Learning-based Nonlinear Model Predictive Control Using Deterministic Actor-Critic with Gradient Q-learning Critic

Amine Salaje, Thomas Chevet, Nicolas Langlois

Learning-based Nonlinear Model Predictive Control Using Deterministic Actor-Critic with Gradient Q-learning Critic

Amine Salaje, Thomas Chevet, and Nicolas Langlois

***Abstract*— In this paper, we present an off-policy reinforcement learning (RL) method used to tune the optimal weights of a nonlinear model predictive control (NMPC) scheme. The objective is to find the optimal policy minimizing the closed-loop performance of point stabilization with obstacle avoidance control task. The parameterized NMPC scheme serves to approximate the optimal policy and update the parameters via compatible off-policy deterministic actor-critic with gradient Q-learning critic (COPDAC-GQ). While efficient, this algorithm requires a heavy computational complexity when combined with NMPC, as two optimal control problems have to be solved at each time instant. We therefore propose two different methods to reduce the real-time computational cost of the algorithm. First, a neural network is used to learn the subsequent state-action features of the advantage function. Then, we propose to use the information delivered by the NMPC scheme to approximate the subsequent state-action features in the critic. Whichever method is used removes the need of a secondary NMPC, significantly improving the training speed. The results show that there is no difference between the original method and the proposed methods in terms of the learned policy and the control performance, whereas the real-time computational burden is almost halved with the proposed methods.**

I. Introduction

When addressing complex control objectives, nonlinear model predictive control (NMPC) has proven to be an efficient control method [1]. Its capability in managing nonlinear systems and constraints renders it well-suited for numerous applications, one of them being mobile robotics [2]. NMPC generates control signals at each control time step by solving a so-called optimal control problem (OCP), i.e., a constrained nonlinear optimization problem. The OCP formulation hinges on the selection of parameters for which reliable tuning methods are highly sought [3]. A way to address this issue is to consider reinforcement learning (RL).

RL has gained widespread popularity due to its remarkable ability to achieve optimal performance across a multitude of fields as varied as games and the control of robotic systems [4]–[6]. RL addresses problems where an agent interacts with an environment, learns from its experience, and aims to make the best decisions over time by learning optimal value functions and optimal policies underlying a Markov decision process (MDP) [7].

*This work was supported by ANR and Région Normandie through the HAISCoDe (ANR-20-THIA-0021) and PAMAP projects.

The authors are with Université de Rouen Normandie, ESIGELEC, IRSEEM, 76000 Rouen, France amine.salaje@groupe-esigelec.org, {thomas.chevet, nicolas.langlois}@esigelec.fr

The deterministic policy gradient (DPG) [8] is an RL method seeking to estimate the optimal policy through a parameterized function approximator. It achieves this by directly optimizing the policy parameters via gradient descent steps of the RL performance [7]. When dealing with problems with large state spaces in high dimensions, modern RL methods aim to estimate the policy function using, among others, deep neural networks (DNN) as nonlinear function approximators [9]. However, DNN-based RL often lacks capabilities in closed-loop stability analysis, and can be difficult to formally analyze [10], leading to lingering critical concerns in effectively managing constraints and ensuring safety when controlling systems. To address these limitations, the notion of using model predictive control (MPC)-based RL has been proposed and justified in [10]. This approach advocates for employing MPC as the function approximator for the optimal policy in RL. Unlike DNNs, MPC-based policies inherently satisfy state/input constraints and safety requirements due to their construction.

Recently, numerous works have been using DPG to improve the closed-loop performance of MPC. In [11], a least squares temporal difference-based deterministic policy gradient method is used to learn the MPC parameters for a simplified freight mission of autonomous surface vehicles. The same RL optimization method has been used in battery storage applications [12], [13], as well as in polytopic LPV systems [14], and for peak power management within smart grids in [15].

Another approach consists in using compatible deterministic actor-critic with gradient Q-learning critic (COPDAC-GQ) [8]. With this technique, the gradient Q-learning is used in the critic [16]. Consequently, the critic parameters are updated towards the true gradient descent and convergence is thus *ensured* [17]. To the best of our knowledge, the only work where an MPC-based COPDAC-GQ approach has been implemented was in [18] for the application to home energy management systems. One of the drawbacks of this approach is the significant computational demand for real-time implementation. Indeed, in the context of updating the critic parameters, it requires to solve two OCPs at each sampling time, one for estimating the current state-action features function and another for approximating the subsequent state-action features in the compatible critic function approximation.

The main objective of this paper is to reduce the real time computational complexity of the algorithm without affecting the learned policy and the control performance. To do so, we propose two different approaches:

(i) We use a neural network (NN) to learn the state-action features. The NN is then trained to solve a regression problem using the states and the NMPC parameterized policy as input, whereas the output is the current state-action features, obtained at previous time instants.
(ii) We approximate the state-action features using the result of the OCP since, according to [10], the NMPC scheme can be used as an action-value function approximator.

In both cases, only one OCP has to be solved at each time instant, thus reducing the computational burden of the learning task.

The remainder of this paper is organized as follows. Section II recalls some background on predictive control and reinforcement learning. Then, Section III presents the algorithm on which our work is based, while Section IV details our contributions. Section V showcases the efficiency of our methods on numerical simulation examples. Finally, Section VI draws concluding remarks.

