (CLC) [11] or rollout of a base policy [3, §6.1.3].

Contribution: In contrast to approaches that directly deploy a learned controller or approximate the terminal value function offline, we follow a rollout-then-optimize principle. We use a learned policy to generate a state-control trajectory which serves as linearization point for constructing a convex quadratic program (QP). A single Newton step, implemented as a Riccati recursion within a real-time iteration (RTI) MPC scheme, is performed on this QP. A backtracking line-search ensures constraint satisfaction. Under standard assumptions, we show that the proposed controller is stabilizing and achieves lower closed-loop cost than the learned policy alone. We demonstrate its effectiveness on a constrained trajectory-tracking task for a quadrotor.

II. MPC Refinement of a Learned Policy Rollout

The proposed control scheme consists of two phases: an *offline* and an *online* phase.

Offline phase: In this phase, we train the policy π_θ in simulation. This requires access to a simulator of the system dynamics f and to a cost function $\hat{\ell}$ that reflects the objective of (2), possibly augmented to encode the system constraints.

Online phase – preparation: We adopt a sequential quadratic programming (SQP) approach to solve (2) [12]. To achieve real-time performance, we rely on the RTI scheme [13], which performs a single SQP iteration per sampling instant. Each iteration therefore requires one linearization of (2) and the solution of one QP. We introduce nonnegative slack variables $\sigma_k \in \mathbb{R}^{n_g}_{\geq 0}$ to rewrite inequality constraints in (2d) as equalities. The QP solved in each RTI step, linearized around a nominal trajectory $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$, reads

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{u},\boldsymbol{\sigma}} \quad & \sum_{k=0}^{N-1} \ell_k^{\mathrm{q}}(x_k, u_k) + \ell_N^{\mathrm{q}}(x_N) && \text{(3a)}\\
\text{s.t.} \quad & x_0 - s_t = 0, && \text{(3b)}\\
& x_{k+1} - A_k x_k - B_k u_k - c_k = 0,\ k \in \mathbb{Z}_0^{N-1}, && \text{(3c)}\\
& d_k + G_k^x x_k + G_k^u u_k + \sigma_k = 0,\ k \in \mathbb{Z}_0^{N-1}, && \text{(3d)}\\
& \sigma_k \geq 0,\ k \in \mathbb{Z}_0^{N-1}. && \text{(3e)}
\end{aligned}$$

Here the quadratic approximations of the stage and terminal costs are

$$\begin{aligned}
\ell_k^{\mathrm{q}}(w_k) &\coloneqq \ell(\bar{w}_k) + \nabla \ell(\bar{w}_k)^\top \delta w_k + \tfrac{1}{2} \delta w_k^\top H_k \delta w_k,\\
\ell_N^{\mathrm{q}}(x_N) &\coloneqq V_f(\bar{x}_N) + \nabla V_f(\bar{x}_N)^\top \delta x_N + \tfrac{1}{2} \delta x_N^\top H_N \delta x_N,
\end{aligned} \tag{4}$$

with $\delta w_k \coloneqq (\delta x_k, \delta u_k) \coloneqq (x_k - \bar{x}_k, u_k - \bar{u}_k)$. The Hessian matrices are defined as (omitting the linearization point for brevity)

$$H_N \coloneqq \nabla^2 V_f(\bar{x}_N),\ \begin{bmatrix} H_{xx} & H_{xu} \\ H_{xu}^\top & H_{uu} \end{bmatrix} \coloneqq H_k,\ k \in \mathbb{Z}_0^{N-1}. \tag{5}$$

In case the exact Hessian is used, $H_{xx} = \nabla^2_{xx} \mathcal{L}(\bar{w}_k, \bar{\lambda}_k, \bar{\mu}_k)$, $H_{xu} = \nabla^2_{xu} \mathcal{L}(\bar{w}_k, \bar{\lambda}_k, \bar{\mu}_k)$, $H_{uu} = \nabla^2_{uu} \mathcal{L}(\bar{w}_k, \bar{\lambda}_k, \bar{\mu}_k)$. Function $\mathcal{L}$ is the Lagrangian of (2), and $\bar{\lambda}_k \in \mathbb{R}^{n_x}, \bar{\mu}_k \in \mathbb{R}^{n_g}$ the multipliers associated with equality and inequality constraints, respectively. Hence, $\bar{\boldsymbol{\lambda}} \coloneqq (\bar{\lambda}_0, \ldots, \bar{\lambda}_N)$ and $\bar{\boldsymbol{\mu}} \coloneqq (\bar{\mu}_0, \ldots, \bar{\mu}_{N-1})$ are required and can be obtained from the previous closed-loop iteration (see Remark 1 below). The linearized dynamics and constraints in (3) use the quantities

$$\begin{aligned}
A_k &\coloneqq \tfrac{\partial f}{\partial x}(\bar{x}_k, \bar{u}_k),\ B_k \coloneqq \tfrac{\partial f}{\partial u}(\bar{x}_k, \bar{u}_k),\\
G_k^x &\coloneqq \tfrac{\partial g}{\partial x}(\bar{x}_k, \bar{u}_k),\ G_k^u \coloneqq \tfrac{\partial g}{\partial u}(\bar{x}_k, \bar{u}_k),
\end{aligned} \tag{6}$$

and

$$\begin{aligned}
c_k &\coloneqq f(\bar{x}_k, \bar{u}_k) - A_k \bar{x}_k - B_k \bar{u}_k,\\
d_k &\coloneqq g(\bar{x}_k, \bar{u}_k) - G_k^x \bar{x}_k - G_k^u \bar{u}_k.
\end{aligned} \tag{7}$$

