Received 8 July 2022, accepted 25 July 2022, date of publication 1 August 2022, date of current version 8 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3195530

RESEARCH ARTICLE

Model Predictive Control-Based Reinforcement Learning Using Expected Sarsa

HOOMAAN MORADIMARYAMNEGARI, MARCO FREGO, (Member, IEEE), AND ANGELIKA PEER, (Member, IEEE)
Human-Centered Technologies and Machine Intelligence Laboratory, Faculty of Science and Technology, Free University of Bozen-Bolzano, 39100 Bolzano, Italy

Corresponding author: Angelika Peer (angelika.peer@unibz.it)

This work was supported in part by a university-sponsored Ph.D. fellowship of the ASE PhD programme, and in part by the Open Access Publishing Fund through the Free University of Bozen-Bolzano.

ABSTRACT Recent studies have shown the potential of Reinforcement Learning (RL) algorithms in tuning the parameters of Model Predictive Controllers (MPC), including the weights of the cost function and unknown parameters of the MPC model. However, a framework for easy and straightforward implementation that allows training in just a few episodes and overcoming the need for imposing extra constraints as required by state-of-the-art methods, is still missing. In this study, we present two implementations to achieve these goals. In the first approach, a nonlinear MPC plays the role of a function approximator for an Expected Sarsa RL algorithm. In the second approach, only the MPC cost function is considered as the function approximator, while the unknown parameters of the MPC model are updated based on more classical system identification. In order to evaluate the performance of the proposed algorithms, first numerical simulations are performed on a coupled tanks system. Then, both algorithms are applied to the real system and their closed-loop performance and convergence speed are compared with each other. The results indicate that the proposed algorithms allow tuning of MPCs over very few episodes. Finally, also the disturbance rejection ability of the proposed methods is demonstrated.

INDEX TERMS Model predictive control, reinforcement learning, expected Sarsa algorithm, model-based learning method.

I. INTRODUCTION

Model-predictive control, while being a powerful method, requires an accurate model and proper tuning of the parameters to be effective [1]. To facilitate obtaining both, recently the combination of machine learning (ML) and model predictive control (MPC) has been proposed.

The most prominent techniques in this context are supervised methods, where a global approximator, like an artificial neural network (ANN), is trained offline, based on the input and output data of the real system, to obtain a model of it [2]. This approach, however, not only comes with significant effort and time needed for training the network, but also with an additional computational time required to solve an optimization problem based on models of complex neural network functions, in contrast to models based on a more traditional approach involving differential equations [3].

On the other hand, RL has demonstrated considerable performance when combined with MPC. While the combination of optimal control, such as linear quadratic regulators (LQR) with RL, has been studied before, e.g., in [4] and [5], one of the most valuable works in expanding the theory of combining RL and MPC is [6], where the Q-Learning algorithm has been employed to tune the parameters of an economic nonlinear MPC (ENMPC), so that it can deliver the optimal policy even when the MPC model is not exactly matching the real system. In this approach, MPC has been used as a function approximator for RL, parameterized with the weights of its cost function as well as the unknown parameters of the model. The Q-Learning tries to find the unknown parameters of the MPC by exploring the environment and by minimizing a predefined reward

The associate editor coordinating the review of this manuscript and approving it for publication was Qi Zhou.

function, resulting in training the function approximator based on RL. Since the MPC model is considered known in structure but unknown in some parameters, the number of required episodes is reduced, compared to deep RL methods, in which a neural network model with several hidden layers needs to be trained. Furthermore, constraints and limitations of the real system can be met in the action value function and the policy, which are based on the MPC.

However, although it has been a significant step to combine a model-based controller with a model-free RL, there is still room for at least two possible improvements. Firstly, finding an MPC-RL framework that can achieve proper training in just a few episodes, is still an open research question. Secondly, in the MPC-Q-Learning framework [6]–[8], an extra constraint with respect to the first input must be imposed on the MPC, leading to degradation of the performance and convergence speed of the whole algorithm.

In order to improve the control performance, robust versions of the MPC with Q-Learning are presented in [9], where the safety of the proposed RL algorithm is studied by satisfying a set of prescribed constraints. In [8], the scenario-tree MPC is tuned by Q-Learning for the control of an autonomous surface vehicle (ASV), where an obstacle penalty, time, and energy are considered in the reward function (baseline stage cost). In [10], the uncertainties are considered as ellipsoidal tubes, in which the Gaussian noise is used to model the disturbances and state deviations. Although the controllers have shown more robust behaviour with respect to bounded disturbances, the computational time increased dramatically due to the robust MPC characteristics. Additionally, the issue related to the hard constraint on the policy is not solved, since the Q-Learning formulation has not been changed.

Another attempt to improve the MPC's performance has been made in [11], where parameters of an MPC are learned by Q-Learning when the MPC model is fitted to the real system using system identification. An implementation study has been carried out for the tracking control of two ASVs in [12] using the aforementioned combination. However, computational burden is added to their algorithm by the different methods suggested to avoid conflicts between the two different updating rules.

With respect to policy-based methods [13], one way to eliminate the aforementioned extra constraint on the first input of the MPC-based Q-Learning method (as in [14]) is to use deterministic policy gradients (DPG) [15], which can lead to an optimal policy, when it has few parameters. Two applications of this MPC-DPG method, a multi-agent battery storage system [16], and a smart grid [17], can be found in literature. As an alternative, we present a value-based method that overcomes the need for imposing an extra constraint on the MPC.

A. PAPER CONTRIBUTION

In this study, a value-based learning method is presented for tuning an MPC. In addition to its fast online training process, it provides an easier and more straightforward framework for the implementation on real systems due to the elimination of the extra constraint on the first input of the MPC. To this end, an RL algorithm called Expected Sarsa is adopted. As shown in [18], given the same amount of experience, Expected Sarsa performs better than Sarsa, and in many cases better than Q-Learning, since the variance is eliminated by the random selection of the action in the subsequent action value function.

In this paper, without adding extra computational overhead, we propose two RL algorithms to tune the weights of the MPC cost function and the unknown parameters of the MPC model and compare their performance with the MPC-based Q-Learning method.

In the first solution (PMPCFA), it is assumed that the whole MPC, together with its equality and inequality constraints, is the value function estimator, while the parameters are found using the continuous version of the Expected Sarsa algorithm.

In the second proposed algorithm (PCFFA), the unknown parameters of the model are found by an auxiliary cost function based on system identification, while the remaining parts are updated using the Bellman equation.

In an numerical example, the advantages and disadvantages of both methods are compared with each others. While the control performance of the PCFFA is found to be more desirable, the convergence process of the PMPCFA resulted to be faster. Additionally, it is shown that both approaches outperform the MPC-based Q-Learning method presented in [6] in terms of convergence speed and control performance. Finally, the proposed algorithms are implemented on a real benchmark coupled tanks system and their disturbance rejection ability is demonstrated in experiments.

The remainder of this paper is organized as follows: Section II provides a brief review of Markov decision processes (MDP), Sarsa and Expected Sarsa algorithms. In Section III, we present the problem definition, which is related to overcoming limitations of the MPC-based Q-learning method presented in literature. In Section IV, the two proposed algorithms are discussed. In IV-A an RL algorithm, based on the Expected Sarsa, is introduced, while a parameterized MPC is used as its function estimator (PMPCFA). In IV-B, a second algorithm in which only the MPC cost function is defined as the function approximator, is presented. Here, an extra cost function is employed to estimate the unknown parameters of the MPC model. Section V discusses the results obtained both in simulation (V-A) and for an implementation on a real coupled tanks system (V-B), also in presence of disturbances. Finally, Section VI draws conclusions and offers new research perspectives.

II. BACKGROUND

A broad variety of engineering problems are solved through an instance of an optimization problem, that, in its most abstract formulation, can be formulated as an optimal control problem (OCP). Since real-life problems can reach significant complexity, only few of these problems are typically solved analytically. Recent techniques consider *discretizing* the problem producing an MDP based on Bellman's theory. In the following paragraph, we briefly review the components of an MDP and introduce the mathematical model and its notation, which are useful to describe the state of the art and the contribution of the present study.

A. ELEMENTS OF AN MDP

An MDP is defined through five components [19], formalized as the tuple $\{\mathcal{T}, \mathcal{S}, \mathcal{A}, R(s,a), p_{ij}(a)\}$, namely: the decision epochs $\mathcal{T}$, the states $\mathcal{S}$, the actions $\mathcal{A}$, the reward function $R(s,a)$, and the transition probabilities $p_{ij}(a)$ to pass from state $s_i \in \mathcal{S}$ to $s_j \in \mathcal{S}$ taking the action $a \in \mathcal{A}$. The process is assumed to be stationary, so that the transition probabilities are time-independent, whereas the admissible actions change with time. The term Markov means that the transition probabilities and the cost function depend on the past history only through the current state of the system and by the selected action.

Decisions are made by a decision maker at points in time called decision epochs, which belong to a discrete and finite set $\mathcal{T} = \{1, 2, 3, \ldots, T\}$ with $T < \infty$. Elements in $\mathcal{T}$ are denoted with t and are referred to as times or time steps, whereby a finite T defines a finite horizon problem.

At each time t, the system is in a state $s \in \mathcal{S}$ and the decision maker chooses an action a from the set of allowed actions $\mathcal{A}$. After the selection of a particular action a in state s at time t, the decision maker benefits from a reward $R(s,a)$, which expected value at decision epoch t is given by:

$$R = \sum_{s_i \in \mathcal{S}} \gamma^t R(s,a)\, p(s_i|s,a), \tag{1}$$

where γ is the discount factor between 0 and 1, giving less importance to future data for values strictly smaller than 1. The new state of the system is determined by the transition probabilities $p(\cdot|s,a)$, with:

$$\sum_{s_i \in \mathcal{S}} p(s_i|s,a) = 1. \tag{2}$$

From the control theory point of view, the transition probabilities define the dynamical system of the model, that is, $p(s_+|s,a)$ is equivalent to:

$$s_+ = f^{\text{system}}(s,a). \tag{3}$$

A decision rule is a procedure for selecting an action in each state at the specified time and ranges from deterministic to randomized and history-dependant, according to how the past states and actions are considered. A decision rule is a function $d_t : \mathcal{S} \mapsto \mathcal{A}$ that may depend on the time epoch t. After a decision rule is specified, the rewards and the transition probabilities are functions of the state only: $p(s_j|s_i, d_t(s_i))$ and $R(s_i, d_t(s_i))$.

A policy is a function that gives the specific decision rule to be used at each epoch, thus it is a sequence of decision rules $\pi = (d_1, d_2, \ldots, d_{T-1})$.

A trajectory or history or sample path of the process is an element $\omega = (s_1, a_1, s_2, a_2, \ldots, s_{T-1}, a_{T-1}, s_T)$ of the space $\Omega = (\mathcal{S} \times \mathcal{A})^{T-1} \times \mathcal{S}$.