II. Background

This section aims to provide some background on the RL methods and the control algorithms we use. We start by giving some elements on Markov decision processes before presenting how a model predictive controller is parameterized.

A. Markov decision process

Consider a finite Markov decision process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}, \ell, \rho, \gamma)$ where $\mathcal{S} \subseteq \mathbb{R}^n$ is the state space, $\mathcal{A} \subseteq \mathbb{R}^m$ the action space, $\mathbb{P}$ the probability law underlying the state transition dynamics so that the state $\mathbf{s}$ at time $k+1$ depends on the state $\mathbf{s}_k$ and action $\mathbf{a}_k$ at time k following

$$\mathbf{s}_{k+1} \sim \mathbb{P}(\cdot|\mathbf{a}_k, \mathbf{s}_k). \tag{1}$$

Depending on the context, this transition can be either stochastic or deterministic. A more control oriented representation of the state transition would be

$$\mathbf{s}_{k+1} = f(\mathbf{a}_k, \mathbf{s}_k) \tag{2}$$

where f is a deterministic case of (1). The other elements of the tuple describing the MDP are the cost function ℓ associated with a state-action pair, the initial state distribution ρ, and the discount factor $\gamma \in (0, 1)$. This discount factor determines the importance of future rewards.

The primary goal of RL is to find a policy, denoted by $\boldsymbol{\pi} \colon \mathcal{S} \to \mathcal{A}$, that effectively minimizes a closed-loop performance objective [7]. Given an objective $J(\boldsymbol{\pi})$ on the expected cumulative cost, the RL problem aims to find the optimal policy $\boldsymbol{\pi}^*$ satisfying $\boldsymbol{\pi}^\star = \arg\min_\pi J(\boldsymbol{\pi})$ where

$$J(\boldsymbol{\pi}) = \mathbb{E}_{\mathbf{s}\sim\rho^\pi}\left[\sum_{k=0}^{K} \gamma^k \ell\left(\mathbf{s}_k, \mathbf{a}_k\right) \middle| \mathbf{a} = \boldsymbol{\pi}(\mathbf{s})\right] \tag{3}$$

where $K \in \mathbb{N} \cup \{+\infty\}$ is a (possibly infinite) horizon, ρ^π is the state distribution resulting from policy $\boldsymbol{\pi}$ and $\mathbb{E}_{\mathbf{s}\sim\rho^\pi}$ denotes the expected value computed over this state distribution.

B. Parameterized NMPC-Based Policy Approximation

A solution to a finite horizon MDP can be provided by a nonlinear model predictive controller under certain assumptions, as elaborated in [10]. Therefore, we consider the following parameterized NMPC scheme

$$\min_{\substack{\mathbf{u}_i, \mathbf{x}_i, \\ \forall i \in \overline{0,N}}} \quad \sum_{i=0}^{N-1} \gamma^i L\left(\mathbf{x}_i, \mathbf{u}_i, \boldsymbol{\theta}_X, \boldsymbol{\theta}_U\right) + \gamma^N W(\mathbf{x}_N, \boldsymbol{\theta}_f) \tag{4a}$$

$$\text{s.t.} \quad \mathbf{x}_0 = \mathbf{s}_k, \tag{4b}$$

$$\mathbf{x}_{i+1} = f(\mathbf{u}_i, \mathbf{x}_i), \quad \forall i \in \overline{1,N}, \tag{4c}$$

$$\mathbf{x}_i \in \mathcal{S}, \quad \forall i \in \overline{1,N}, \tag{4d}$$

$$\mathbf{u}_i \in \mathcal{A}, \quad \forall i \in \overline{0,N-1}, \tag{4e}$$

$$g(\mathbf{x}_i) + \theta_c \leq 0, \quad \forall i \in \overline{1,N}, \tag{4f}$$

where L is called the stage cost, W the terminal cost, $N \in \mathbb{N}$ the prediction horizon that may be shorter than K from the performance measure (3), and γ is as in (3). The deterministic dynamic model driven by the NMPC scheme (4) is denoted by f as in (2). The function g is used to impose constraints on the state at each time step over the prediction horizon N. Finally, $\boldsymbol{\theta}_X$, $\boldsymbol{\theta}_U$, $\boldsymbol{\theta}_f$, and θ_c are vectors of real parameters, usually chosen *a priori* based, e.g., on the dynamics f or the control objective for the system. However, in this paper, they are tuned through an RL algorithm.

In order to enhance the closed-loop performance of the NMPC scheme, as evaluated by the RL cost, it can be advantageous to parameterize the NMPC cost function, model, and constraints. RL then adjusts these parameters in a way to improve the closed-loop performance. Theoretically, according to [10, Theorem 1], we know that, under certain assumptions, and if the parameterization is rich enough, the MPC scheme is capable of capturing the optimal policy $\boldsymbol{\pi}^\star$.