Online phase – feedback: At each closed-loop step $t \in \mathbb{N}$, we perform a rollout $\bar{x}_{k+1} = f(\bar{x}_k, \pi_\theta(\bar{x}_k)), \bar{x}_0 = s_t, k \in \mathbb{Z}_0^{N-1}$, using the learned RL policy π_θ. We collect the state-control trajectory in the vectors $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ and use it as linearization point for constructing (3). The Karush-Kuhn-Tucker (KKT) system of (3) has to be solved. For a stage $k \in \mathbb{Z}_1^{N-1}$, the system reads

$$\begin{aligned}
H_{xx}\delta x_k + H_{xu}\delta u_k + A_k^\top \lambda_{k+1} - \lambda_k + G_k^{x\top} \mu_k &= 0, && \text{(8a)}\\
H_{uu}\delta u_k + H_{ux}\delta x_k + B_k^\top \lambda_{k+1} + G_k^{u\top} \mu_k &= 0, && \text{(8b)}\\
A_k \delta x_k + B_k \delta u_k - \delta x_{k+1} &= 0, && \text{(8c)}\\
G_k^x \delta x_k + G_k^u \delta u_k + \sigma_k &= 0, && \text{(8d)}\\
\sigma_k \geq 0,\ \mu_k \geq 0,\ \mathrm{diag}(\sigma_k)\mu_k &= \tau \mathbf{1}, && \text{(8e)}
\end{aligned}$$

where $\tau > 0$ and $\mathbf{1}$ is a vector of ones with appropriate dimension. We directly consider the relaxed complementarity condition (8e), as we employ an interior-point solver that avoids us to explicitly handle active-set changes. Optimality of (3) is obtained for $\tau \to 0$. Equation (8e) renders (8) a nonlinear system and may require multiple Newton steps for convergence. To limit online computation, we linearize (8e)

$$\mathrm{diag}(\sigma_k)\bar{\mu}_k + \mathrm{diag}(\bar{\sigma}_k)\mu_k = \tau \mathbf{1}, \tag{9}$$

where $\bar{\sigma}_k, \bar{\mu}_k$ denote slack and multiplier values from the previous closed-loop iteration. Thus, the system composed of equations (8a)-(8d) and (9) with $k \in \mathbb{Z}_0^N$ is linear and can be solved with a single Newton step. Finally, we need to perform a backtracking line-search to ensure $\sigma_k, \mu_k \geq 0, k \in \mathbb{Z}_0^{N-1}$.

Remark 1 (Dependency on dual variables): If the QP (3) is solved with exact Hessian, one would need access to the dual variables $\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}$ to construct the KKT system (8). However, the rollout with the RL policy provides only the primal variables (state–control trajectory) but not the multipliers. Since OCPs typically employ a least-squares stage cost, it is

common to adopt the Gauss-Newton Hessian approximation which does not require explicit dual information. A second issue is related to the linearized complementarity (9) which requires $\bar{\mu}_k, \bar{\sigma}_k, k \in \mathbb{Z}_0^{N-1}$. An automatic way to obtain these quantities is to solve the first QP(s_0) to optimality, then its primal-dual solution can be used in (9) for the consecutive closed-loop iterations. Hence, for $s_t, t \geq 1$, we can approximately solve the QP (3) by performing a single Newton step on equations (8a)-(8d) and (9) followed by a backtracking line-search.

Remark 2 (Block structure and Riccati recursion): The linearized KKT equations form a symmetric block-tridiagonal system due to the sequential dynamics (8c). This structure enables a forward-backward (Riccati-type) factorization [14], allowing high-performance solution of QPs, as implemented for instance in the open-source solver `HPIPM` [15].

III. Theoretical Properties

We next formalize the theoretical justification of the proposed method.

Assumption 1 (Continuity and compactness): The functions $f : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$, $\ell : \mathcal{S} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ and $V_f : \mathcal{S} \to \mathbb{R}_{\geq 0}$ are continuous, $f(0,0) = 0$, $\ell(0,0) = 0$, $V_f(0) = 0$. The set $\mathcal{S}$ is closed and $\mathcal{U}$ is compact. Moreover, f and ℓ are Lipschitz continuous on $\mathcal{S} \times \mathcal{U}$ with constants $L_f, L_\ell > 0$, and V_f is Lipschitz with constant $L_V > 0$.
Under Assumption 1 a solution of (2) exists [2, Prop. 2.4].

Assumption 2 (MPC policy is nominally robust): For every $s_0 \in \mathcal{S}$ and sufficiently small bounded disturbances e_t, w_t, the closed-loop system

$$s_{t+1} = f\big(s_t, \pi_{\mathrm{MPC}}(s_t) + e_t\big) + w_t, \quad t \in \mathbb{N},$$

is nominally robustly asymptotically stabilizing (NRAS) the origin $(s,u) = (0,0) \in \mathcal{S}$.
Sufficient conditions for Assumption 1 are given, e.g., in [2, Th. 2.19, Pr. 3.5], and rely on Lyapunov stability. Throughout the analysis we consider a deterministic setting ($e_t = w_t = 0$) and constant horizon N. As $N \to \infty$, the policy π^N_{MPC} converges pointwise to the infinite-horizon optimal policy π^* [2].