The computation of the optimal policy is based on backwards induction or *dynamic programming*, derived from Bellman's dynamic programming principle (DPP). Let $Q_t(s_i, a_i)$ be the expected total reward during the next $T - t$ epochs, then the objective of an MDP is to compute $Q_0(s_i, a_i)$, which represents the expected rewards for the next $T - 0 = T$ epochs, if the system starts in state s_i. Since the planning horizon ends at T, terminal values must be specified for all states, e.g., $Q_T(s_i, a_i)$ for all $i = 1, 2, \ldots, N$. The set of all total costs for all states at epoch t is denoted with:

$$Q_t = [Q_t(s_1, a_1), \ldots, Q_t(s_N)]^T. \tag{4}$$

For a planning horizon of length T, Q_t can be computed in terms of Q_{t+1} to produce the recursive compact relation

$$Q_t(s_i,a_i) = R(s_i, d_t(s_i)) + \gamma \sum_{j=1}^{N} p(s_j|s_i, d_t(s_i)) Q_{t+1}(s_j,a_j). \tag{5}$$

The previous equation is called the value iteration equation and is composed of the sum of two terms, the current reward at epoch t and the expected total reward for the remaining times, weighted by the probability p_{ij} that state s_j can be reached in one step from state s_i if decision $d_t(s_i)$ is taken. To find the maximum reward, an optimal policy must be computed: this is possible applying for all states s_i and all times t

$$Q_t(s_i,a_i) = \max_{a \in \mathcal{A}} \left\{ R(s_i, a) + \gamma \sum_{j=1}^{N} p(s_j|s_i, a) Q_{t+1}(s_j,a_j) \right\} \tag{6}$$

which is called the Bellman's Value Function. The sequence of actions that maximizes the previous equation and forms the optimal policy for each state.

In practise, there are many causes that prevent the decision maker to solve exaclty an MDP, the size of the states that can be very large or infinite, the consequent computational complexity, uncertainty in the values of some variables and parameters. Therefore, the classic algorithms for solving *exactly* an MDP are often put aside in favor of more performing *approximating* methods. The next section describes briefly two algorithms named Sarsa and Expected Sarsa that benefit from the aforementioned framework. The description is also beneficial to collocate properly the two methods presented in this study.

B. EXPECTED SARSA ALGORITHM

In the Sarsa algorithm, the elements of the transitions, $(\boldsymbol{s}, \boldsymbol{a}, R, \boldsymbol{s}_+, \boldsymbol{a}_+)$, are employed to form the Bellman equation [18], [20]:

$$Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a}) = R + \gamma Q_{\boldsymbol{\theta}}(\boldsymbol{s}_+, \boldsymbol{a}_+), \tag{7}$$

where $\boldsymbol{a}_+$ is the action computed at the subsequent state $\boldsymbol{s}_+$ by the policy $\pi_{\boldsymbol{\theta}}$ (called control law in control literature) and

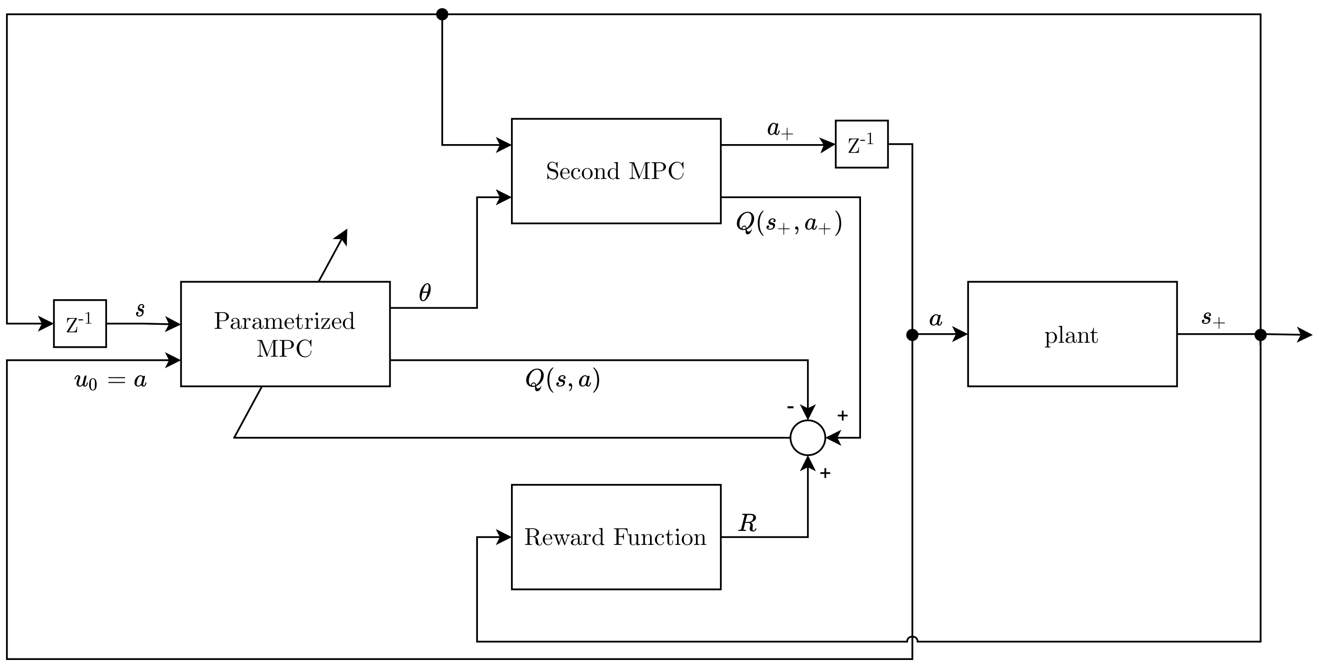


FIGURE 1. Block diagram of the model predictive control-based Q-learning method.

applied to the system in the next state s_+. $Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a})$ expresses the estimation of the value of the state-action pair called the action value function. In continuous systems, regardless of the performance of the RL algorithm, every function parameterized with unknown parameters $\boldsymbol{\theta}$ can be used as the action value function. Using this algorithm, an agent has to estimate how much reward it might receive in a state by doing an action, or, more precisely, what is the expected return (the sum of the future rewards) of a state-action pair [18].

At each sampling time, the unknown parameters, $\boldsymbol{\theta}$, are updated based on the temporal difference error, δ_k, stating the difference between the expected return and the Q value of the current state-action as follows [20]:

$$\delta_k = R + \gamma Q_{\boldsymbol{\theta}}(\boldsymbol{s}_+, \boldsymbol{a}_+) - Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a}), \tag{8}$$

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} + \boldsymbol{\alpha}\delta_k \nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a}), \tag{9}$$

where α is the learning rate. By updating the unknown parameters, the policy and action value function can both converge to an optimal policy and action value function, respectively, provided that every state-action pair can be visited finitely often [21].

In one of the variations of the Sarsa algorithm, named Expected Sarsa, instead of just using $\boldsymbol{a}_+$ in $\boldsymbol{s}_+$ in the target policy in (7), its expected value is computed over all actions available in s_+, because it is found that by doing this, the variance in the updates is reduced [21]. Therefore, the error δ_k can be rewritten as follows:

$$\delta_k = R + \gamma \sum_{\boldsymbol{a}'} \pi(\boldsymbol{a}'|s_+) Q_{\boldsymbol{\theta}}(\boldsymbol{s}_+, \boldsymbol{a}') - Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a}), \tag{10}$$

with $\pi(\boldsymbol{a}'|s_+)$ the probability of taking action $\boldsymbol{a}'$ in state $\boldsymbol{s}_+$.

III. PROBLEM DEFINITION

For a better understanding of the problem we tackled, we first briefly describe the MPC-Q-Learning method introduced in [6] and explain its limitations. The update rule of Q-Learning can be considered a special case of the one adopted for Expected Sarsa, where, instead of the expected value of the subsequent action value function over all available actions in the next state, only the action that makes the target policy greedy is selected:

$$Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a}) = R + \gamma \min_{\boldsymbol{a}'} Q_{\boldsymbol{\theta}}(\boldsymbol{s}_+, \boldsymbol{a}'). \tag{11}$$

In the MPC-Q-Learning formulation as indicated in Fig. 1 and adopted in [6]–[8], the target policy is defined as a typical MPC, where the first element of the control output vector is applied to the system with one sampling time delay. Because of this, the behavior policy in the next sampling time, which is another MPC, must be constrained with the action obtained by the target policy in the previous time step ($u_0 = a$). Since the behavior policy in Q-Learning usually acts exploratory [21], the whole exploration capability of the algorithm is compromised by the aforementioned constraint.

In this paper, we overcome this limitation by employing the Expected Sarsa algorithm with two different definitions for the action value function. We show that by eliminating the aforementioned constraint, not only the convergence speed increases, but also the implementation process becomes easier and more straightforward.

IV. METHODS

In this section, two different definitions are adopted for the MPC-based value function that result in two algorithms to be presented. Our proposed solutions both overcome the aforementioned issue of the presence of the extra input constraint on the MPC. Indeed, with reference to Fig. 1 (the state-of-the-art Q-Learning method), the block *Second MPC* imposes an extra input constraint on the block *Parametrized MPC* ($u_0 = a$). This forces the block *Parametrized MPC* to apply the same action as generated by the block *Second MPC* at the previous time step. However, with respect to our proposed solutions, the actions generated by the blocks *Parametrized MPC* and *Second MPC* do not need to be the same, thus avoiding this constraint.

A. PARAMETRIZED MODEL PREDICTIVE CONTROL (MPC) AS FUNCTION APPROXIMATOR (PMPCFA)

In this section, the whole MPC is considered as the action value function, which is parameterized in the unknown parameters of its model as well as in the weights of the cost function:

$$Q_\theta(\boldsymbol{s},\boldsymbol{a}) = \min \sum_{i=0}^{N-1} \gamma^i [(\boldsymbol{x}_i - \boldsymbol{x}_{\mathrm{ref}_i})^2_{\boldsymbol{\theta}_{\boldsymbol{x}}} + (\boldsymbol{u}_i - \boldsymbol{u}_{\mathrm{ref}_i})^2_{\boldsymbol{\theta}_{\boldsymbol{u}}} + \boldsymbol{\theta}_{\boldsymbol{w}}^T \boldsymbol{\sigma}_i] + \gamma^N [(\boldsymbol{x}_N - \boldsymbol{x}_{\mathrm{ref}_N})^2_{\boldsymbol{\theta}_{\boldsymbol{x}_f}} + \boldsymbol{\theta}_{\boldsymbol{w}_f}^T \boldsymbol{\sigma}_N], \quad (12)$$

$$\text{subject to } \boldsymbol{x}_{i+1} = f_{\boldsymbol{\theta}_m}^{\mathrm{model}}(\boldsymbol{x}_i,\boldsymbol{u}_i),\ \ \boldsymbol{x}_0 = \boldsymbol{s},\ \text{and} \quad (13)$$

$$\boldsymbol{g}(\boldsymbol{u}_i) \le 0, \quad (14)$$

$$\boldsymbol{h}(\boldsymbol{x}_i,\boldsymbol{u}_i) \le \boldsymbol{\sigma}_i, \quad (15)$$

$$\boldsymbol{h}^f(\boldsymbol{x}_N) \le \boldsymbol{\sigma}_N, \quad (16)$$

$$\boldsymbol{\sigma}_{0,\cdots,N} \ge 0, \quad (17)$$

where $Q_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a})$ acts as the action value function estimator at the current state $\boldsymbol{s}$ and action $\boldsymbol{a}$; $\boldsymbol{x}_i$, $\boldsymbol{u}_i$ and $\boldsymbol{\sigma}_i$ are the predicted state, control input, and slack variables at the sampling time i, respectively; $\boldsymbol{x}_{\mathrm{ref}}$ and $\boldsymbol{u}_{\mathrm{ref}}$ are the reference inputs of the state and action, N is the prediction horizon, $\boldsymbol{g}(\boldsymbol{u}_i)$ is the input constraint, $\boldsymbol{h}(\boldsymbol{x}_i,\boldsymbol{u}_i)$ is the mixed constraint, $\boldsymbol{h}^f(\boldsymbol{x}_i)$ is the final constraint, the discount factor is $0 < \gamma \le 1$; $f_{\boldsymbol{\theta}_m}^{\mathrm{model}}$ is the model of the system with unknown parameters $\boldsymbol{\theta}_m$. The variables in $\boldsymbol{\theta} = [\boldsymbol{\theta}_{\boldsymbol{x}}, \boldsymbol{\theta}_{\boldsymbol{u}}, \boldsymbol{\theta}_{\boldsymbol{w}}, \boldsymbol{\theta}_{\boldsymbol{x}_f}, \boldsymbol{\theta}_{\boldsymbol{w}_f}, \boldsymbol{\theta}_m]$ are the unknown parameters which are found during the interaction with the environment. Parameters $\boldsymbol{\theta}_{\boldsymbol{x}}$, $\boldsymbol{\theta}_{\boldsymbol{u}}$, $\boldsymbol{\theta}_{\boldsymbol{w}}$, $\boldsymbol{\theta}_{\boldsymbol{x}_f}$, and $\boldsymbol{\theta}_{\boldsymbol{w}_f}$ are the weights of cost function terms including the tracking error, control input, slack variables, final tracking error, and final slack variable, respectively.