When solving (4) at time $k \in \mathbb{N}$, we obtain the optimal control sequence $\mathbf{u}^\star(\mathbf{s}_k, \boldsymbol{\theta}) = \{\mathbf{u}_0^\star(\mathbf{s}_k, \boldsymbol{\theta}), \ldots, \mathbf{u}_{N-1}^\star(\mathbf{s}_k, \boldsymbol{\theta})\}$, with $\boldsymbol{\theta} = \begin{bmatrix}\boldsymbol{\theta}_X & \boldsymbol{\theta}_U & \boldsymbol{\theta}_f & \theta_c\end{bmatrix}$, driving system (2) towards its objective over the prediction horizon N. We define, for all $k \in \mathbb{N}$, the parameterized deterministic policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}_k)$ and the action $\mathbf{a}_k$ as

$$\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}_k) = \mathbf{u}_0^\star(\mathbf{s}_k, \boldsymbol{\theta}), \tag{5}$$

$$\mathbf{a}_k = \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}_k) + c_\rho^k \boldsymbol{\rho} \tag{6}$$

where $\boldsymbol{\rho}$ is a Gaussian term introducing weighted exploration of the action space $\mathcal{A}$ decreasing exponentially during training because of the coefficient $c_\rho \in (0, 1)$. In the learning process, we apply $\mathbf{a}_k$ given by (6) to system (2) at time k to get $\mathbf{s}_{k+1}$.

III. Compatible deterministic actor-critic with gradient Q-learning

In this section we present the original COPDAC-GQ algorithm as presented in [8]. To simplify the notations, the time dependence is dropped for states $\mathbf{s} \in \mathcal{S}$ and actions $\mathbf{a} \in \mathcal{A}$. The state at the next time instant is denoted $\mathbf{s}^+ \in \mathcal{S}$.

The policy parameters $\boldsymbol{\theta}$ can be directly optimized by gradient descent on the performance $J(\boldsymbol{\pi}_{\boldsymbol{\theta}})$ given in (3). The

update rule is then

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\pi}_{\boldsymbol{\theta}}) \tag{7}$$

where $\alpha < 0$ is the step size. Based on the DPG theorem proposed in [8], the policy gradient equation is

$$\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\pi}_{\boldsymbol{\theta}}\right) = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}) \nabla_{\mathbf{a}} Q_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a}) | \mathbf{a} = \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})\right] \tag{8}$$

where $Q_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a})$ is the action-value function associated to the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$. The sensitivity analysis of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\pi}_{\boldsymbol{\theta}})$ and $\nabla_{\mathbf{a}} Q_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a})$ is discussed in the following.

A. Gradient of the policy function

This paragraph describes how to compute $\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})$ appearing in the policy gradient equation (8). This method is inspired by the work of [10]. Let $\mathcal{L}_\theta$, the Lagrange function associated to the NMPC scheme (4), be

$$\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{z}) = \Omega_{\boldsymbol{\theta}} + \boldsymbol{\mu}^\top \mathbf{H}_{\boldsymbol{\theta}}, \tag{9}$$

where $\Omega_{\boldsymbol{\theta}}$ is the parameterized NMPC cost function (4a) and $\mathbf{H}_{\boldsymbol{\theta}}$ is a vector stacking vertically the inequality constraints appearing in problem (4). The real vector $\boldsymbol{\mu}$ is the Lagrange multiplier associated to the inequality constraint, having the size of $\mathbf{H}_{\boldsymbol{\theta}}$.

Let $\boldsymbol{\zeta} = \{\mathbf{x}, \mathbf{u}\}$ and $\mathbf{z} = \{\boldsymbol{\zeta}, \boldsymbol{\mu}\}$ be the primal variables and the primal-dual variables of the NMPC (4), respectively. Consequently, the sensitivity of the policy with respect to the policy parameters, can be obtained by using the implicit function theorem on the Karush-Kuhn-Tucker conditions written as

$$\frac{\partial \mathbf{z}^\star}{\partial \boldsymbol{\theta}} = -\frac{\partial \boldsymbol{\kappa}_{\boldsymbol{\theta}}}{\partial \mathbf{z}}^{-1} \frac{\partial \boldsymbol{\kappa}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$$

where $\mathbf{z}^\star$ is the primal-dual solution of problem (4), and

$$\boldsymbol{\kappa}_\theta = \begin{bmatrix} \nabla_{\boldsymbol{\zeta}} \mathcal{L}_{\boldsymbol{\theta}} \\ \operatorname{diag}(\boldsymbol{\mu}) \mathbf{H}_{\boldsymbol{\theta}} \end{bmatrix}. \tag{10}$$

Finally, $\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})$ can be extracted from the columns of $\partial \mathbf{z}^\star / \partial \boldsymbol{\theta}$.

B. Gradient of the action-value function

Following the conditions in [8], we can substitute the true action-value function $Q_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a})$ by a class of *compatible* function approximators $Q_{\mathbf{w}}(\mathbf{s}, \mathbf{a})$. Such approximators satisfy $Q_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) \approx Q_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a})$ without affecting the deterministic policy gradient in (8). The compatible function is

$$Q_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) = \underbrace{\left(\mathbf{a} - \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})\right)^\top \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})^\top}_{\boldsymbol{\Psi}(\mathbf{s}, \mathbf{a})^\top} \mathbf{w} + V_{\mathbf{v}}(\mathbf{s}), \tag{11}$$

where $\boldsymbol{\Psi}(\mathbf{s}, \mathbf{a})$ is the state-action features vector weighted by $\mathbf{w}$, and $V_{\mathbf{v}}(\mathbf{s})$ is the baseline value function approximating the true value function $V_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(\mathbf{s})$, taking the form

$$V_{\mathbf{v}}(\mathbf{s}) = \mathbf{v}^\top \boldsymbol{\phi}(\mathbf{s}), \tag{12}$$

where the state features $\boldsymbol{\phi}(\mathbf{s})$ is designed in this paper to constitute all monomials of the state with degrees less than or equal to 2 and $\mathbf{v}$ is the corresponding weight vector. Finally, $\nabla_{\mathbf{a}} Q_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a})$ appearing in (8) is computed as

$$\nabla_{\mathbf{a}} Q_{\boldsymbol{\pi}_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a}) \approx \nabla_{\mathbf{a}} Q_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})^\top \mathbf{w}. \tag{13}$$