Now consider training a parameterized policy π_θ (e.g., by reinforcement learning). For an infinite number of simulations N_{s}, the policy π_θ converges pointwise to π^*. In practice, however, the number of training episodes, hyperparameter tuning, and early stopping are limited, so π_θ only approximates the MPC feedback family. We formalize this approximation in the following assumption.

Definition 1 (Family of shrinking-horizon MPC policies): For a finite horizon with N steps, the solution $(\mathbf{x}^*, \mathbf{u}^*)$ of (2) corresponds to the closed-loop system

$$s_{t+1} = f\big(s_t, \pi^{N-t}_{\mathrm{MPC},\tau}(s_t)\big), \quad t \in \mathbb{Z}_0^{N-1}, \tag{10}$$

where $\pi^{N-t}_{\mathrm{MPC},\tau} : \mathcal{S} \to \mathcal{U}$ denotes the optimal control law for the shrinking horizon MPC problem with $N-t$ remaining steps and with τ being the minimum value of the barrier parameter achieved by the interior-point method solving (8). For brevity, we denote the policy $\pi^N_{\mathrm{MPC},\tau}$ as π_{MPC}.

Assumption 3 (Approximation of shrinking-horizon MPC): Suppose there exist sets $\mathcal{S}_N \supseteq \mathcal{S}_{N-1} \supseteq \cdots \supseteq \mathcal{S}_0$ such that the origin $(s,u) = (0,0) \in \mathcal{S}_0$ and the closed-loop system under the learned policy is $f(s, \pi_\theta(s)) \in \mathcal{S}_{i-1}, \forall s \in \mathcal{S}_i$. Moreover, the learned policy $\pi_\theta : \mathcal{S} \to \mathcal{U}$ satisfies the uniform stage-wise error bound

$$\|\pi_\theta(s) - \pi^i_{\mathrm{MPC},\tau}(s)\| \leq \varepsilon_u, \quad \forall s \in \mathcal{S}_i, \tag{11}$$

for some $\varepsilon_u > 0$. In addition, the policy outputs satisfy the problem constraints, i.e., $\pi_\theta(s) \in \mathcal{U}$ and $g(s, \pi_\theta(s)) \leq 0$ for all $s \in \mathcal{S}$.

Thanks to Assumption 3, the behavior of the closed-loop system $s_{t+1} = f(s_t, \pi_\theta(s_t))$ is directly linked to that of the nominal MPC closed-loop $s_{t+1} = f(s_t, \pi_{\mathrm{MPC}}(s_t))$.

Corollary 1: Under Assumption 2, the system $s_{t+1} = f\big(s_t, \pi_{\mathrm{MPC}}(s_t) + e_t\big)$ remains NRAS for all bounded perturbations $\|e_t\| \leq \varepsilon_u$ and all $s_t \in \mathcal{S}_N$. Consequently, the closed-loop system $s_{t+1} = f(s_t, \pi_\theta(s_t))$ is asymptotically stable for sufficiently small ε_u.

Remark 3: With some additional assumption it is possible to extend the stability results to tracking problems with time-varying references [2, §2.4.5].

Remark 4 (Interpretation of the approximation bound): The scalar ε_u in (11) represents the worst-case deviation (supremum over all stages t and states s) between the learned and the MPC policies. It therefore provides a conservative measure of the approximation quality. In practice, ε_u depends on the training accuracy of π_θ, the smoothness of f and the cost functions, and on how much the control law $\pi^{N-t}_{\mathrm{MPC},\tau}$ vary with t for a given $\tau \geq 0$. If stage-wise error bounds ε_u^{N-t} are available, the subsequent state- and cost-error recursions can be written with time-varying terms, yielding tighter (less conservative) estimates of the trajectory deviation and cost difference.

We turn our attention to the numerical solution of (2) and consider the following standard assumptions.

Definition 2: We define by z the concatenation of primal and dual variables of (2). Then, $z \coloneqq (\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, where $\boldsymbol{\lambda} \in \mathbb{R}^{Nn_x}$ and $\boldsymbol{\mu} \in \mathbb{R}^{(N-1)n_g}$ are the multipliers associated with equality constraints (2b)-(2c) and inequality constraints (2d), respectively. We denote by z^* the optimal solution of (2). Let $R_\tau(z)$ denote the KKT residual of a nonlinear interior-point (IP) formulation, obtained by relaxing complementarity as $\mathrm{diag}(\mu)\, g(x,u) = \tau \mathbf{1}$ with $\tau > 0$.

Assumption 4 (MPC regularity for IP method): For all $s \in \mathcal{S}_N$, the optimal solution z^* of (2) satisfies the linear independence constraint qualification (LICQ) and the second-order sufficient conditions (SOSC). For τ sufficiently small, $z^*(\tau)$ denotes the solution of the perturbed system $R_\tau(z) = 0$, and the Jacobian $\nabla R_\tau(z^*(\tau))$ is nonsingular in a neighborhood of $z^*(\tau)$.