The solution of the above optimization problem leads to a sequence of control outputs, $[\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_N]$, at each sampling time, while the policy is to choose the first element with probability 1 and apply it to the system. Since the same policy is used for the target policy, the temporal difference error δ_k in (8) must be rewritten as follows [20]:

$$\delta_k = R + \gamma Q_{\boldsymbol{\theta}}(\boldsymbol{s}_+, \boldsymbol{a}'_+) - Q_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a}), \quad (18)$$

where $\boldsymbol{a}'_+$ is the action computed in $\boldsymbol{s}_+$ by another MPC after applying $\boldsymbol{a}$ to the system by the first MPC. In fact, since the action space is continuous, computing $\sum_{\boldsymbol{a}'} \pi(\boldsymbol{a}'|\boldsymbol{s}_+)$ is not possible [20]. Then, one possibility is to simply consider the action computed by the second MPC as $\boldsymbol{s}_+$.

Since the goal of this algorithm is to minimize the expected reward, the reward function must be defined such that the MPC performance increases when the reward decreases. One simple choice can be the difference between the current state and the reference input as follows:

$$R = (\boldsymbol{s} - \boldsymbol{x}_{\mathrm{ref}})^T diag(\boldsymbol{v})(\boldsymbol{s} - \boldsymbol{x}_{\mathrm{ref}}), \quad (19)$$

where $\boldsymbol{v}$ is a weighting vector with which the importance of the error of each state can be changed.

As explained earlier, (9) is used to update the parameters of the MPC, where $\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a})$ needs to be solved in each time step, k. It has been proven in [6] and [22] that:

$$\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(s,a) = \nabla_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a},z), \quad (20)$$

$$L_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a},z) = \Phi_{\boldsymbol{\theta}} + \boldsymbol{\lambda}^T G_{\boldsymbol{\theta}}, \quad (21)$$

where L is the Lagrange function associated to the MPC in (12)-(17), $\Phi_{\boldsymbol{\theta}}$ is the cost function of the MPC in (12), $G_{\boldsymbol{\theta}}$ is the equality constraint of the MPC related to the dynamic model of the system in (13), $\boldsymbol{\lambda}$ are the associated dual variables of $G_{\boldsymbol{\theta}}$. Vector $\boldsymbol{z} = [\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\sigma}, \boldsymbol{\lambda}]$ is the solution of the MPC obtained from solving (12)-(17).

According to the Expected Sarsa algorithm presented in [21], after initializing $\boldsymbol{\theta}$ and $\boldsymbol{s}$, we need to solve the parametrized MPC in (12)-(17) in order to obtain $\boldsymbol{z}$ in each episode. Doing this, the MPC cost function, $\Phi(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\sigma}, \boldsymbol{x}_{\mathrm{ref}}, \boldsymbol{u}_{\mathrm{ref}}, \boldsymbol{\theta})$, can be computed and defined as the value of $Q_{\boldsymbol{\theta}}(\boldsymbol{s},\boldsymbol{a})$. Afterwards, according to the defined policy, the first element of $\boldsymbol{u}$ is applied to the real system so that the next state, $\boldsymbol{s}_+$, and the reward value, R, can be observed. In order to approximate the target action value function, another MPC needs to be solved by considering $\boldsymbol{x}_0 = \boldsymbol{s}_+$, so that $Q(\boldsymbol{s}_+, \boldsymbol{a}'_+)$ can be obtained. Now, we are able to compute the error δ_k in (18) with which parameters of the MPC can be updated by (9). The pseudocode of the PMPCFA algorithm is given in Alg. 1. The block diagram of the algorithm is shown in Fig. 2.

The advantage of the proposed algorithm based on Expected Sarsa rather than MPC-based Q-Learning as presented in [6]–[8], is that there is no need to add an extra constraint related to the first input of the parametrized MPC. In other words, in the Q-Learning formulation, to obtain $Q(\boldsymbol{s},\boldsymbol{a})$, an MPC needs to be solved in which $\boldsymbol{u}_0 = \boldsymbol{a}$ is an additional constraint. Moreover, since two MPCs have to be solved in each sampling time, the horizon needs to be chosen short to reduce the computational cost. In this way, forcing the controller to start from a specific input, has a negative effect on the performance, as illustrated later in Section V-A. On the other hand, even if more computational resources were available to allow for a longer horizon, the result would be a poor control performance since the model outputs are not necessarily the same as the ones of the real system.

Our proposed method PMPCFA has a faster convergence, as explained in Section III, thanks to the removal of the extra input constraint.

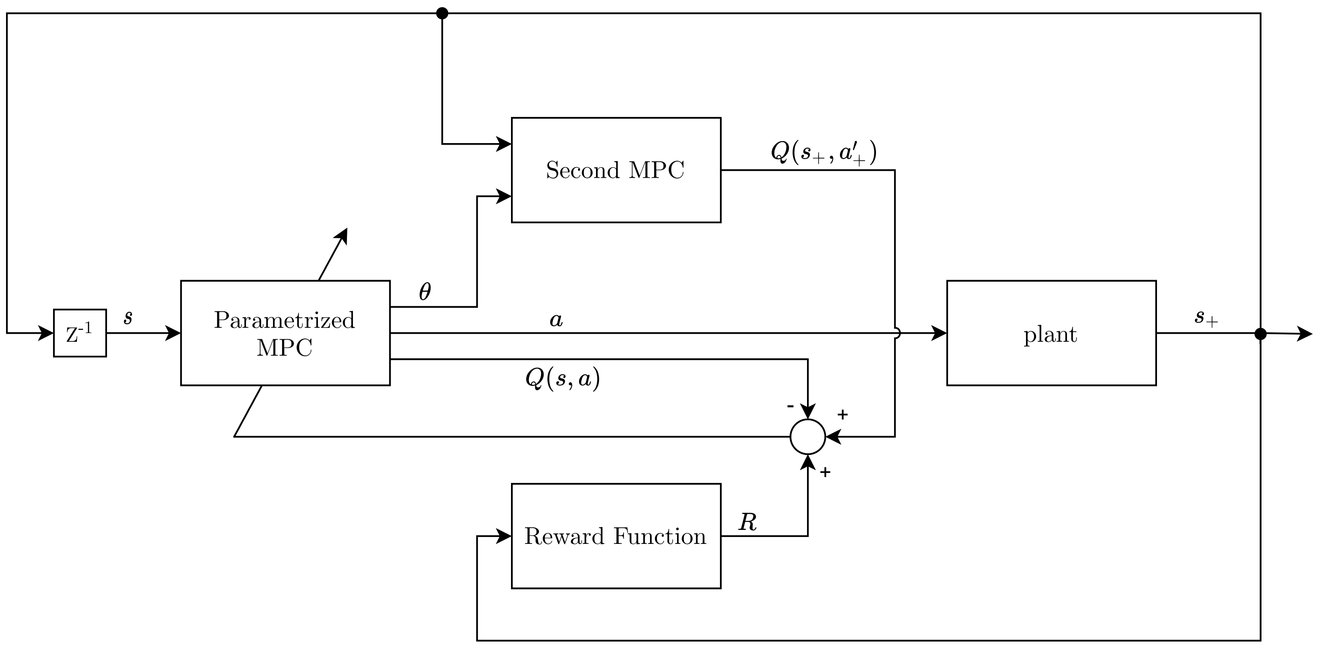


FIGURE 2. Block diagram of the proposed model predictive control-based reinforcement learning using the PMPCFA algorithm.

Algorithm 1 The MPC-Based RL Using the PMPCFA Algorithm

```
Data: $\alpha \geq 0, 0 \leq \gamma \leq 1$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
while $episode < FinalEpisode$ do
    $\boldsymbol{s} \leftarrow \boldsymbol{s}_0$
    while $k < FinalTime/SamplingTime$ do
        Obtain $\boldsymbol{x}$, $\boldsymbol{u}$, $\boldsymbol{\sigma}$, $\boldsymbol{\lambda}$, and $Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a})$ from (12) to (17)
        by $\boldsymbol{x}_0 = \boldsymbol{s}$.
        Apply the first element of $\boldsymbol{u}$ to the real system
        Observe $\boldsymbol{s}_+$ and $R$
        Obtain $Q_{\boldsymbol{\theta}}(\boldsymbol{s}_+, \boldsymbol{a}'_+)$ from (12) to (17) by $\boldsymbol{x}_0 = \boldsymbol{s}_+$
        Compute $\delta_k$ according to (18)
        Obtain $\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a})$ from (20) to (21)
        Update $\boldsymbol{\theta}$ according to (9)
        $k \leftarrow k + 1$
        $\boldsymbol{s} \leftarrow \boldsymbol{s}_+$
    end
    $episode \leftarrow episode + 1$
end
```

B. PARAMETRIZED COST FUNCTION AS FUNCTION APPROXIMATOR (PCFFA)

In the second proposed method, instead of considering the whole MPC as function estimator, the MPC cost function is parameterized just in its weights and defined as the action value function. The cost function is then given by:

$$Q_{\boldsymbol{\theta}_{\text{cost}}}(\boldsymbol{s}, \boldsymbol{a}) = \sum_{i=0}^{N-1} \gamma^i [(\boldsymbol{x}_i - \boldsymbol{x}_{\text{ref}_i})^2_{\boldsymbol{\theta}_{\boldsymbol{x}}} + (\boldsymbol{u}_i - \boldsymbol{u}_{\text{ref}_i})^2_{\boldsymbol{\theta}_{\boldsymbol{u}}} + \boldsymbol{w}^T \boldsymbol{\sigma}_i] + \gamma^N [(\boldsymbol{x}_N - \boldsymbol{x}_{\text{ref}_N})^2_{\boldsymbol{\theta}_{\boldsymbol{x}_f}} + \boldsymbol{\theta}^T_{\boldsymbol{w}_f} \boldsymbol{\sigma}_N]. \tag{22}$$

where $\boldsymbol{\theta}_{\text{cost}} = [\boldsymbol{\theta}_{\boldsymbol{x}}, \boldsymbol{\theta}_{\boldsymbol{u}}, \boldsymbol{\theta}_{\boldsymbol{w}}, \boldsymbol{\theta}_{\boldsymbol{x}_f}, \boldsymbol{\theta}_{\boldsymbol{w}_f}]$ does not include the unknown parameters of the model $\boldsymbol{\theta}_m$.

