Computers and Chemical Engineering 161 (2022) 107760

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Reinforcement learning approach to autonomous PID tuning☆

Oguzhan Dogru[a], Kirubakaran Velswamy[a], Fadi Ibrahim[a], Yuqi Wu[b], Arun Senthil Sundaramoorthy[a], Biao Huang[a,*], Shu Xu[c], Mark Nixon[c], Noel Bell[c]

[a] Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada
[b] Department of Electrical and Electronics Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada
[c] Emerson Electric Co., Austin, TX 78681, USA

A R T I C L E I N F O

Article history:
Received 2 November 2021
Revised 22 January 2022
Accepted 7 March 2022
Available online 11 March 2022

Keywords:
Contextual bandits
PID tuning
Process control
Reinforcement learning
Step-response model

A B S T R A C T

Many industrial processes utilize proportional-integral-derivative (PID) controllers due to their practicality and often satisfactory performance. The proper controller parameters depend highly on the operational conditions and process uncertainties. This study combines the recent developments in computer sciences and control theory to address the tuning problem. It formulates the PID tuning problem as a reinforcement learning task with constraints. The proposed scheme identifies an initial approximate step-response model and lets the agent learn dynamics off-line from the model with minimal effort. After achieving a satisfactory training performance on the model, the agent is fine-tuned on-line on the actual process to adapt to the real dynamics, thereby minimizing the training time on the real process and avoiding unnecessary wear, which can be beneficial for industrial applications. This sample efficient method is tested and demonstrated through a pilot-scale multi-modal tank system. The performance of the method is verified through setpoint tracking and disturbance regulatory experiments.

1. Introduction

Modern industries rely on complex processes that need to be operated efficiently and environmentally friendly. Advanced control schemes, such as model predictive controllers, provide effective solutions to achieve such goals (Blevins et al., 2013; McMillan, 1999). However, model identification for such schemes may be challenging due to ever-changing process conditions, uncertainty and complexities. Although these controllers deliver satisfactory performance, their operation also heavily depends on the PID control performance in the lower layer of the control hierarchy, which can vary over time.

PID controllers (Ziegler, 1975) provide stable, robust and simple solutions to control problems. In their simple form, PID controllers have three parameters, which reduce the design effort and hence, make them preferable over more complicated controllers. Their simplistic nature makes them applicable to numerous real-world problems (Åström, 2002). However, their parameters are process-specific and require careful and continuous tuning.

Optimality is a general term that depends on the current market demand. For example, the goal of a plant may be a function of the resource availability, and the specifications. Other factors, such as physical conditions and the age of the equipment, also determine whether these goals are achievable. In addition, some equipment (e.g., fire extinguisher systems) may need to have aggressive behaviour due to safety concerns. On the other hand, highly nonlinear, reactive or noisy systems may need smoother control policies. Thus, a universal PID parameter setting does not exist. Instead, several methods have been proposed to obtain the optimal PID parameters for the process of interest (Seborg et al., 2010).

The most primitive tuning method is trial-and-error. In this method, PID parameters can be arbitrarily adjusted according to the system response. However, this method requires skilled experts and may damage the equipment if not done carefully. Similarly, *rule-based* methods (Ziegler et al., 1942; Cohen and Coon, 1983) focus on various signal properties such as the decay ratio, settling/rise times etc. For example, Ziegler–Nichols method (Ziegler et al., 1942) pushes the system to an unstable state and adjusts the parameters based on some practical rules. On contrary, *model-based* methods (Tjokro and Shah, 1985; Rivera et al., 1986; Chien and Fruehauf, 1990; Skogestad, 2003; Åström and Hägglund, 2004) use an approximate process model (e.g., a first order plus

☆ This work was supported in part by the Natural Sciences Engineering Research Council of Canada (NSERC), and the Industrial Research Chair (IRC) Program.

* Corresponding author.
E-mail addresses: dogru@ualberta.ca (O. Dogru), velswamy@ualberta.ca (K. Velswamy), fibrahim@ualberta.ca (F. Ibrahim), yuqi14@ualberta.ca (Y. Wu), arunsent@ualberta.ca (A.S. Sundaramoorthy), biao.huang@ualberta.ca (B. Huang), richard.xu@emerson.com (S. Xu), mark.nixon@emerson.com (M. Nixon), noel.bell@emerson.com (N. Bell).

time delay (FOPTD) model Madhuranthakam et al., 2008) to adjust the aggressiveness/robustness. Though these methods may provide sufficiently good parameters, frequent tuning may still be necessary in the case of operational variations.