Assumption 5 (Lipschitz KKT Jacobian): There exists $\omega > 0$ such that for all z_1, z_2 in a neighborhood of $z^*(\tau)$,

$$\big\|\nabla R_\tau(z_1) - \nabla R_\tau(z_2)\big\| \leq \omega \, \|z_1 - z_2\|. \tag{12}$$

Lemma 1 (Quadratic contraction of exact Newton step): If the initial guess z^0 satisfies $\|z^0 - z^*(\tau)\| \leq \rho$ for a

sufficiently small radius ρ, the exact Newton step applied to the perturbed residual

$$z^1 = z^0 - \left[\nabla R_\tau(z^0)\right]^{-1} R_\tau(z^0), \tag{13}$$

obeys

$$\|z^1 - z^*(\tau)\| \le \tfrac{\omega}{2} \|z^0 - z^*(\tau)\|^2. \tag{14}$$

Proof 1: Standard local-quadratic result for Newton's method on a smooth perturbed KKT system applies (see e.g. [12, Th. 3.5 and §19]). For brevity, we omitted the dependence of z^0, z^1 on the barrier parameter $\tau > 0$ in the notation. The contraction result holds uniformly for small τ.

Theorem 1 (Quadratic improvement): Let V be the cost functional $V(\mathbf{x}, \mathbf{u}) = V_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)$ which is Lipschitz with constant L_J (consequence of Assumption 1). Let z^1 be the KKT vector obtained after a single Newton step starting from z^0 computed with the RL rollout

$$z^1 = z^0 - \left[\nabla R_\tau(z^0)\right]^{-1} R_\tau(z^0), \qquad \tau > 0 \text{ small.}$$

Then

$$\left\|V(\mathbf{x}^1, \mathbf{u}^1) - V(\mathbf{x}^*, \mathbf{u}^*)\right\| \le L_J \, C_N \, \varepsilon_u^2, \tag{15}$$

with $C_N > 0$ a constant depending only on the problem data and uniform for sufficiently small τ. Hence, the suboptimality of the RL-based trajectory is reduced quadratically with respect to the uniform control approximation error ε_u.

Proof 2: Let $\mathbf{s}^{\mathrm{MPC}}$ and $\mathbf{s}^{\mathrm{RL}}$ denote the trajectories with length N generated by rollouts of $\pi_{\mathrm{MPC},\tau}^{N-t}, t \in \mathbb{Z}_0^{N-1}$ and π_θ, respectively, starting from the same s_0. Define $e_k = s_k^{\mathrm{RL}} - s_k^{\mathrm{MPC}}$, $e_0 = 0$. By Lipschitz continuity of f and the uniform policy bound (11),

$$\|e_{k+1}\| \le L_f(\|e_k\| + \varepsilon_u), \qquad k \in \mathbb{Z}_0^{N-1}, \tag{16}$$

which yields the closed-form bound

$$\|e_k\| \le G_k \varepsilon_u, \quad G_k = \begin{cases} \dfrac{L_f^k - 1}{L_f - 1}, & L_f \neq 1, \\ k, & L_f = 1. \end{cases}$$

Hence, the state-error after N steps satisfies $\left\|\mathbf{s}^{\mathrm{RL}} - \mathbf{s}^{\mathrm{MPC}}\right\| \le G_N \varepsilon_u$. Because the dual variables depend Lipschitz-continuously on the primal ones (LICQ), there exists $c_d > 0$ such that

$$\|z^0 - z^*(\tau)\| \le c_d G_N \varepsilon_u, \tag{17}$$

where z^0 is the KKT vector corresponding to the RL rollout and $z^*(\tau)$ the optimal solution of (2). Invoking Lemma 1 (applied to the perturbed residual R_τ), one Newton step yields, uniformly for small τ,

$$\|z^1 - z^*(\tau)\| \le \tfrac{\omega}{2} c_d^2 G_N^2 \varepsilon_u^2 =: C_N \varepsilon_u^2. \tag{18}$$

The uniformity in τ ensures that the cost bound holds independently of the particular barrier value used in the IP formulation. Since the cost functional $V(\mathbf{x}, \mathbf{u}) = V_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)$ is Lipschitz on the feasible set with constant L_J,

$$\left\|V(\mathbf{x}^1, \mathbf{u}^1) - V(\mathbf{x}^*, \mathbf{u}^*)\right\| \le L_J \|z^1 - z^*(\tau)\| \le L_J C_N \varepsilon_u^2.$$

This shows that the suboptimality after one Newton correction is of order $\mathcal{O}(\varepsilon_u^2)$.

This result formalizes that, starting from the rollout of the learned policy, one Newton-step for the linearized MPC improves the cost of the resulting trajectory quadratically with respect to the policy approximation error ε_u.

Remark 5 (Contraction of Gauss-Newton Hessian): The Newton step in Lemma 1 assumes access to the exact Jacobian of the KKT residual, which requires the Lagrange multipliers. Similarly to what highlighted in Remark 1, in MPC formulation we can usually adopt the Gauss-Newton Hessian approximation which does not require dual information. When using this approximation, the resulting iteration no longer enjoys quadratic contraction: convergence becomes at best linear, with a convergence factor that depends on the magnitude of the neglected higher-order residual terms (see, e.g., [12, Ch. 10]). Nevertheless, this Gauss-Newton step often provides a sufficiently accurate correction in practice.