C. Parameter update

To update the parameters $\boldsymbol{\theta}$ of the policy and $\mathbf{v}$ and $\mathbf{w}$ of the action-value function, we use the COPDAC-GQ optimization method [8]. The critic updates the action-value and value functions parameters $\mathbf{v}$ and $\mathbf{w}$ using the gradient Q-learning technique [16], thus ensuring the critic's convergence towards the true gradient descent [17]. Let δ, the temporal difference (TD) error, be

$$\delta = \ell(\mathbf{s}, \mathbf{a}) + \gamma Q_{\mathbf{w}}\left(\mathbf{s}^+, \boldsymbol{\pi}_{\boldsymbol{\theta}}\left(\mathbf{s}^+\right)\right) - Q_{\mathbf{w}}(\mathbf{s}, \mathbf{a}). \tag{14}$$

The action-value and value function approximators $Q_{\mathbf{w}}$ and $V_{\mathbf{v}}$ are updated using gradient Q-learning. We also define a set of weights $\boldsymbol{\beta}$ using the weight-doubling trick [17] so that

$$\begin{aligned} \mathbf{w} \leftarrow \mathbf{w} &+ \alpha_{\mathbf{w}} \delta \boldsymbol{\Psi}(\mathbf{s}, \mathbf{a}) \\ &- \alpha_{\mathbf{w}} \gamma \boldsymbol{\Psi}\left(\mathbf{s}^+, \boldsymbol{\pi}_{\boldsymbol{\theta}}\left(\mathbf{s}^+\right)\right)\left(\boldsymbol{\Psi}(\mathbf{s}, \mathbf{a})^\top \boldsymbol{\beta}\right) \end{aligned} \tag{15a}$$

$$\mathbf{v} \leftarrow \mathbf{v} + \alpha_{\mathbf{v}} \delta \boldsymbol{\phi}(\mathbf{s}) - \alpha_{\mathbf{v}} \gamma \boldsymbol{\phi}\left(\mathbf{s}^+\right)\left(\boldsymbol{\phi}(\mathbf{s}, \mathbf{a})^\top \boldsymbol{\beta}\right) \tag{15b}$$

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha_{\boldsymbol{\beta}}\left(\delta - \boldsymbol{\phi}(\mathbf{s}, \mathbf{a})^\top \boldsymbol{\beta}\right) \boldsymbol{\phi}(\mathbf{s}, \mathbf{a}), \tag{15c}$$

where $\alpha_{\mathbf{w}}$, $\alpha_{\mathbf{v}}$, and $\alpha_{\boldsymbol{\beta}}$ are the critic's learning rates.

Then, the actor updates the policy function's parameters $\boldsymbol{\theta}$ following the rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}) \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})^\top \mathbf{w}, \tag{16}$$

where $\alpha_{\boldsymbol{\theta}}$ is the actor's learning rate.

IV. Proposed methods

This section presents the main contributions of this paper, aiming to reduce the real-time complexity of the algorithm we described above.

A. Subsequent state-action features approximation with neural network

In the calculation of the TD error in (14), the subsequent action-value function $Q_{\mathbf{w}}(\mathbf{s}^+, \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+))$ can be expressed as

$$\begin{aligned} &Q_{\mathbf{w}}(\mathbf{s}^+, \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)) = \\ &\qquad \underbrace{\left(\mathbf{a} - \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)\right)^\top \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)^\top}_{\boldsymbol{\Psi}(\mathbf{s}^+, \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+))^\top} \mathbf{w} + V_{\mathbf{v}}(\mathbf{s}^+), \end{aligned} \tag{17}$$

where $\mathbf{a}$ remains the same action given by (6), the target policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)$ is, according to (5),

$$\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+) = \mathbf{u}_0^\star(\mathbf{s}^+, \boldsymbol{\theta}). \tag{18}$$

It follows that obtaining $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)$ and $\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)$ requires solving a second NMPC problem (4) at the next state $\mathbf{s}^+$. However, solving two constrained nonlinear optimization problems at each time step makes the real-time complexity higher, leading to a longer training time, specifically when the length of the prediction horizon N is larger or the state space has a high dimension.

Our first contribution consists in approximating the state-action features $\boldsymbol{\Psi}(\mathbf{s}, \mathbf{a})$ using a neural network in order to omit the need of solving a second NMPC problem. To train the NN, we store, at each learning step, the tuple $\{\mathbf{s}, \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}), \boldsymbol{\Psi}(\mathbf{s}, \mathbf{a})\}$

into a replay buffer $\mathcal{D}$. The current state $\mathbf{s}$ and the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})$ are the NN input while $\boldsymbol{\Psi}(\mathbf{s},\mathbf{a})$ is the regression target.