On-line tuning schemes have been proposed to incorporate these process variations during operation. Earlier examples include but are not limited to *continuous cycling method* (Ziegler et al., 1942), *relay auto tuning* (Åström and Hägglund, 1984), and *step test method* (Ziegler et al., 1942). These methods excite the system from a steady state to an oscillatory one, and re-tune the parameters. However, these prolonged methodologies may not be acceptable for slow (e.g., chemical) processes.

As a part of machine learning (Ge et al., 2017), reinforcement learning (RL) (Sutton and Barto, 2018; Bertsekas, 2019) has shown successful applications in various fields, such as control and monitoring recently (Mnih et al., 2015; Nian et al., 2020; Zhu et al., 2020; Xu et al., 2021; Dogru et al., 2021b; 2021a). RL formulates any task as a decision making process and represents it with an *environment*. An *agent*, within this environment, tries to maximize its performance by taking the relevant *actions* in various *states* through smart trial-and-error techniques. Since the agent learns continuously, it adapts to changes in the process, such as variations in the feed properties or operational conditions.

Owing to the above-mentioned attributes of the RL methodology, it has been employed in various process control and optimization problems (Wang et al., 2017; Pandian and Noel, 2018; Ma et al., 2019; Lambert et al., 2019; Shafi et al., 2020; Pi et al., 2020; Zhu et al., 2020; Powell et al., 2020; Li et al., 2021; Bao et al., 2021; Dogru et al., 2021c). Several works have reported the use of RL for PID tuning in simulated environments (Brujeni et al., 2010; El Hakim et al., 2013; Kofinas and Dounis, 2019; Sun et al., 2019; Shipman and Coetzee, 2019; Sedighizadeh and Rezazadeh, 2008; Brujeni et al., 2010; Carlucho et al., 2017). However, they either assume having a complete system model, propose complicated solutions, do not consider safe training, or are only applied to simulated processes. Similarly, Lawrence et al. (2020) developed an algorithm for a simulated process by using random search, which may result in frequent excitation of the system. Lawrence et al. (2021) recently developed a shallow network-based scheme to learn the PID parameters more quickly but with low exploration capability due to its incremental form. Nonetheless, in many industries, the information about the system may be limited to step test data. In addition, general RL schemes may not be used in real-time industrial applications due to safety concerns. Some examples to address the safety issue include off-line learning (Levine et al., 2020), model-based (Chang et al., 2015) and expert-based (Brown et al., 2019) methods. However, they may not be sufficient when the system model is inaccurate or the data is not sufficiently diverse. As an alternative, *constrained* RL has been proposed to limit the agent's behaviour during the learning (Altman, 1999; Zheng and Ratliff, 2020). This scheme introduces a Lagrangian penalty into the general RL goal and promotes safety on-line. Thus, it makes the RL solution more applicable to real-world problems.

As an extension to the existing non-constrained RL-based tuning schemes, this study utilizes an on-line constrained RL technique for safe and autonomous PI tuning. The proposed method also simplifies the problem formulation by relying on a contextual bandit approach without need to assume Markovian transitions of the state. Utilizing a simple step-response model makes the proposed method an easy-to-utilize tool for the practitioners. Unlike the existing methodologies, the proposed method utilizes an entropy-based systematic exploration scheme and is employed in an experimental setting to demonstrate its efficiency and real-time applicability. The multi-modal control tasks are used to showcase its effectiveness in the presence of setpoint-dependent non-stationarities.

For simplicity and considering many industrial PID controllers take PI form, this paper will focus on PI tuning. The proposed approach can be extended to a full PID controller by including one additional parameter following the same procedure. As shown in Fig. 1, the agent observes a setpoint, changes the PI parameters, and receives a reward signal depending on the performance given the new PI parameters. The agent combines multiple objective goals to provide PI tunings for any given setpoint that the users wish the PI controller to control, and explores the state space safely. The main contributions of this study are summarized as follows: Formulating the PI tuning as an on-line RL problem. Constraining the PI parameters and process variables to achieve safe operation. Modelling the system as a step-response model, and gradually learning the model plant mismatch by on-line tuning thus significantly reducing the online training time. Demonstrating the feasibility of the proposed method by using simulated and more importantly experimental case studies. Presenting the general practically through an industrial distributed control system (DCS), namely, DeltaV.