Unlike the previous algorithm, a greedy policy is now defined, such that the minimization of the action value function is computed to generate the action. Therefore, the policy is written as follows:

$$\pi(\boldsymbol{\theta}) = \min Q_{\boldsymbol{\theta}_{\text{cost}}}(\boldsymbol{s}, \boldsymbol{a}) \quad \text{subject to } (13) - (17). \tag{23}$$

While the output of this policy is a sequence of predictions of future inputs, just its first element is chosen with probability 1 and applied to the real system. Since the action value function does not include $\boldsymbol{\theta}_m$, whereas the policy $\pi(\boldsymbol{\theta})$ is a function of all parameters $\boldsymbol{\theta}$ (including the unknown parameters of the MPC model), an auxiliary cost function is required, and can be defined as:

$$\Phi_{\boldsymbol{\theta}_m} = \frac{1}{2}[(\boldsymbol{s}_+) - f^{\text{model}}_{\boldsymbol{\theta}_m}(\boldsymbol{s}, \boldsymbol{a})]^2. \tag{24}$$

After applying the action $\boldsymbol{a}$, derived from the policy $\pi(\boldsymbol{\theta})$, to the real system, $\boldsymbol{s}_+$ can be observed. The square error between this next state and the output of the MPC model in state $\boldsymbol{s}$ and action $\boldsymbol{a}$ is used to define the aforementioned auxiliary cost function. Employing a system identification method based on gradient descent, to minimize the auxiliary cost function in (24), similar to [11] and [12], the model parameters can be included in the update rule [23], [24] (a similar combination of the Bellman equation and the gradient decent is presented in [25] and [26]):

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} + \boldsymbol{\alpha}[\delta_k \nabla_{\boldsymbol{\theta}_{\text{cost}}} Q_{\boldsymbol{\theta}_{\text{cost}}}(\boldsymbol{s}, \boldsymbol{a}), -\nabla_{\boldsymbol{\theta}_m} \Phi_{\boldsymbol{\theta}_m}]. \tag{25}$$

In fact, $\nabla_{\boldsymbol{\theta}_m} \Phi_{\boldsymbol{\theta}_m}$ is a gradient descent-based updating rule which defines the last element of the parameter vector in (25).

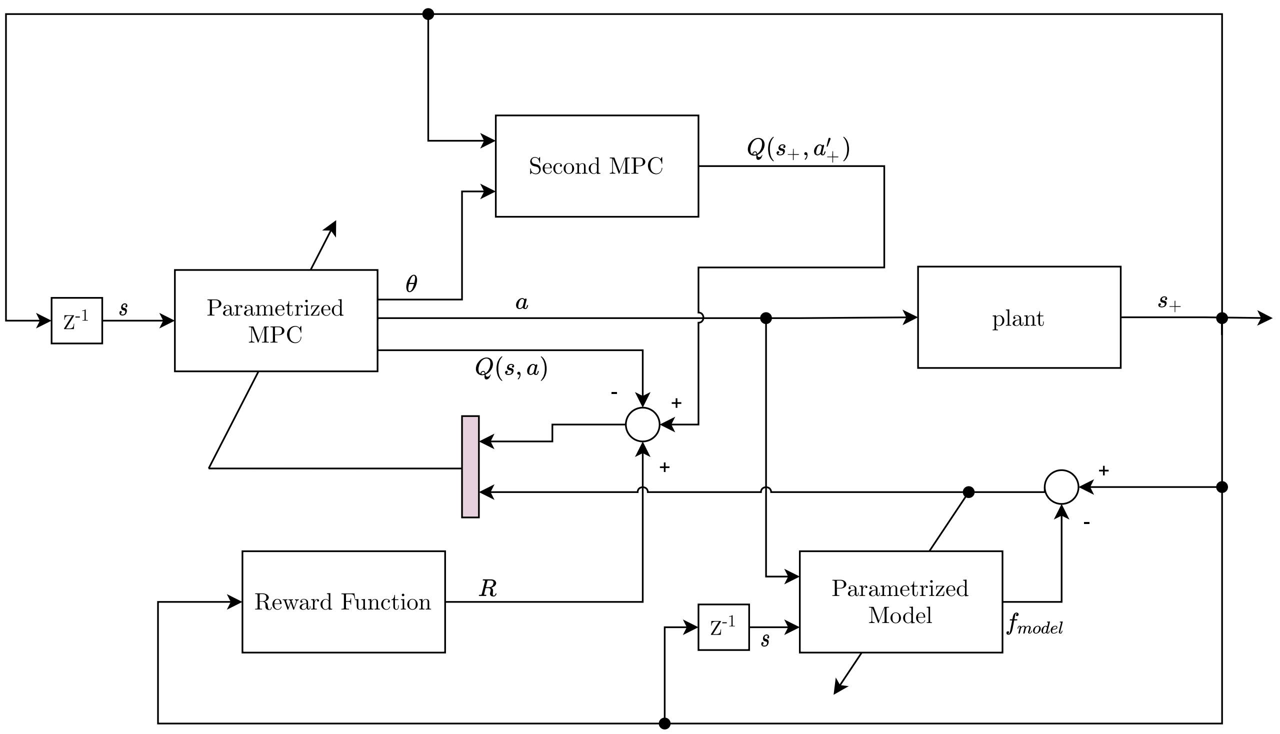


FIGURE 3. Block diagram of the proposed model predictive control-based reinforcement learning using the PCFFA algorithm.

Summarizing, first an MPC is solved according to the policy in (23), so that the sequence of $\boldsymbol{x}$, $\boldsymbol{u}$, and $\boldsymbol{\sigma}$ can be obtained. In this way, $Q_{\boldsymbol{\theta}_{\text{cost}}}(\boldsymbol{s}, \boldsymbol{a})$ is computed using (22). Similar to the PMPCFA algorithm, the first element of $\boldsymbol{u}$ is then applied to the real system. After receiving the next state $\boldsymbol{s}_+$ and the reward value R, the policy needs to minimize the cost function with respect to the defined constraints for $\boldsymbol{x}_0 = \boldsymbol{s}_+$ according to (23). Having the new $\boldsymbol{x}$, $\boldsymbol{u}$, and $\boldsymbol{\sigma}$, $Q_{\boldsymbol{\theta}_{\text{cost}}}(\boldsymbol{s}_+, \boldsymbol{a}'_+)$ can be computed using (22). The temporal difference error δ_k in (18) can now be formed using just $\boldsymbol{\theta}_{\text{cost}}$. However, the issue is that updating the parameters of the model $\boldsymbol{\theta}_m$ is not possible with the help of the Bellman equation because the MPC cost function does not include them. To address this issue, the difference between the next state and the output of the MPC model is used to form an additional cost function as in (24). Therefore, an extra term, which is a simple gradient descend, is added to the update rule as stated in (25). The overall PCFFA algorithm is presented in Alg. 2. The corresponding block diagram is illustrated in Fig. 3.

The advantage of this proposed algorithm compared to the previous one is that since the system identification is used to estimate the unknown parameters of the model, the open-loop behavior of the model becomes similar to the real system. This eliminates the estimation bias that can occur in the PMPCFA algorithm, leading to a more desired control performance. However, as typical for system identification, the input signal must be sufficiently rich in order to excite all frequencies of the real system. In the first proposed algorithm (PMPCFA), there is no such requirement.

Algorithm 2 The MPC-Based RL Using the PCFFA Algorithm

```
Data: $\alpha \geq 0, 0 \leq \gamma \leq 1$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
while $episode < FinalEpisode$ do
    $\boldsymbol{s} \leftarrow \boldsymbol{s}_0$
    while $k < FinalTime/SamplingTime$ do
        Obtain $\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\sigma}$, and $Q_{\boldsymbol{\theta}_{\text{cost}}}(\boldsymbol{s}, \boldsymbol{a})$ according to (23) by $\boldsymbol{x}_0 = \boldsymbol{s}$
        Apply the first element of $\boldsymbol{u}$ to the real system
        Observe $\boldsymbol{s}_+$ and $R$
        Obtain $Q_{\boldsymbol{\theta}_{\text{cost}}}(\boldsymbol{s}_+, \boldsymbol{a}'_+)$ according to (23) by $\boldsymbol{x}_0 = \boldsymbol{s}_+$
        Compute $\delta_k$ according to (18)
        Compute $\Phi_{\boldsymbol{\theta}_m}$ according to (24)
        Obtain $\nabla_{\boldsymbol{\theta}_{\text{cost}}} Q_{\boldsymbol{\theta}_{\text{cost}}}(\boldsymbol{s}, \boldsymbol{a})$
        Obtain $\nabla_{\boldsymbol{\theta}} \Phi_{\boldsymbol{\theta}_m}$
        Update $\boldsymbol{\theta}$ according to (25)
        $k \leftarrow k + 1$
        $\boldsymbol{s} \leftarrow \boldsymbol{s}_+$
    end
    $episode \leftarrow episode + 1$
end
```

V. RESULTS

In this section, the proposed algorithms are evaluated and compared on a case study of a coupled tanks system in simulations as well as in experiments.

A. SIMULATION RESULTS

First, both proposed algorithms are applied to a nonlinear simulation model of a coupled tanks system as shown in

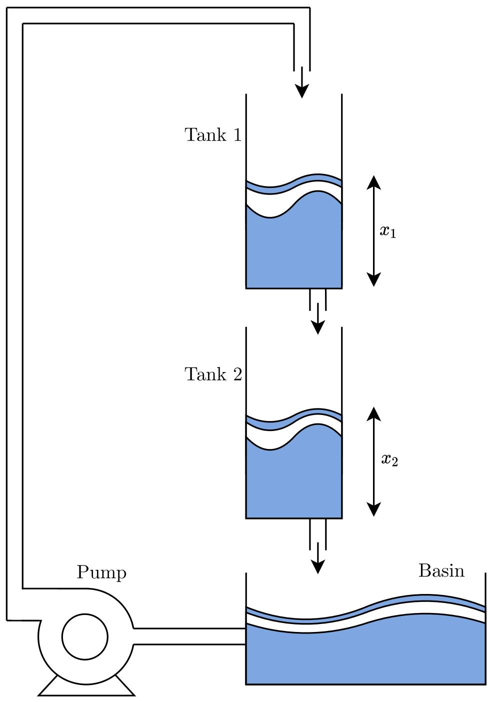


FIGURE 4. Schematic of the coupled tanks system.

TABLE 1. MPC and RL parameters specification.

A_t	$= 1551.79\ cm^2$	$\theta_{x_{1_0}}$	$= 4$
A_o	$= 0.1781\ cm^2$	$\theta_{x_{2_0}}$	$= 4$
g	$= 981\ cm/s^2$	θ_{u_0}	$= 2.5$
K	$= 3.3\ cm^3/s/V$	$\theta_{x_{f_{1_0}}}$	$= 4$
N	$= 10$	$\theta_{x_{f_{2_0}}}$	$= 4$
γ	$= 0.8$	θ_{m_0}	$= 10\ cm^3/s/V$
$\alpha_{\theta_{x_1}}$	$= 10^{-7}$	$\min x_1$	$= \min x_2 = 0\ cm$
$\alpha_{\theta_{x_2}}$	$= 10^{-7}$	$\max x_1$	$= \max x_2 = 30\ cm$
α_{θ_u}	$= 10^{-7}$	$\min u$	$= 0\ V$
$\alpha_{\theta_{x_{f_1}}}$	$= 10^{-7}$	$\max u$	$= 22\ V$
$\alpha_{\theta_{x_{f_2}}}$	$= 10^{-7}$	u_0	$= 0\ V$
$\alpha_{\theta_{m_{\text{PMPCFA}}}}$	$= 10^{-7}$	$\alpha_{\theta_{m_{\text{PCFFA}}}}$	$= 10^{-3}$

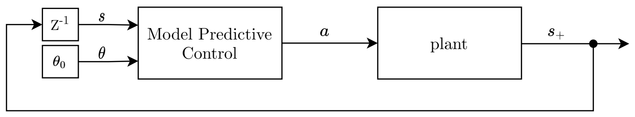


FIGURE 5. Block diagram of typical model predictive control used for comparison.