Then, a mini-batch $\mathcal{B} \subset \mathcal{D}$ of a selected size $\mathrm{card}(\mathcal{B})$ is randomly sampled and we update the NN parameters $\boldsymbol{\omega}$ using the Adam algorithm [19] with a learning rate $\alpha_{\boldsymbol{\omega}}$ to minimize the cost function

$$C_{\mathrm{NN}}(\boldsymbol{\omega}) = \frac{1}{\mathrm{card}(\mathcal{B})} \sum_{\{\boldsymbol{\varsigma},\boldsymbol{\varpi},\boldsymbol{\psi}\}\in\mathcal{B}} \left(\boldsymbol{\Psi}_{\boldsymbol{\omega}}(\boldsymbol{\varsigma},\boldsymbol{\varpi}) - \boldsymbol{\psi}\right)^2, \quad (19)$$

where $\boldsymbol{\Psi}_{\boldsymbol{\omega}}(\boldsymbol{\varsigma},\boldsymbol{\varpi})$ for $\{\boldsymbol{\varsigma},\boldsymbol{\varpi},\boldsymbol{\psi}\} \in \mathcal{B}$ is the output of the NN with parameters $\boldsymbol{\omega}$ and inputs $\boldsymbol{\varsigma}$ and $\boldsymbol{\varpi}$, whereas $\boldsymbol{\psi}$ is the regression target.

After updating the neural network, the subsequent state-action features $\boldsymbol{\Psi}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+))$ in (17) can be estimated by passing the next state $\mathbf{s}^+$ and the new target policy to the neural network, where it is approximated by

$$\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+) = \mathbf{u}_1^\star(\mathbf{s},\boldsymbol{\theta}) \quad (20)$$

where $\mathbf{u}_1^\star(\mathbf{s},\boldsymbol{\theta})$ is the second element of the optimal control sequence when solving the primary NMPC (4). It follows that only one constrained optimization problem is solved at each time step along the NN optimization. Consequently, the subsequent action-value function in (17) can be rewritten as

$$Q_{\mathbf{w}}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)) = \boldsymbol{\Psi}_{\boldsymbol{\omega}}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+))^\top \mathbf{w} + V_{\mathbf{v}}(\mathbf{s}^+). \quad (21)$$

B. Subsequent state-action features approximation with NMPC scheme

In addition to the parameterization of the policy given by (4), the NMPC scheme can also deliver the optimal action-value function under some assumptions given in [10]. We then have

$$Q_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) = (4). \quad (22)$$

Following this observation, we can assume that the subsequent action-value function in the TD error (14) can be estimated using information delivered by (22) instead of using a neural network as in the previous paragraph. Particularly, our second contribution consists in approximating $\boldsymbol{\Psi}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+))$ using

$$\boldsymbol{\Psi}_{\mathrm{MPC}}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)) = Q_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) + \nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}), \quad (23)$$

which provides useful features to construct the subsequent state-action features function. As proved in [10], the sensitivity $\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})$ is obtained as

$$\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{z}), \quad (24)$$

where $\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{z})$ is as defined in (9). Then, the subsequent action-value function in (17) can be rewritten

$$Q_{\mathbf{w}}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+)) = \boldsymbol{\Psi}_{\mathrm{MPC}}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+))^\top \mathbf{w} + V_{\mathbf{v}}(\mathbf{s}^+). \quad (25)$$

As in the previous paragraph, we only solve one NMPC problem (4) at each time step, therefore reducing the overall computation time.

The COPDAC-GQ-based scheme to tune the parameters of a nonlinear model predictive controller is summarized in Algorithm 1.

Algorithm 1: COPDAC-GQ-based NMPC parameter tuning.

Input: $\boldsymbol{\theta}$, $\mathbf{w}$, $\mathbf{v}$, $\boldsymbol{\beta}$, dynamical system f, an NMPC (4)
Output: $\boldsymbol{\theta}^\star$
Initialization: $\mathcal{D} \leftarrow \emptyset$, an initial state $\mathbf{s}_0 \in \mathcal{S}$
for $k \leftarrow 1$ **to** *selected number of episodes* N_{ep} **do**
  $\mathbf{s} \leftarrow \mathbf{s}_0$
  for $t \leftarrow 0$ **to** K **do**
    Solve (4) to obtain $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s})$
    Get the action $\mathbf{a}$ from (6)
    Observe $\mathbf{s}^+ = f(\mathbf{a},\mathbf{s})$ and get the reward ℓ
    if *method from paragraph IV-A used* **then**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{s},\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}),\boldsymbol{\Psi}(\mathbf{s},\mathbf{a})\}$
      if $\mathrm{card}(\mathcal{D}) \geq \mathrm{card}(\mathcal{B})$ **then**
        Randomly sample a mini-batch of $\mathrm{card}(\mathcal{B})$ elements of $\mathcal{D}$
        Get $\boldsymbol{\omega}$ from (19)
        Compute δ with (14) using the modified $Q_{\mathbf{w}}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+))$ from (21)
    else if *method from paragraph IV-B used* **then**
      Compute δ with (14) using the modified $Q_{\mathbf{w}}(\mathbf{s}^+,\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s}^+))$ from (25)
    Update the critic parameters $\mathbf{w}$,$\mathbf{v}$, $\boldsymbol{\beta}$ by (15)
    Update the actor parameters $\boldsymbol{\theta}$ by (16)
    $\mathbf{s} \leftarrow \mathbf{s}^+$

V. Numerical simulations

In this section, we apply in simulation the proposed COPDAC-GQ-based NMPC to solve setpoint tracking problems with static obstacle avoidance for a differential drive mobile robot. These simulations show the clear improvement in training the control scheme compared with [18].