The remainder of the paper is organized as follows: Detailed background information is provided in Section 2, the proposed method is described in Section 3, the results are discussed in Section 4, and the concluding remarks are presented in Section 5.

2. Background

The goal of an RL agent is to obtain an optimal policy by means of smart trial-and-error. This involves an interaction with the environment that emits a reward during the process. In this study, an RL agent is used to tune a PI controller optimally while considering safety rules. This section explains the background of the relevant techniques in detail.

2.1. PID controllers

In its simplest form, a digital PID controller in the velocity form can be written as Seborg et al. (2010):

$$\Delta MV_t = K_c\left[(\varepsilon_t - \varepsilon_{t-1}) + \frac{\Delta t}{\tau_I}\varepsilon_t + \frac{\tau_d}{\Delta t}(\varepsilon_t - 2\varepsilon_{t-1} + \varepsilon_{t-2})\right] \tag{1}$$

$$\text{with } \varepsilon = SP - CV \tag{2}$$

where ΔMV represents the difference of the manipulated variable (MV. That is, the controller output or process input), K_c is the proportional gain, τ_I is the time constant, τ_d is the derivative constant, Δt indicates the sampling period, and ε is the difference between a setpoint (SP) and the controlled variable (CV) respectively.

Many engineering problems have noisy sensory measurements. To avoid chattering controller outputs, which may cause unstable process behaviour, the derivative term is often ignored. This simplification also reduces the number of parameters to be tuned, which can be beneficial practically. In fact, the majority of basic industrial feedback controllers are PI controllers due to their simplicity and practicality (Åström, 2002). The resulting PI controller can be written as:

$$\Delta MV_t = K_c\left[(\varepsilon_t - \varepsilon_{t-1}) + \frac{\Delta t}{\tau_I}\varepsilon_t\right] \tag{3}$$

2.2. PID tuning

In addition to the design criteria that were discussed above, optimality of PID controllers depends on their parameters. Hence, they should be found optimally and tuned continuously with an

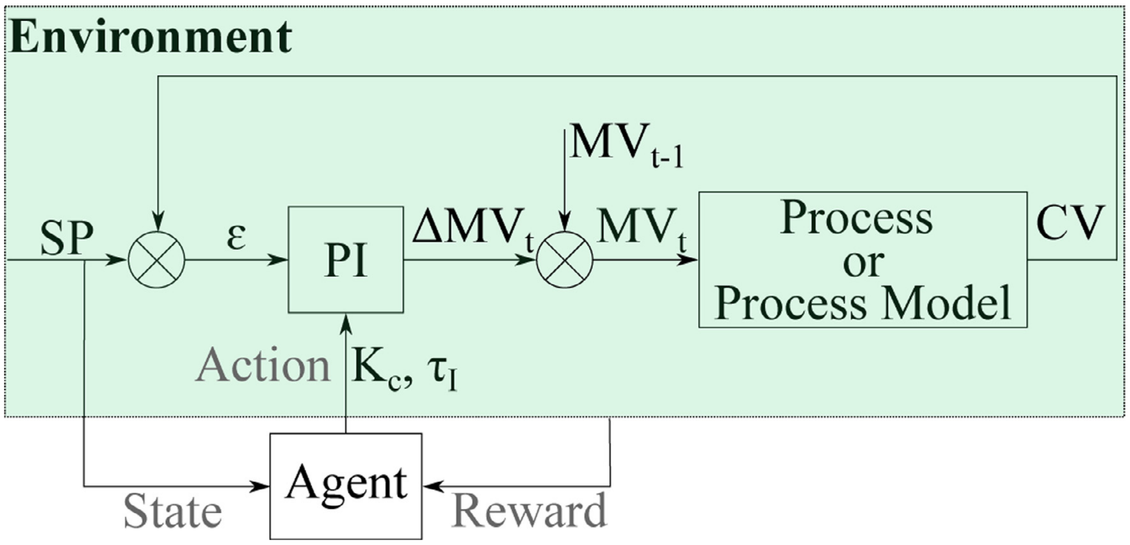

Fig. 1. Schematic of the proposed scheme. The agent observes the setpoint and adjusts the PI parameters. It calculates the cumulative error (return) at the end of an episode, and improves its policy.