Fig. 4. The nonlinear dynamics governing the system can be described as follows:

$$\dot{x}_1(t) = \frac{1}{A_{t1}}\left(-A_{o1}\sqrt{2gx_1(t)} + Ku(t)\right), \tag{26}$$

$$\dot{x}_2(t) = \frac{A_{o1}}{A_{t2}}\sqrt{2gx_1(t)} - \frac{A_{o2}}{A_{t2}}\sqrt{2gx_2(t)}, \tag{27}$$

where $x_1(t)$ and $x_2(t)$ are the levels of the water in the upper and lower tanks, the control input $u(t)$ is the voltage of the pump that fills the upper tank, A_{t1} and A_{t2} are upper and lower tank sectional areas, the sections of the outlets are A_{o1} and A_{o2}, and g is the gravitational constant. The pump flow constant K is considered as the unknown parameter of the model, θ_m.

Linearizing the model around the operating points L_{o1} and L_{o2}, the following linear equations are obtained:

$$\dot{x}_1(t) = -\frac{gA_{o1}}{A_{t1}\sqrt{2gL_{o1}}}x_1(t) + \frac{K}{A_{t1}}u(t), \tag{28}$$

$$\dot{x}_2(t) = \frac{gA_{o1}}{A_{t2}\sqrt{2gL_{o1}}}x_1(t) - \frac{gA_{o2}}{A_{t2}\sqrt{2gL_{o2}}}x_2(t). \tag{29}$$

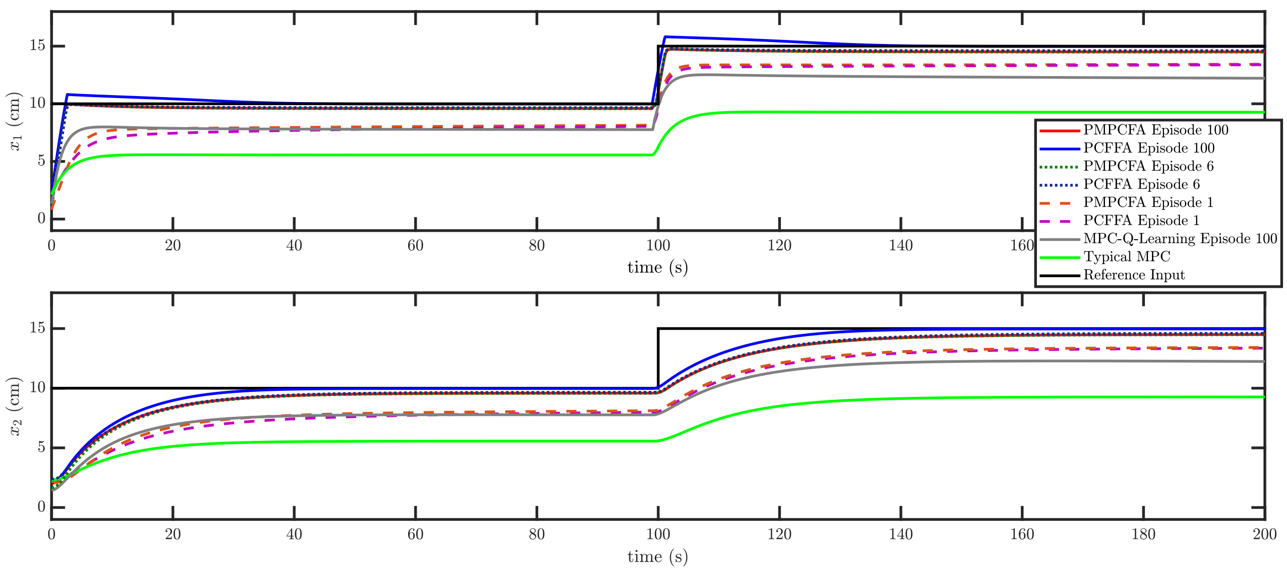
While the nonlinear model in (26) and (27) is employed as the plant in the simulation, the linear model of (28) and (29) is considered as the MPC model in (13). Consequently, in addition to the unknown parameter, K, the linearization of the model may result in a mismatch between model and plant, when not operated in the vicinity of the operation points; especially in the transient phase before converging to the operating points.

The MPC is formulated as follows:

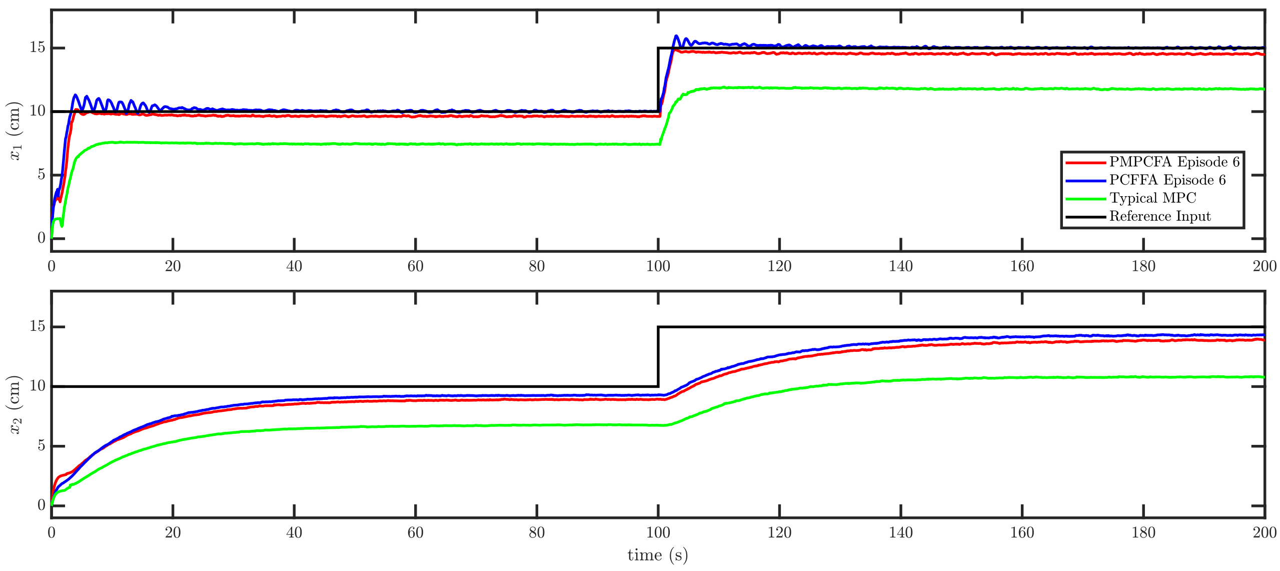
$$\begin{aligned}
&Q_\theta(\boldsymbol{s}, \boldsymbol{a}) \\
&= \min \sum_{i=0}^{N} \gamma^i \begin{bmatrix} x_{i_1} - x\text{ref}_{i_1} \\ x_{i_2} - x\text{ref}_{i_2} \\ u_i - u\text{ref}_i \end{bmatrix}^T \begin{bmatrix} \theta_{x_1}^2 & 0 & 0 \\ 0 & \theta_{x_2}^2 & 0 \\ 0 & 0 & \theta_u^2 \end{bmatrix} \begin{bmatrix} x_{i_1} - x\text{ref}_{i_1} \\ x_{i_2} - x\text{ref}_{i_2} \\ u_i - u\text{ref}_i \end{bmatrix} \\
&\quad + \gamma^N \begin{bmatrix} x_{N_1} - x\text{ref}_{N_1} \\ x_{N_2} - x\text{ref}_{N_2} \end{bmatrix}^T \begin{bmatrix} \theta_{xf_1}^2 & 0 \\ 0 & \theta_{xf_2}^2 \end{bmatrix} \begin{bmatrix} x_{N_1} - x\text{ref}_{N_1} \\ x_{N_2} - x\text{ref}_{N_2} \end{bmatrix} \\
&\quad \text{subject to (28) and (29)}, \quad \boldsymbol{x}_0 = \boldsymbol{s}, \\
&\quad \min x_1 \leq x_{i_1} \leq \max x_1, \\
&\quad \min x_2 \leq x_{i_2} \leq \max x_2, \\
&\quad \min u \leq u_i \leq \max u.
\end{aligned} \tag{30}$$

The proposed MPC-based RL algorithms are applied to the coupled tanks system for 100 episodes and compared with the MPC-based Q-Learning method presented in [6]. The numerical computation is performed using the Ipopt solver provided by the CasADi software framework [27] on a PC equipped with an AMD Threadripper Pro CPU and 64 GB of RAM. The initial and constant parameters of the model, MPC, and RL are given in Table 1.

The water levels for both tanks in the first, sixth, and last episodes are shown in Fig. 6a and compared to the MPC-based Q-learning method [6] and a typical MPC. The typical MPC [28] refers to an MPC, which cost function weights and model parameters remain constant throughout and are considered the same as the initial conditions of the proposed algorithms. The block diagram of the typical MPC used for comparison is shown in Fig. 5.



(a) Simulation results: For the first, the 6^{th}, and 100^{th} episodes as well as the typical MPC and MPC-Q-Learning method.



(b) Experimental results: For the 6^{th} episode and the typical MPC.

FIGURE 6. **Trajectories of the water levels for upper and lower tanks.**

As can be observed in Fig. 6a, the performance of the PMPCFA algorithm outperforms the PCFFA algorithm in the first episode. However, in the last episode, it is the PCFFA algorithm that has the best control performance in terms of tracking the reference input. There exist steady state errors in both upper and lower levels for the PMPCFA algorithm, even after 100 episodes. This can be explained by the way the parameters of the MPC model are tuned. Since the update law in (18) and (9) does not necessarily cause the same open-loop behavior of the MPC model and the plant, and due to the lack of a strategy for more exploration, it is more likely to have a steady-state error in case of deterministic environments. On the other hand, as the outputs of the MPC model and the system are more similar in the PCFFA algorithm, if the controller output is persistently exciting, tracking can be achieved without steady-state error even in case of a deterministic environment without sufficient exploration. In fact, since the coupled tanks system is an underactuated system, the PCFFA algorithm learns to make an overshoot for the trajectory of the upper tank, so that the lower tank can be filled sooner. In this way, the value of the baseline stage cost can be minimized. The tracking performance of the PMPCFA for both the 6^{th} and 100^{th} episodes remains the same, showing that the training is completed after few episodes. However, although the performance of the PCFFA in the 6^{th} episode is slightly better than the one of PMPCFA, the performance

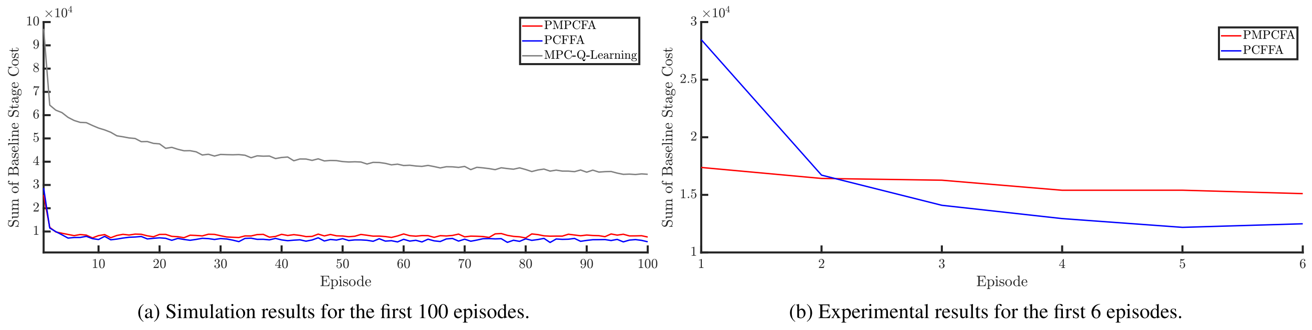


FIGURE 7. Sum of the baseline stage cost (reward) for MPC-based RL methods.