IV. Practical Extensions of the Basic Algorithm

We present extensions of the basic algorithm proposed in Sec. II that are useful in practice.

Soft constraints or barriers: To avoid possible solver failures due to numerical issues, modeling errors, or unknown disturbances, inequality constraints (2d) can be relaxed and penalized properly in the cost. Alternatively, inequalities (2d) can be formulated in the cost as logarithmic barriers as done in [16]. However, barrier functions are not Lipschitz continuous and would therefore invalidate the Lipschitz assumptions used in the theoretical analysis above.

Synthesizing terminal cost: The Riccati recursion performed on the KKT system with linearized complementarity conditions of (3) can be interpreted as a method for synthesizing approximate terminal costs. Indeed, the by-product of the Riccati recursion is a positive-semidefinite Hessian matrix and a gradient of the value function $V_N^0(s_t)$. The formulas of the Riccati recursion and corresponding value function are reported explicitly in [16]. This terminal cost interpretation of the Riccati recursion allows defining an MPC problem with two phases (or horizons) as in [11]. The first phase, $k \in \mathbb{Z}_0^{M-1}, M < N$, called the *control horizon*, corresponds to a standard MPC problem as in (2). The second phase, $k \in \mathbb{Z}_M^N$, called the *simulation horizon*, applies the proposed algorithm. In the second phase, we generate a rollout using π_θ starting from $\bar{x}_M$, construct a QP, and derive its KKT system with linearized complementarity. Then, a Riccati recursion is performed backward from N to M to solve the KKT system. The recursion yields a Hessian $H_M(\bar{x}_M) \succeq 0$ and a gradient $h_M(\bar{x}_M)$ which compose the quadratic terminal cost term used in the first-phase MPC problem.

Remark 6: Both the soft-constrained and the two-phase formulations can be useful in practice, especially when the RL policy is still being learned and its performance is not yet satisfactory. Extending the first (control) horizon and shrinking the second (simulation) horizon makes the overall

MPC problem approach closer to a standard single-phase formulation. Thanks to its limited online computation, the proposed approach is attractive for RL training frameworks employing MPC-in-the-loop, where thousands of MPC evaluations are typically needed [1].

Real-time implementation: In a RTI scheme it is customary to divide the computations into two phases, *preparation* and *feedback* [13]. During the preparation phase, the KKT matrix of the QP (3) can be factorized without requiring the latest state measurement. Once the new state is available, the *feedback* phase starts and the QP is solved to compute the control law. In the proposed algorithm, before assembling the QP, we perform the N-step rollout with the learned policy π_θ, which in principle should start from the current measured state s_t. However, performing this rollout only after obtaining s_t shifts all computations to the feedback phase and may introduce an undesirably large delay between the measurement of s_t and the application of the control $\pi_{\mathrm{MPC}}(s_t)$. A practical way to minimize feedback time is to compute the rollout in advance, starting from $x_1^*(s_{t-1})$, i.e., the one-step predicted optimal state in the previous closed-loop iteration.

V. Trajectory tracking with a quadcopter

We test the proposed scheme considering the control of a nano-quadcopter that has to track a lemniscate trajectory in the 3D space. The results are obtained via numerical simulations using the quadcopter environment of `safe-control-gym` [17]. The MPC problems described next are formulated and solved using `acados` [18] with `HPIPM` as QP solver [15]. All computations are performed on a MacBook with Intel(R) Core(TM) i7-8559U CPU @ 2.70GHz and 16 GB of memory running macOS 15.1.

A. Quadcopter modeling

The quadcopter state is defined as $s \coloneqq (\mathbf{p}, \mathbf{v}, \mathbf{\Psi}, \boldsymbol{\omega})$, where $\mathbf{p} \in \mathbb{R}^3$ is the position of the quadcopter center of mass (CoM) in the inertial frame, $\mathbf{v} \in \mathbb{R}^3$ is the linear velocity of the CoM in the world frame, $\mathbf{\Psi} \in \mathbb{R}^3$, is the orientation in the body frame expressed as roll, pitch, yaw, and $\boldsymbol{\omega} \in \mathbb{R}^3$ contains angular velocities in the body frame with respect to the inertial frame. The control of the quadcopter corresponds to the thrust of single propellers, $u \coloneqq (\tau_1, \tau_2, \tau_3, \tau_4) \in \mathbb{R}^4$. The dynamics of the drone are described by the ordinary differential equation (ODE)

$$\dot{s} = f(s,u) = \begin{pmatrix} \mathbf{v} \\ \frac{1}{m}\mathbf{R}_{\mathrm{bw}}\mathbf{e}_z \sum_{i=1}^{4} \tau_i - \mathbf{e}_z g \\ \mathbf{T}\boldsymbol{\omega} \\ J^{-1}(\mathbf{M}(\tau_1,\tau_2,\tau_3,\tau_4) - \boldsymbol{\omega} \times J\boldsymbol{\omega}) \end{pmatrix}, \quad (19)$$

where $\mathbf{R}_{\mathrm{bw}} \in \mathbb{R}^{3\times3}$ is the rotation matrix from the body frame to the world frame, $\mathbf{e}_z \in \mathbb{R}^3$ is the unit vector for the z-direction of the world frame, g is the gravity acceleration, $\mathbf{T} \in \mathbb{R}^{3\times3}$ is a matrix that maps angular velocity of the body frame into the time derivative of the orientation, $J \in \mathbb{R}^{3\times3}$ is the inertia matrix of the quadcopter, and $\mathbf{M} \in \mathbb{R}^3$ is a vector containing the angular momentum generated by the propellers. We adopt SI units. For more details and parameter values of the model see [17], [19].