A. Simulation Model

For simulation purposes, we consider a differential drive robot having for continuous-time dynamics [20]

$$\dot{x} = v\cos\varphi \quad (26a)$$

$$\dot{y} = v\sin\varphi \quad (26b)$$

$$\dot{\varphi} = \nu \quad (26c)$$

where (x,y) denote the robot's coordinates in a two-dimensional plane, φ is its orientation in the plane, v its linear speed, and ν its rotation speed. These dynamics are discretized at a sampling period $T_s = 0.2\,\mathrm{s}$ using the Runge-Kunta 4 method [20] and are denoted, in the following, by

$$\mathbf{s}_{k+1} = f(\mathbf{u}_k,\mathbf{s}_k) \quad (27)$$

where $\mathbf{s}_k^\top = \begin{bmatrix} x_k & y_k & \varphi_k \end{bmatrix}$ and $\mathbf{u}_k^\top = \begin{bmatrix} v_k & \nu_k \end{bmatrix}$.

The goal of the robot is to reach an equilibrium point $(\mathbf{s}^{\mathrm{ref}},\mathbf{u}^{\mathrm{ref}})$ of the dynamics (27), with given state vector $\mathbf{s}^{\mathrm{ref}}$, from a given initial position. To do so, it is driven with a nonlinear model predictive controller as defined in (4). The

TABLE I

CONTROLLER AND COPDAC-GQ CONFIGURATION.

Parameter	Value
$\alpha_{\mathbf{w}}$, $\alpha_{\mathbf{v}}$, $\alpha_{\boldsymbol{\beta}}$, $\alpha_{\boldsymbol{\theta}}$	10^{-5}, 10^{-5}, 10^{-5},10^{-7}
$\boldsymbol{\theta}_{\text{init}}$, $\mathbf{w}_{\text{init}}$, $\mathbf{v}_{\text{init}}$, $\boldsymbol{\beta}_{\text{init}}$	10^{-4}, 0, 0, 0
Law of $\boldsymbol{\rho}$, γ	$\mathcal{N}(0, 10^{-2})$, 0.97
N, K, N_{ep}	10, 129, 30
c, d_t, d_0	100, 4, 0.25
q_1, q_2, q_3	1, 1, 0.1
Mini-batch size card($\mathcal{B}$) , $\alpha_{\boldsymbol{\omega}}$	128, 10^{-2}
Hidden layers, neurons per layer	3, 64

cost function (4a) is such that, for any $i \in \overline{0, N-1}$,

$$L(\mathbf{x}_i, \mathbf{u}_i, \theta_X, \theta_U) = \left\|\mathbf{x}_i - \mathbf{s}^{\text{ref}}\right\|^2_{\text{diag}(\boldsymbol{\theta}_X)^2} + \left\|\mathbf{u}_i - \mathbf{u}^{\text{ref}}\right\|^2_{\text{diag}(\boldsymbol{\theta}_U)^2},$$
$$W(\xi_N, \boldsymbol{\theta}_f) = \left\|\mathbf{x}_N - \mathbf{s}^{\text{ref}}\right\|^2_{\text{diag}(\boldsymbol{\theta}_f)^2}$$

where $\text{diag}(\boldsymbol{\theta})$ is a diagonal matrix having for elements the components of $\boldsymbol{\theta}$, with $\boldsymbol{\theta}_X = \begin{bmatrix}\theta_x & \theta_y & \theta_\varphi\end{bmatrix}$, $\boldsymbol{\theta}_U = \begin{bmatrix}\theta_v & \theta_\nu\end{bmatrix}$, and $\boldsymbol{\theta}_f = \begin{bmatrix}\theta_{x_f} & \theta_{y_f} & \theta_{\varphi_f}\end{bmatrix}$, and $\|\mathbf{x}\|^2_Q = \mathbf{x}^\top Q \mathbf{x}$.

The action space is $\mathcal{A} = [-0.6, 0.6] \times [-\pi/2, \pi/2]$. The state space is $\mathcal{S} = [-4, 4] \times [-4, 4] \times \mathbb{R}$. The robot has to stay at a given distance from an obstacle located at $(x_{\text{obs}}, y_{\text{obs}})$ by satisfying

$$\underbrace{1 - 4\frac{(x_i - x_{\text{obs}})^2 + (y_i - y_{\text{obs}})^2}{(d_{\text{rob}} + d_{\text{obs}})^2}}_{\Xi(\mathbf{x}_i)} + \theta_c \leq 0, \qquad (28)$$

where $(x_{\text{obs}}, y_{\text{obs}}) = (0, 0)$, and $d_{\text{rob}} = 0.5\,\text{m}$ and $d_{\text{obs}} = 2\,\text{m}$ are the diameters of a safety circle around the robot and the obstacle, respectively. Parameter θ_c is a tightening variable used to adjust the strength of the collision avoidance constraint.