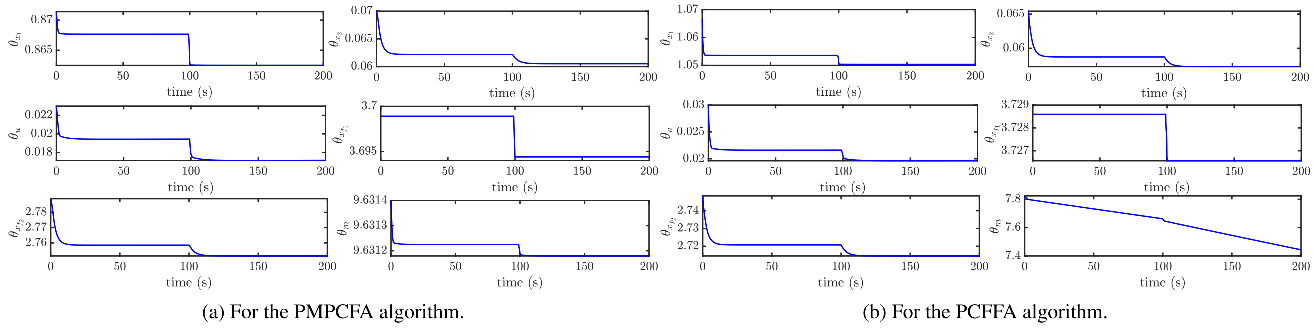


FIGURE 8. Simulation results: Learned parameters of both algorithms for the 6th episode.

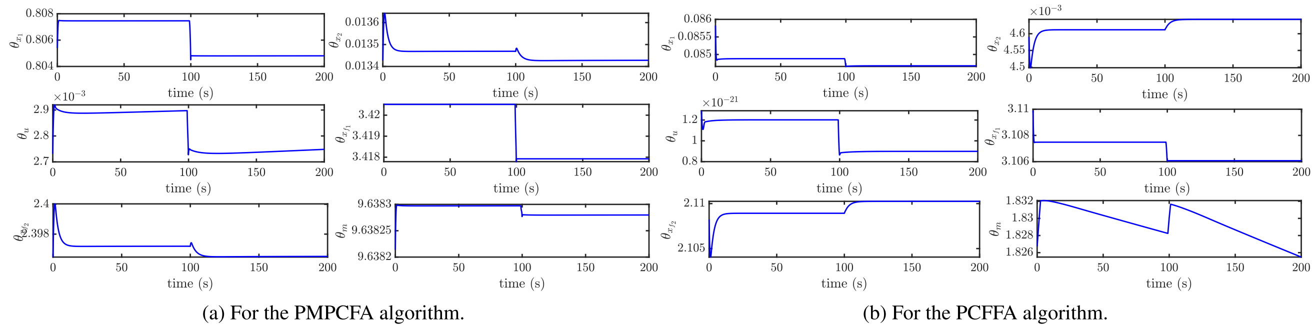
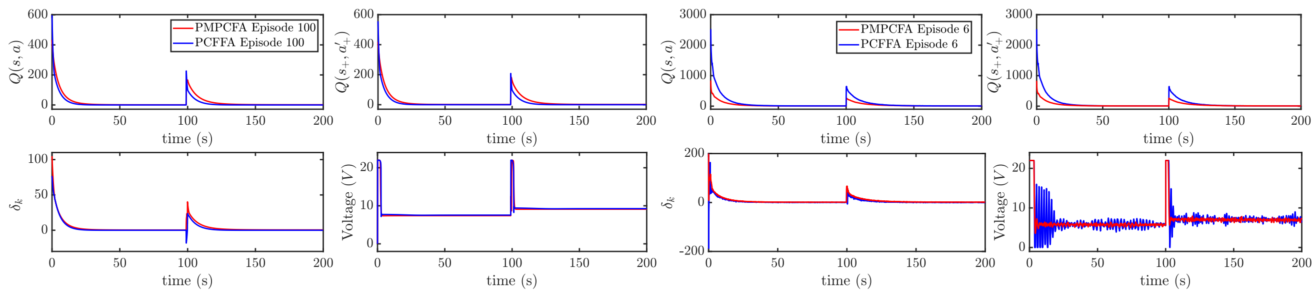


FIGURE 9. Simulation results: Learned parameters of both algorithms for the 100th episode.

of PCFFA in the 100th episode further improved as the system learns to apply the aforementioned overshoot in the height of the upper tank. Therefore, while PMPCFA can be considered trained after few episodes, the PCFFA needs more episodes. Compared with MPC-based Q-Learning, even after 100 episodes, its performance is still worse than the one observed in the first episodes of the proposed PMPCFA and PCFFA algorithms.

In order to evaluate the closed-loop performance, the sum of the baseline stage cost (rewards) in each episode for both proposed algorithms and the MPC-based Q-Learning method is shown in Fig. 7a. As discussed, in the first episode, the PMPCFA has a lower baseline stage cost due to the faster convergence of its parameters. Although this situation remains the same in the second and third episodes, the PCFFA shows a convergence to a lower value after the fourth episode. After this episode, until the 100th one, a small reduction can be seen in the sum of the baseline stage cost for the PCFFA, while this value remains more or less the same for the PMPCFA. Generally, while the PMPCFA is faster in learning, the PCFFA has a better final performance. For the Q-Learning method, the sum of its baseline stage cost in the first episode, is three times larger than the one of the proposed algorithms. Even with a sudden drop in the second episode, the convergence speed of the Q-Learning method is much lower than the one of the other two algorithms.

The parameters that are learned during the 6th and 100th episodes, for the MPC cost function, $\theta_{x_1}, \theta_{x_2}, \theta_u, \theta_{xf1}, \theta_{xf2}$, and its model, θ_m, are shown in Fig. 8a and Fig. 9a for the PMPCFA algorithm and in Fig. 8b and Fig. 9b for the PCFFA algorithm, respectively. In the PMPCFA, the parameter of the MPC model is not changed from the 6th to the 100th episode.



(a) Simulation results for the 100th episode. (b) Experimental results for the 6th episode.

FIGURE 10. Value functions of the current and next state-action, the temporal difference error, and the output of the controller for the PMPCFA and PCFFA algorithms.

On the other hand, this parameter significantly reduces over episodes for the PCFFA algorithm, which explains also the longer training process of the PCFFA algorithm. In addition to the slow convergence of the model parameter, other cost function parameters, which are tuned by the Bellman equation, need to be adapted based on the new value of the model parameter at each episode.

When comparing the PCFFA algorithm with the PMPCFA in the 100th episode shown in Figs. 9a and 9b: the ratio of the weights of both states over the weights of their final states is less for the PMPCFA, indicating that the PCFFA performs better than the PMPCFA. The weight of the input for the PCFFA is also much less, leading to a more desired tracking performance. The model parameter θ_m, estimated by the PCFFA is closer to its real value because it is obtained by system identification, while in the PMPCFA, the algorithm tries to find a combination of parameters in order to increase the expected return without taking care about the real value of the model parameter.

The value functions of the current and next state-action, the temporal difference error, and the controller output are compared for both proposed algorithms in Fig. 10a. The main difference appears when the setpoint changes; the convergence of the PCFFA is faster for the deterministic environment.

B. IMPLEMENTATION RESULTS

In this section, the PMPCFA and PCFFA algorithms are implemented on a coupled tanks system made by Quanser INC., as shown in Fig. 11. A software framework named GRAMPC, is used for implementation of the MPC [29]. As discussed in Section IV-A, the proposed algorithms lead to a straightforward implementation, because with Expected Sarsa, only one MPC needs to be designed (in contrast to the two different MPCs in the MPC-based Q-Learning method). In our algorithms, the same MPC is called twice at every sampling time, but with different initial states.

In Fig. 6b, the trajectories of both states of the real system are compared for episode 6. Similar to the simulation, the PCFFA algorithm causes the water level of the upper tank to have an overshoot in order to speed up the convergence

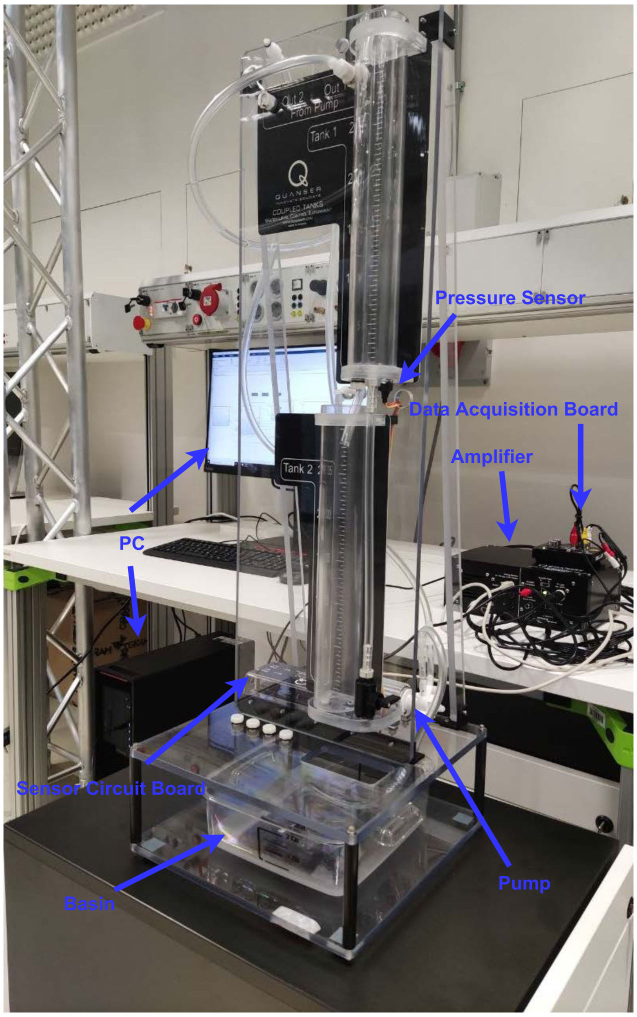
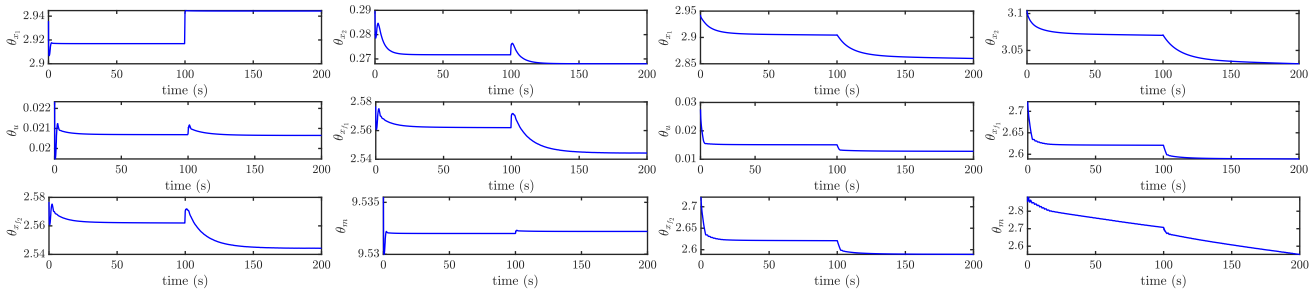