B. MPC and RL policies

The MPC policy to accomplish the tracking task solves in closed-loop iteration $t \in \mathbb{N}$ the following NLP

$$\begin{aligned} \min_{\mathbf{x},\,\mathbf{u}} \quad & \sum_{k=0}^{N-1} \delta z_k^\top H_k \delta z_k + \delta z_N^\top H_N \delta z_N \\ \text{s.t.} \quad & x_0 - s_t = 0, \\ & x_{k+1} - F(x_k, x_k) = 0, \quad k \in \mathbb{Z}_{[0,N-1]}, \\ & u_{\mathrm{lb}} \le u_k \le u_{\mathrm{ub}},\, k \in \mathbb{Z}_{[0,N-1]}, \\ & x_{\mathrm{lb}} \le x_k \le x_{\mathrm{ub}},\, k \in \mathbb{Z}_{[0,N]}, \end{aligned} \quad (20)$$

where $N = 40$, $\delta z_k \coloneqq (x_k - x^o_{t+k}, u_k - u^o_{t+k})$, $\delta z_N \coloneqq (x_N - x^o_{t+N})$, and x^o_{t+k}, u^o_{t+k} are references to track. The quadratic cost is defined by the matrix $H_N \coloneqq Q$, and by the block diagonal matrix $H_k \coloneqq \mathrm{diag}(Q, R)$, with $Q = \mathrm{diag}(5,5,5,0.1\cdot\mathbf{1}_8)$, $R = \mathrm{diag}(0.1\cdot\mathbf{1}_4)$, and $\mathbf{1}_n$ is a vector of ones with dimension n. Function $F : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ corresponds to the time-discretization of the ODE (19) with an explicit Runge-Kutta integrator of order 4, and sampling time $t_{\mathrm{d}} = 0.02$ s. The position reference consists of a 3D lemniscate with a period of $T = 5$ s and frequency $\rho = \frac{2\pi}{T}$, its derivative constitutes the reference for the velocity. The lemniscate equations and numerical values for the bounds $u_{\mathrm{lb}}, x_{\mathrm{lb}}, u_{\mathrm{ub}}, x_{\mathrm{ub}}$ are available online[1]. For the orientation and angular velocity of the quadcopter we consider the reference to be zero. Note that the resulting reference might be dynamically infeasible for the quadcopter. As a reference for the controls, we consider the thrust required for hovering, $u^o_{t+k} = 0.06615 \cdot \mathbf{1}_4$ N $, k \in \mathbb{Z}_{[0,N-1]}$. Finally, we adopt a Gauss-Newton Hessian approximation.

Reinforcement-learning policy: We adopt the RL policy obtained via PPO that is available in `safe-control-gym` for the specified quadcopter model and tracking task. The policy input consists of the current state s_t and the state reference to track one-step forward, i.e., x^o_{t+1}. The RL policy has been trained by minimizing the stage cost of the MPC (20). A training episode of the RL policy is truncated in case of constraint violations and incurs a higher cost. Hence, we promote safer but potentially suboptimal trajectories. From the stochastic PPO policy parameterized as Gaussian, we consider the predicted mean value to obtain a deterministic policy, denoted here as PPO-RL. The PPO-RL policy is a fully-connected neural network with one hidden layer with 128 elements and $\tanh$ activation functions.

C. Comparison

We compare four policies: *RTI*, *Riccati-RTI*, *Riccati-RL*, and *PPO-RL*. These correspond respectively to solving (20) (i) with a standard RTI method, (ii) with an RTI method limited to a single iteration, (iii) with the proposed method

[1] `https://github.com/aghezz1/rl-riccati`

TABLE I

COMPARISON OF CLOSED-LOOP COST

Approach	Tracking Cost			Constraint Violation Cost		
	Sim. 1	Sim. 2	Sim. 3	Sim. 1	Sim. 2	Sim. 3
RTI	0.086	0.073	0.145	0	0	0
Riccati-RTI	0.095	0.083	0.179	0.027	0.012	0.033
Riccati-RL	0.101	0.087	0.177	0.022	0.016	0.022
PPO-RL	0.181	0.177	0.229	0.155	0.144	0.159

TABLE II

AVERAGE AND MAXIMUM RUNTIME FOR ONE CONTROL STEP

Approach	Initialization (ms)		Runtime (ms)		Feedback Time (ms)	
	Avg.	Max.	Avg.	Max.	Avg.	Max.
RTI	-	-	1.86	4.55	1.00	3.58
Riccati-RTI	-	-	1.00	1.66	0.17	0.47
Riccati-RL	11.56	15.09	1.11	2.27	0.19	0.31
PPO-RL	-	-	0.16	0.46	-	-

described in Sec. II, and (iv) directly with the trained PPO policy.