To train the RL agent, the robot's initial state and objective are $\mathbf{s}_0^\top = \begin{bmatrix}-2 & -2 & 0\end{bmatrix}$ and $\mathbf{s}^{\text{ref}} = \begin{bmatrix}4 & 4 & 0\end{bmatrix}^\top$ so that $\mathbf{u}^{\text{ref}} = 0$. The vectors $\mathbf{s}_0$ and $\mathbf{s}^{\text{ref}}$ remain the same over all the training episodes. The RL stage cost ℓ used in the TD error (14) and the RL performance (3) is defined as

$$\ell = \begin{cases} \left\|\mathbf{s}_k - \mathbf{s}^{\text{ref}}\right\|^2_Q + \|\mathbf{u}_k\|_2 & \text{if } \left\|\mathbf{s}_k - \mathbf{s}^{\text{ref}}\right\|_2 < d_t \\ \left\|\mathbf{s}_k - \mathbf{s}^{\text{ref}}\right\|^2_Q + \Upsilon(\mathbf{s}_k) & \text{if } \left\|\mathbf{s}_k - \mathbf{s}^{\text{ref}}\right\|_2 \geq d_t \end{cases}, \qquad (29)$$

where $Q = \text{diag}(q_1, q_2, q_3)$ allow assigning different weights to each error term, $d_t > 0$ is a constant distance threshold, and $\Upsilon(\mathbf{s}_k)$ is the obstacle penalty collision

$$\Upsilon(\mathbf{s}_k) = c \max\{0, \Xi(\mathbf{s}_k) + d_o\}, \qquad (30)$$

where Ξ is as defined as in (28) and $d_o > 0$ are the distance and the desired safe distance between the robot and the obstacle, respectively, and c is a penalty weight. If a collision occurs, $\Upsilon(\mathbf{s}_k)$ adds a positive penalty to the stage cost ℓ.

The numerical computation is performed using the Ipopt solver provided by the CasADi software framework [21] on a PC equipped of 16 GB of RAM. The initial and constant parameters of the COPDAC-GQ-based NMPC algorithm are provided in Table I.

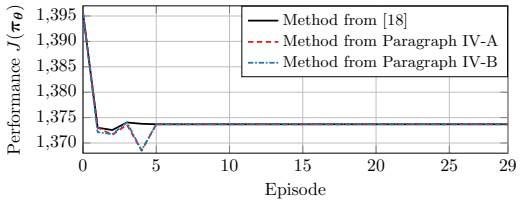

Fig. 1. Performance $J(\boldsymbol{\pi}_{\boldsymbol{\theta}})$ across learning episodes.

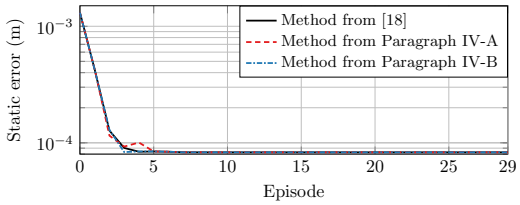

Fig. 2. Variations of the static error across learning episodes.

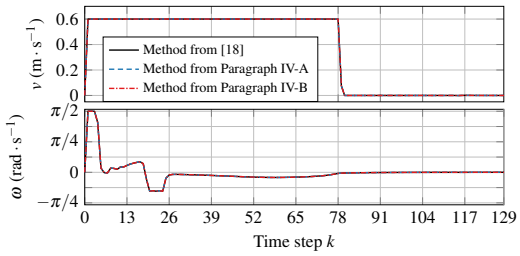

Fig. 3. Variation of controls signals during the final training episode.

B. Simulation results

To showcase the computational efficiency of our algorithms, we train $\boldsymbol{\theta}$ in three ways: one using the method from [18], one using method from Paragraph IV-A, and finally one using the method from Paragraph IV-B. The learning algorithm from [18] update parameters using batch (offline) learning, whereas our methods update the parameters continuously using incremental (online) learning. We note that our proposed methods are valid in both cases.

For all methods, the closed-loop performance improves significantly across the learning episodes, converging to the optimal (or sub-optimal) parameters within just the first 6 episodes (for a total of 780 training steps), as depicted in Figure 1. Figure 2 displays the variation of the static error between the robot final position and the reference. The behavior presented in this figure confirms the conclusions we got from Figure 1.

Figures 3 and 4 present a comparison of the control signals and robot's trajectories obtained with the different methods. We see that from a control point of view, we get the same performance with all the methods, showing that our proposed algorithms do not affect the control performance.

Table II presents the real-time training duration of the three methods for 30 learning episodes. As we can see, the proposed method from Paragraph IV-A reaches the final episode in half the time required for the method from [18]. In addition, since it does not require to train a neural network, the method from

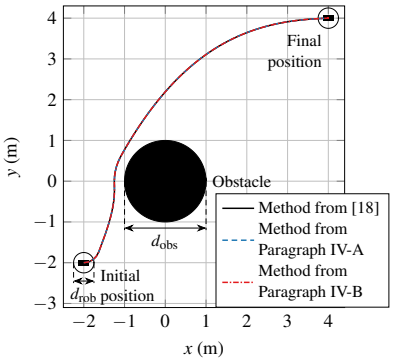

Fig. 4. Robot trajectory during the final training episode.