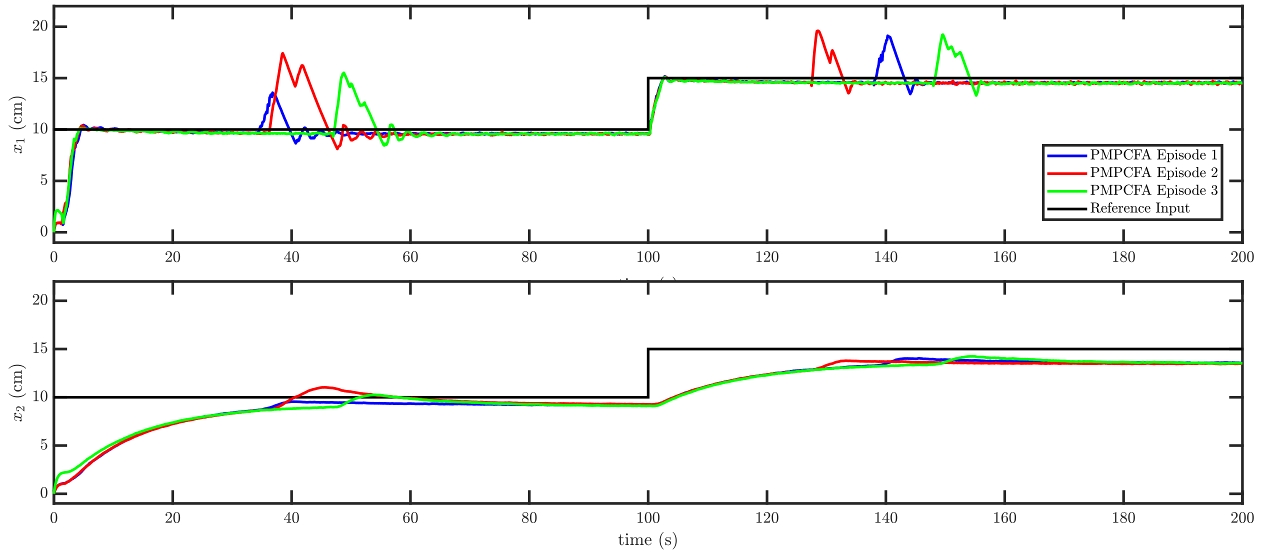
FIGURE 11. Coupled tanks system.

of the water level of the lower tank to its setpoints, but requiring fewer episodes (the same tracking performance took 100 episodes in the deterministic environment). The reason behind the steady-state error in x_2 is that in the real coupled tanks system, a delay is formed by the time required to pour the water from the upper to the lower tank, which is not considered in the simulation. Additionally,

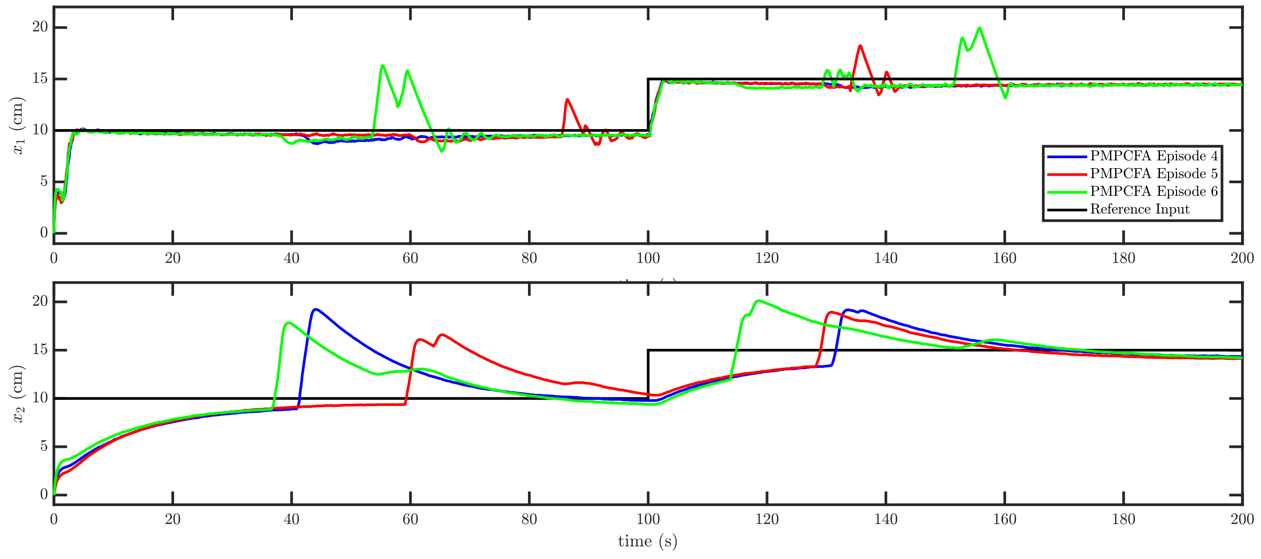


(a) For the PMPCFA algorithm. (b) For the PCFFA algorithm.

FIGURE 12. Experimental results: Learned parameters of both algorithms for the 6th episode.



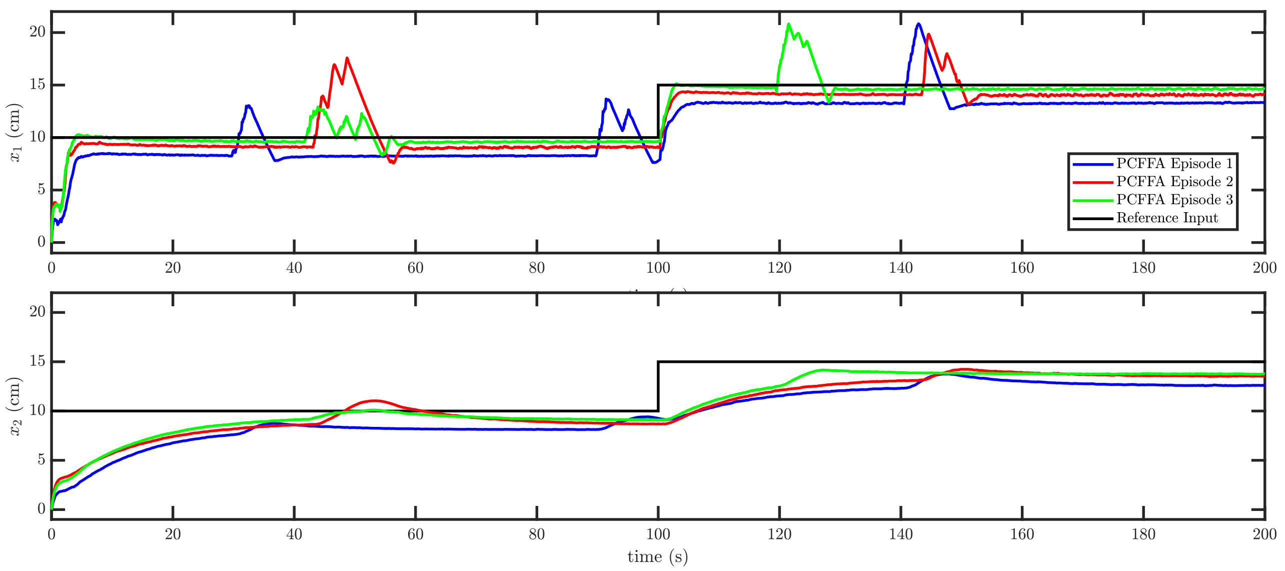
(a) Experimental results for the first, second, and third episodes.



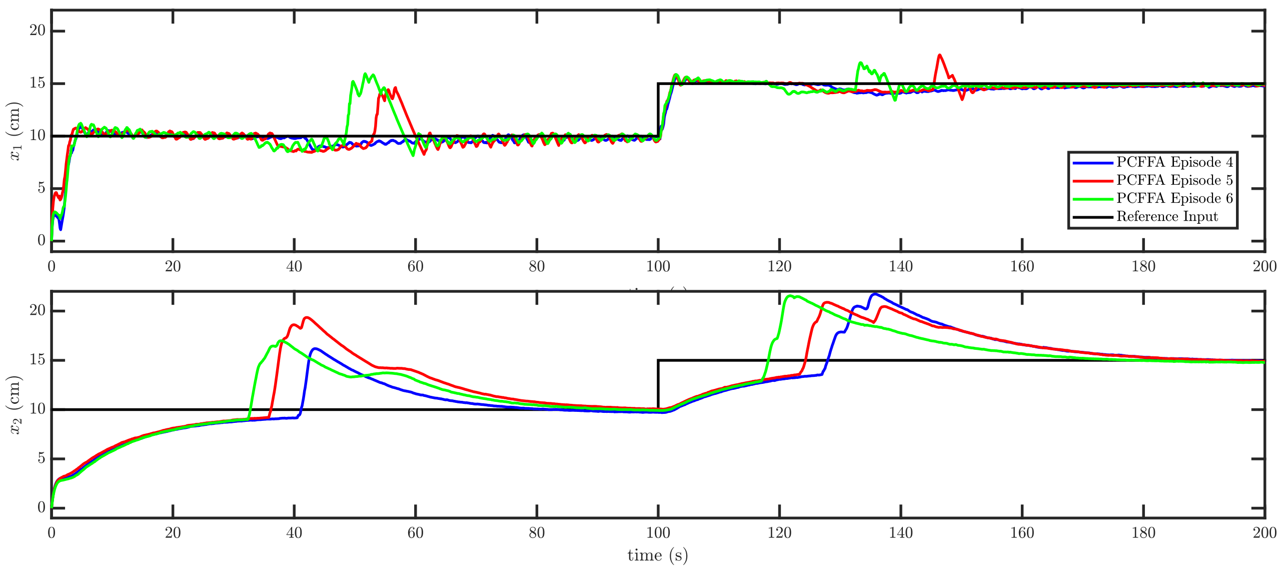
(b) Experimental results for the fourth, fifth, and sixth episodes.

FIGURE 13. Trajectories of the water levels for upper and lower tanks for PMPCFA algorithm with the existence of disturbances.

during the transfer of water from the upper to the lower tank, some water sticks to the inner surface of the lower tank without being added to the water. Regarding the PMPCFA algorithm, in the experiments (the noise perturbed



(a) Experimental results for the first, second, and third episodes.



(b) Experimental results for the fourth, fifth, and sixth episodes.

FIGURE 14. Trajectories of the water levels for upper and lower tanks for PCFFA algorithm with the existence of disturbances.

environment), the steady-state error stays the same as in the simulations.

The sum of the baseline stage cost for each episode is illustrated in Fig. 7b. The PCFFA algorithm shows the same trend but with a faster speed of convergence compared to the simulations. The rewards gradually decrease over the first 5 episodes before converging. In the PMPCFA algorithm, the learning process is completed after the first episode, while after that, only a very small reduction can be seen in the sum of the stage costs. Comparing the deterministic environment in the simulations to the perturbed environment in the real experiments, one can observe that the speed of the learning process in both PMPCFA and PCFFA algorithms increases in the experiments.

The learning process of unknown parameters is shown in Fig. 12a and Fig. 12b for the PMPCFA and PCFFA algorithms for episode 6. Since the PCFFA is able to track the setpoints for the upper tank in the 6th episode, it decreases its weight θ_{x_1}, while the weight of the term related to the lower tank θ_{x_2} gets larger to achieve a better tracking performance. For the PMPCFA algorithm, due to the steady-state error of the upper tank, its weight θ_{x_1} remains much larger than θ_{x_2} because the PMPCFA first attempts to eliminate this error. Due to the noisy behaviour of the environment, the model parameter θ_m

results to be closer to its real value in the PCFFA algorithm compared to the deterministic environment in the simulation.