For the RTI controller, the quadratic programs (3) are solved by iterating on the smoothed nonlinear KKT system (8), starting from an initial barrier parameter $\tau_t = 1$ and driving it to $\tau_{\min} = 10^{-8}$ at the optimum. The Riccati-RTI and Riccati-RL controllers perform only one Newton step on the primal-dual system (8) with the linearized complementarity condition (9). Regarding the initial point of each Newton step, primal variables in Riccati-RTI are initialized using the previous solution, and for the very first step we use the tracking reference. Instead, for Riccati-RL, the primal variables are initialized via a rollout of the PPO policy. Dual variables are initialized using the previous solution for both approaches, the initial value of the barrier parameter is determined as $\tau_t = \|\text{diag}(\sigma_{t-1})\mu_{t-1}\|_1$.

Three closed-loop simulations with duration T are performed starting from a hovering condition and from three initial positions shifted along the lemniscate by ± 0.15 m with respect to its center. We compare tracking accuracy, constraint violations, and computational time. Figure 1 shows the time trajectories of positions and linear velocities along the x- and z-axes.

Table I reports the average closed-loop cost for each simulation. The cost consists of a tracking term, identical to the quadratic stage cost in (20), and a penalty for constraint violations, defined as $\mathcal{J}_{\text{v}} = \frac{1}{N_{\text{s}}} \sum_{i=1}^{N_{\text{s}}} \|\max(0, h(z))\|_1$, where $N_{\text{s}} = T/t_{\text{d}}$ and $h(z) \leq 0$ collects the state and input bounds. As expected, RTI achieves the lowest tracking cost in all scenarios, with no constraint violations, and thus serves as the performance reference. Restricting RTI to a single iteration (Riccati-RTI) degrades the performance, yielding higher tracking cost and some constraint violation cost. The purely learned PPO-RL controller incurs in about double the tracking cost compared to RTI, and it exhibits constraint violations, concentrated mainly on the linear velocity v^{x}. This is why we omitted plots of other states or controls. Finally, the proposed Riccati-RL scheme is able to recover almost the same performance of RTI, dramatically improving both tracking and violation cost compared to PPO-RL, confirming the benefit of the optimization-based refinement step.

Table II summarizes the computational performance in terms of initialization time ("Initialization"), total solver runtime ("Runtime"), and *feedback* time (see Sec. IVa). For the RTI-based schemes, "Runtime" is the sum of the *preparation* and *feedback* phases. The "Initialization" time accounts for performing the RL rollout and setting the resulting state-control trajectory as the initial guess for `HPIPM`. The *preparation* phase is identical for all optimization-based approaches since they assemble the same QP (3). Both Riccati-RTI and Riccati-RL exhibit a much shorter *feedback* time than full RTI because they perform only a single Riccati recursion. As anticipated, PPO-RL achieves the smallest overall runtime because it evaluates only a neural network. The measured initialization time of Riccati-RL is relatively large, mainly due to Python overhead in the implementation of the policy rollout. In practice, the rollout involves the evaluation of a small fully-connected neural network and the integration of the system dynamics with a simple explicit Runge-Kutta scheme over N steps. Therefore, its execution time is expected to decrease by at least one order of magnitude in compiled code. Despite this overhead, all methods remain well below the 20 ms sampling interval, guaranteeing realtime feasibility.

VI. CONCLUSIONS AND OUTLOOK

This work demonstrated that learned control policies can be effectively improved online through a single Riccati-based Newton correction executed within a real-time-iteration MPC framework. We established theoretical bounds on the approximation error between the learned and MPC policies and proved that a single correction step reduces the trajectory suboptimality quadratically with the learned-policy error magnitude. Simulations on a quadcopter tracking problem confirmed that the Riccati-RL controller significantly improves the learned policy, achieving performance close to RTI at lower computational cost. Future work will focus on a high-performance implementation of the policy rollout to minimize initialization overhead, and on experimental validation on real hardware to confirm the predicted computational advantages and closed-loop performance.

REFERENCES

[1] R. Reiter, J. Hoffmann, D. Reinhardt, F. Messerer, K. Baumgärtner, S. Sawant, J. Boedecker, M. Diehl, and S. Gros, "Synthesis of model predictive control and reinforcement learning: Survey and classification," *Annual Reviews in Control*, vol. 61, p. 101045, 2026.

[2] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Nob Hill, 2017.

[3] D. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] M. Hertneck, J. Köhler, S. Trimpe, and F. Allgöwer, "Learning an approximate model predictive controller with guarantees," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 543–548, 2018.