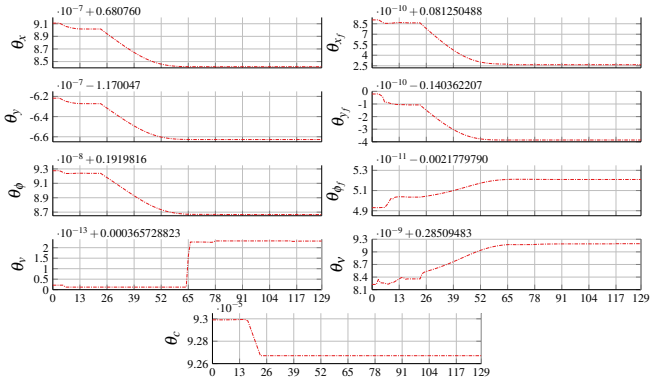

Fig. 5. Variation of the tuned parameters during the last training episode.

TABLE II

TRAINING TIME COMPARISON.

Method based aproximation	Training time (min)
COPDAC-GQ from [18]	15.42
NN-based COPDAC-GQ	8.04
NMPC-based COPDAC-GQ	**7.85**

Paragraph IV-B is even faster. Finally, figure 5 depicts the evolution of the NMPC parameters $\boldsymbol{\theta}$ over the last training episode using the method from Paragraph IV-B.

The robustness of the COPDAC-GQ-based NMPC is clearly demonstrated in the results in terms of speed of convergence, learning stability, and the learned optimal policy. The results also show that the proposed methods do not affect the overall performance of the algorithm while reducing the training time, making it useful for more complicated control tasks with high-dimensional spaces.

VI. CONCLUSION

This paper presents a reinforcement learning-based (RL) nonlinear model predictive control (NMPC) method for mobile robot to accomplish a point stabilization with obstacle avoidance mission. We use a parameterized NMPC scheme as the policy approximation function, and adopt the compatible deterministic actor-critic with gradient Q-learning (COPDAC-GQ) RL method to update the parameters such that the closed-loop performance gets improved with learning. We propose two methods in order to reduce the real-time computation complexity. While we succeed in halving the overall training time, we do not affect the closed loop performance. In future work, the COPDAC-GQ value function will be improved since it can be approximated by the NMPC scheme.

REFERENCES

[1] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control*, 2nd ed. Cham, Switzerland: Springer, 2017.

[2] T. P. Nascimento, C. E. T. Dórea, and L. M. G. Gonçalves, "Nonholonomic mobile robots' trajectory tracking model predictive control: a survey," *Robotica*, vol. 36, no. 5, pp. 676–696, 2018.

[3] Z. Zhang, O. Babayomi, T. Dragicevic, R. Heydari, C. Garcia, J. Rodriguez *et al.*, "Advances and opportunities in the model predictive control of microgrids: Part i–primary layer," *Int. J. Electr. Power Energy Syst.*, vol. 134, p. 107411, 2022.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] D. Han, B. Mulyana, V. Stankovic, and S. Cheng, "A survey on deep reinforcement learning algorithms for robotic manipulation," *Sensors*, vol. 23, no. 7, p. 3762, 2023.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. MIT Press, 2018.

[8] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML '14*, 2014, pp. 387–395.

[9] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint*, p. 1701.07274, 2017.

[10] S. Gros and M. Zanon, "Data-driven economic NMPC using reinforcement learning," *IEEE Trans. Autom. Control*, vol. 65, no. 2, pp. 636–648, 2019.

[11] W. Cai, A. B. Kordabad, H. N. Esfahani, A. M. Lekkas, and S. Gros, "MPC-based reinforcement learning for a simplified freight mission of autonomous surface vehicles," in *60th IEEE CDC*. IEEE, 2021, pp. 2990–2995.

[12] A. B. Kordabad, H. N. Esfahani, A. M. Lekkas, and S. Gros, "Reinforcement learning based on scenario-tree MPC for ASVs," in *Proc. ACC*. IEEE, 2021, pp. 1985–1990.

[13] A. B. Kordabad, W. Cai, and S. Gros, "Multi-agent battery storage management using MPC-based reinforcement learning," in *Proc. CCTA*. IEEE, 2021, pp. 57–62.

[14] H. N. Esfahani, A. B. Kordabad, and S. Gros, "Approximate robust NMPC using reinforcement learning," in *Proc. ECC*. IEEE, 2021, pp. 132–137.

[15] W. Cai, H. N. Esfahani, A. B. Kordabad, and S. Gros, "Optimal management of the peak power penalty for smart grids using MPC-based reinforcement learning," in *60th IEEE CDC*. IEEE, 2021, pp. 6365–6370.

[16] H. R. Maei, C. Szepesvári, S. Bhatnagar, and R. S. Sutton, "Toward off-policy learning control with function approximation," in *ICML '10*, 2010, pp. 719–726.

[17] R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *ICML '09*, 2009, pp. 993–1000.

[18] W. Cai, S. Sawant, D. Reinhardt, S. Rastegarpour, and S. Gros, "A learning-based model predictive control strategy for home energy management systems," *IEEE Access*, vol. 11, pp. 145 264–145 280, 2023.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint*, p. 1412.6980, 2014.

[20] M. Sani, B. Robu, and A. Hably, "Dynamic obstacles avoidance using nonlinear model predictive control," in *Proc. IECON*. IEEE, 2021, pp. 1924–1929.

[21] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.