The value functions of the current and next state-action, the temporal difference error, and the control output for the PMPCFA and PCFFA algorithms are compared in Fig. 10b. As can be observed, the convergence of the value functions in the PMPCFA is faster when the setpoints change which is due to the lower value of θ_{x_2} in the PMPCFA than the PCFFA. However, the temporal difference error of the PCFFA is less than that of the PMPCFA during the whole 6^{th} episode, showing that the PCFFA algorithm has a better performance in reducing the error between the current Q function and its target.

In order to show the disturbance rejection property of the proposed controllers, we added water to the upper and lower tanks during the learning phase, beginning from the first episode to the sixth one. The water levels for both tanks are shown in Fig. 13a for the first 3 episodes and for the fourth to sixth episodes in Fig. 13b, both for the PMPCFA algorithm. Not only does the algorithm show a desirable and fast disturbance rejection in all episodes, but also the steady state error still remains the same.

A similar experiment is also repeated for the PCFFA algorithm. The water level for upper and lower tanks are shown in Fig. 14a and Fig. 14b for the first and second 3 episodes, respectively. As can be seen, the algorithm reacts quickly to the disturbance and sets the water levels to their reference inputs with the desired performance. Compared to Fig. 6b, the learning speed is not affected by the disturbance and remained the same during the learning phase.

VI. CONCLUSION

In this paper, a reinforcement learning (RL) algorithm named Expected Sarsa was employed to tune a Model Predictive Controller (MPC) which was parameterized in the weights of its cost function as well as in the unknown parameters of its model. Two different implementations were compared: an algorithm called PMPCFA, where it was assumed that the whole MPC was the action value function, while the policy was to select the first element of the control output sequence as the action. In the second algorithm called PCFFA, where only the MPC cost function was defined as the action value function, the policy was to first minimize it with respect to the constraints and then to choose the first element of the control output sequence. The difference between the two algorithms was in the updating process of the parameters, whereby in the PMPCFA, all MPC parameters were updated by the Bellman equation. In the PCFFA, the MPC model was updated by an auxiliary cost function, which was defined based on the difference between the output of the MPC model and the real system, while the other parameters were found by the Bellman equation.

In deterministic simulations of a coupled tanks system, it was shown that the control performance obtained by the PCFFA algorithm was more desirable than the one of the PMPCFA algorithm. However, the training process took several more episodes compared to the PMPCFA algorithm. Both proposed algorithms were also compared with the MPC-based Q-Learning method in terms of convergence speed and final control performance. Not only was the convergence speed of both presented algorithms faster, but also their tracking performance was found to be more desirable.

In order to evaluate the performance of both proposed algorithms for a noisy environment, they were also implemented on a real coupled tanks system. While for the deterministic environment, the training of the PMPCFA resulted to be faster than for the PCFFA, in the noise perturbed environment, this effect became more pronounced and the PMPCFA resulted to be trained already properly after the first episode. On the other hand, unlike the simulations, 5 episodes were needed to complete the training process for the PCFFA algorithm in the experiments compared to 100 episodes in the simulations. Regarding the tracking performance, the noise in the real system did not have any effect on both algorithms.

In order to show the disturbance rejection capability of the proposed methods, some water was added to the upper and lower tanks during the learning phase. It was found that both algorithms can quickly reject bounded disturbance. Compared to the previous experiments, the existence of the disturbance neither affected the steady state error nor decreased the speed of the learning process for both proposed algorithms (PMPCFA and PCFFA).

All in all, the PMPCFA algorithm resulted suitable for continuous tasks already with a minimum number of episodes, while several episodes were needed for the PCFFA algorithm. In terms of the final control performance, PCFFA outperformed PMPCFA.

As a future research direction, we will investigate how to increase the exploration of the proposed algorithms to improve the performance of the closed-loop control system. This way, we hope to reduce the steady-state error that appeared in the PMPCFA algorithm without affecting the convergence speed.

REFERENCES

[1] B. Chen, Z. Huang, R. Zhang, W. Liu, H. Li, J. Wang, Y. Fan, and J. Peng, "Data-driven Koopman model predictive control for optimal operation of high-speed trains," *IEEE Access*, vol. 9, pp. 82233–82248, 2021.
[2] M. Rokonuzzaman, N. Mohajer, S. Nahavandi, and S. Mohamed, "Model predictive control with learned vehicle dynamics for autonomous vehicle path tracking," *IEEE Access*, vol. 9, pp. 128233–128249, 2021.
[3] S. Xie and J. Ren, "Linearization of recurrent-neural-network- based models for predictive control of nano-positioning systems using data-driven Koopman operators," *IEEE Access*, vol. 8, pp. 147077–147088, 2020.
[4] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits Syst. Mag.*, vol. 9, no. 3, pp. 32–50, Aug. 2009.
[5] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Syst.*, vol. 32, no. 6, pp. 76–105, Dec. 2012.
[6] S. Gros and M. Zanon, "Data-driven economic NMPC using reinforcement learning," 2019, *arXiv:1904.04152*.
[7] H. N. Esfahani, A. B. Kordabad, and S. Gros, "Reinforcement learning based on MPC/MHE for unmodeled and partially observable dynamics," in *Proc. Amer. Control Conf. (ACC)*, May 2021, pp. 2121–2126.

[8] A. B. Kordabad, H. N. Esfahani, A. M. Lekkas, and S. Gros, "Reinforcement learning based on scenario-tree MPC for ASVs," in *Proc. Amer. Control Conf. (ACC)*, May 2021, pp. 1985–1990.

[9] M. Zanon and S. Gros, "Safe reinforcement learning using robust MPC," *IEEE Trans. Autom. Control*, vol. 66, no. 8, pp. 3638–3652, Aug. 2021.

[10] H. N. Esfahani, A. B. Kordabad, and S. Gros, "Approximate robust NMPC using reinforcement learning," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2021, pp. 132–137.

[11] A. B. Martinsen, A. M. Lekkas, and S. Gros, "Combining system identification with reinforcement learning-based MPC," 2020, *arXiv:2004.03265*.

[12] A. B. Martinsen, A. M. Lekkas, and S. Gros, "Reinforcement learning-based NMPC for tracking control of ASVs: Theory and experiments," *Control Eng. Pract.*, vol. 120, Mar. 2022, Art. no. 105024.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[14] S. Gros and M. Zanon, "Reinforcement learning based on MPC and the stochastic policy gradient method," in *Proc. Amer. Control Conf. (ACC)*, May 2021, pp. 1947–1952.

[15] 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 *Proc. 60th IEEE Conf. Decis. Control (CDC)*, Dec. 2021, pp. 2990–2995.

[16] A. B. Kordabad, W. Cai, and S. Gros, "Multi-agent battery storage management using MPC-based reinforcement learning," in *Proc. IEEE Conf. Control Technol. Appl. (CCTA)*, Aug. 2021, pp. 57–62.

[17] 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 *Proc. 60th IEEE Conf. Decis. Control (CDC)*, Dec. 2021, pp. 6365–6370.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018. [Online]. Available: https://mitpress.mit.edu/books/reinforcement-learning-second-edition and http://www.incompleteideas.net/book/the-book-2nd.html

[19] M. L. Puterman, *Markov decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, Jan. 2008, pp. 1–649. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9780470316887

[20] C. Szepesvári, "Algorithms for reinforcement learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 9, pp. 1–89, Jul. 2010.

[21] H. Van Seijen, H. Van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of expected sarsa," in *Proc. IEEE Symp. Adapt. Dyn. Program. Reinforcement Learn.*, Mar. 2009, pp. 177–184.

[22] M. Zanon, S. Gros, and A. Bemporad, "Practical reinforcement learning of stabilizing economic MPC," in *Proc. 18th Eur. Control Conf. (ECC)*, Jun. 2019, pp. 2258–2263.

[23] Y. Pan, Q. Li, H. Liang, and H.-K. Lam, "A novel mixed control approach for fuzzy systems via membership functions online learning policy," *IEEE Trans. Fuzzy Syst.*, early access, Nov. 2021, doi: 10.1109/TFUZZ.2021.3130201.

[24] Y. Pan, Y. Wu, and H.-K. Lam, "Security-based fuzzy control for nonlinear networked control systems with DoS attacks via a resilient event-triggered scheme," *IEEE Trans. Fuzzy Syst.*, early access, Feb. 7, 2022, doi: 10.1109/TFUZZ.2022.3148875.

[25] S. Ohnishi, E. Uchibe, Y. Yamaguchi, K. Nakanishi, Y. Yasui, and S. Ishii, "Constrained deep Q-learning gradually approaching ordinary Q-learning," *Frontiers Neurorobotics*, vol. 13, p. 103, Dec. 2019.

[26] T. Pohlen, B. Piot, T. Hester, M. Gheshlaghi Azar, D. Horgan, D. Budden, G. Barth-Maron, H. Van Hasselt, J. Quan, M. Večerík, M. Hessel, R. Munos, and O. Pietquin, "Observe and look further: Achieving consistent performance on Atari," 2018, *arXiv:1805.11593*.

[27] 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, Mar. 2019, pp. 1–36, doi: 10.5281/zenodo.1257968.

[28] S. S. Dughman and J. A. Rossiter, "A survey of guaranteeing feasibility and stability in MPC during target changes," *IFAC-PapersOnLine*, vol. 48, no. 8, pp. 813–818, Jan. 2015.

[29] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)," *Optim. Eng.*, vol. 20, no. 3, pp. 769–809, Sep. 2019. [Online]. Available: https://link.springer.com/article/10.1007/s11081-018-9417-2

HOOMAAN MORADIMARYAMNEGARI received the bachelor's degree in mechanical engineering from Bu-Ali Sina University, in 2012, and the master's degree in aerospace engineering from the K. N. Toosi University of Technology, in 2015. He is currently pursuing the Ph.D. degree with the Faculty of Science and Technology, Free University of Bozen-Bolzano, Italy.

From 2015 to 2020, he has worked as a Researcher at the K. N. Toosi University of Technology. Since May 2020, he has been a Research Assistant with the Free University of Bozen-Bolzano. His main research interests include robust adaptive control, model predictive control, and reinforcement learning for robotic systems.

MARCO FREGO (Member, IEEE) received the M.S. degree in mathematics and the Ph.D. degree in mechatronics from the University of Trento, Trento, Italy, in 2010 and 2014, respectively. From 2014 to 2015, he was an Assistant Professor with the Hamburg University of Technology (TUHH), Hamburg, Germany, before moving to the Department of Information Engineering and Computer Science (DISI) and the Department of Industrial Engineering (DII), University of Trento. He joined the Faculty of Science and Engineering, Free University of Bolzano, in 2020. His current research interests include optimization, optimal control, and clothoid curves for the planning of autonomous and assistive robots.

ANGELIKA PEER (Member, IEEE) was born in Brunico, Italy. She received the Diploma Engineering degree in electrical engineering and information technology and the Ph.D. degree in engineering from the Technical University of Munich, Germany, in 2004 and 2008, respectively.

She was a Senior Researcher and a Lecturer at the Institute of Automatic Control Engineering and a TUM-IAS Junior Fellow with the Institute of Advanced Studies, Technical University of Munich. From 2014 to 2017, she was a Full Professor at the Bristol Robotics Laboratory, University of the West of England, Bristol, U.K. She has been a Full Professor with the Free University of Bozen-Bolzano, Italy, since November 2017.