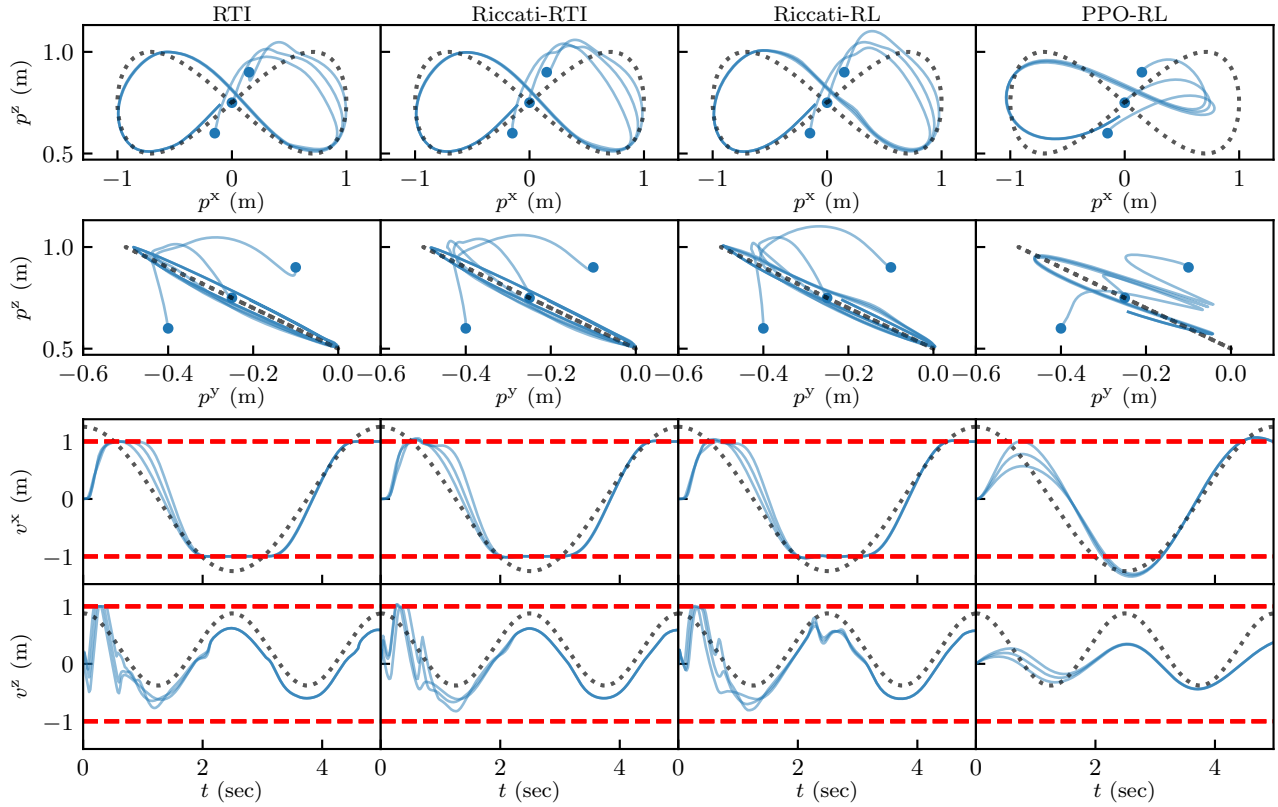

Fig. 1. Comparison of trajectory tracking performance for four control strategies: RTI, Riccati-RTI, Riccati-RL (*the proposed method*), and PPO-RL. The first two rows show the position trajectories in the space p^x, p^z and p^y, p^z, respectively. The blue dots correspond to the initial position of each simulation. The bottom two rows present the time evolution of velocity v^x and v^z, respectively. The black dashed lines indicate the reference trajectories, and the red dashed lines denote constraint limits.

[6] A. Ghezzi, J. Hoffman, J. Frey, J. Boedecker, and M. Diehl, "Imitation learning from nonlinear MPC via the exact Q-loss and its Gauss-Newton approximation," *Proc. IEEE Conf. Decis. Control (CDC)*, 2023.

[7] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, and A. Knoll, "A review of safe reinforcement learning: Methods, theories and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[8] R. Reiter, A. Ghezzi, K. Baumgärtner, J. Hoffmann, R. D. McAllister, and M. Diehl, "Ac4mpc: Actor-critic reinforcement learning for guiding model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 34, no. 1, pp. 395–410, 2026.

[9] K. Seel, A. B. Kordabad, S. Gros, and J. T. Gravdahl, "Convex neural network-based cost modifications for learning model predictive control," *IEEE Open Journal of Control Systems*, vol. 1, pp. 366–379, 2022.

[10] S. Abdufattokhov, M. Zanon, and A. Bemporad, "Learning lyapunov terminal costs from data for complexity reduction in nonlinear model predictive control," *International Journal of Robust and Nonlinear Control*, vol. 34, no. 13, pp. 8676–8691, 2024.

[11] G. De Nicolao, L. Magni, and R. Scattolini, "Stabilizing Receding-Horizon control of nonlinear time varying systems," *IEEE Trans. Automatic Control*, vol. AC-43, no. 7, pp. 1030–1036, 1998.

[12] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., ser. Operations Research and Financial Eng. Springer-Verlag, 2006.

[13] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM J. Control Optim.*, vol. 43, no. 5, pp. 1714–1736, 2005.

[14] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of Optimization Theory and Applications*, vol. 99, pp. 723–757, 1998.

[15] G. Frison and M. Diehl, "HPIPM: a high-performance quadratic programming framework for model predictive control," in *Proc. IFAC World Congr.*, Berlin, Germany, July 2020.

[16] A. Zanelli, R. Quirynen, G. Frison, and M. Diehl, "A partially tightened real-time iteration scheme for nonlinear model predictive control," in *Proc. IEEE Conf. Decis. Control (CDC)*, December 2017.

[17] Z. Yuan, A. W. Hall, S. Zhou, L. Brunke, M. Greeff, J. Panerati, and A. P. Schoellig, "Safe-control-gym: A unified benchmark suite for safe learning-based control and reinforcement learning in robotics," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 142–11 149, 2022.

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

[19] C. Luis and J. L. Ny, "Design of a trajectory tracking controller for a nanoquadcopter," *arXiv preprint arXiv:1608.05786*, 2016.