effective tuning method to maximize the profit and safety. Earlier studies focused on various model-based, heuristic, and data-driven techniques (Tjokro and Shah, 1985; Rivera et al., 1986; Chien and Fruehauf, 1990; Yu et al., 2011; Skogestad, 2003; Bequette, 2003; Åström and Hägglund, 2004; Berner et al., 2018; Hägglund, 2019; Pongfai et al., 2021; Wang, 2020; Huba et al., 2020; Ulusoy et al., 2021; Pandey et al., 2021). Detailed reviews of additional techniques can be found in O'dwyer (2009); Irshad and Ali (2017); Bharat et al. (2019); Dev et al. (2020); Borase et al., 2021; Nath et al. (2021); Somefun et al. (2021). Nevertheless, identifying system models, ensuring optimal performance while using heuristic techniques, and achieving general solutions with data-driven techniques may be challenging and time consuming. RL can provide an alternative model-agnostic solution by learning how to tune the controller through smart trial-and-error. Furthermore, it allows the users to include state constraints to improve safety, which is crucial for industrial applications.

2.3. Contextual bandits

A contextual bandit agent observes a state, $x_t \in \mathcal{X}$, of an environment at time t, and takes an action $u_t \in \mathcal{U}$, where $\mathcal{X}$ and $\mathcal{U}$ are the state space and the action space respectively (Sutton and Barto, 2018). During this interaction, the agent receives a *reward* signal, $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$, by using the sensory information. Note that the subscript of the reward signal indicates that the reward is obtained after taking an action. The agent's goal is to find the best policy, $\pi^*(u|x)$, that is an optimal mapping from states to actions while maximizing the reward. Note also that contextual bandits (Sutton and Barto, 2018) simplifies the RL problem without the need to consider the Markovian state transitions.

In the presence of finite state and action spaces, the policy can be represented as a table, in which the rewards obtained for each state-action pair during the interaction are saved. However, forming a table becomes challenging for large or continuous state-action pairs due to the curse of dimensionality. An alternative approach is to parameterize the policy by θ, and optimize it directly by using the rewards. Following the policy gradient theorem (Sutton and Barto, 2018), the policy update rule for $\pi_\theta = \pi(u|x, \theta)$ of a contextual bandit agent can be written as shown in Eq. (4).

$$\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla \ln \pi(U_t|X_t, \theta_t) \tag{4}$$

where α is the learning rate, and the capital letters denote random variables. Although Eq. (4) updates θ in the direction of high reward values, converge of θ depends on α and R. Moreover, $\theta_t \neq \theta_{t+1}$ if $R \neq 0$ or $\nabla \ln \pi \neq 0$. Inspired from the actor-critic methodology (Barto et al., 1983), a parameterized baseline, $V(x, \omega)$, can be introduced to reduce the variability in θ and speed up convergence (Mnih et al., 2016). The modified policy update rule can be defined as shown in Eq. (5).

$$\theta_{t+1} = \theta_t + \alpha(R_{t+1} - V(X_t, \omega)) \nabla \ln \pi(U_t|X_t, \theta_t) \tag{5}$$

At the beginning, $V(\cdot, \omega) \neq R$ since ω is often a randomly initialized parameter set (Mnih et al., 2016). Hence, large $(R_{t+1} - V(X_t, \omega))$ results in aggressive updates in the policy parameters, θ. However, when $V(\cdot)$ converges to R, variability in θ will decrease over time (Mnih et al., 2016).

In addition to the abovementioned modifications to the policy gradient theorem, multiple worker-based asynchronous learning schemes can be used to reduce the learning duration significantly (Mnih et al., 2016) because multiple workers interact with their own environments to allow simultaneous learning while introducing data diversity. These local workers share their knowledge with a global network asynchronously during the off-line training phase, as shown in Fig. 2 (Dogru et al., 2021c; 2021a; 2021b). During on-line implementation, only the global network can be used. There are other techniques to improve learning such as utilizing multiple critics, target networks, delayed updates, etc. Fujimoto et al. (2018), which are not covered in this study.

Thus, updating the policy by using contextual bandit avoids the necessity of the Markovian state transition assumption and simplifies the learning task. In addition, using the *critic* concept of RL as a learnable baseline reduces the variance and improves convergence speed during policy learning without introducing theoretical challenges, resulting in an overall practical scheme.

3. Constrained PI tuning with on-line learning

The PI tuning task can be represented as a *contextual bandit* problem (Li et al., 2010) due to its simplicity. The goal of the PI tuning task can be defined as finding the best PI parameters by maximizing the *reward function* while respecting the constraints. The key elements for the proposed RL-based PI tuning problem can be defined as follows:

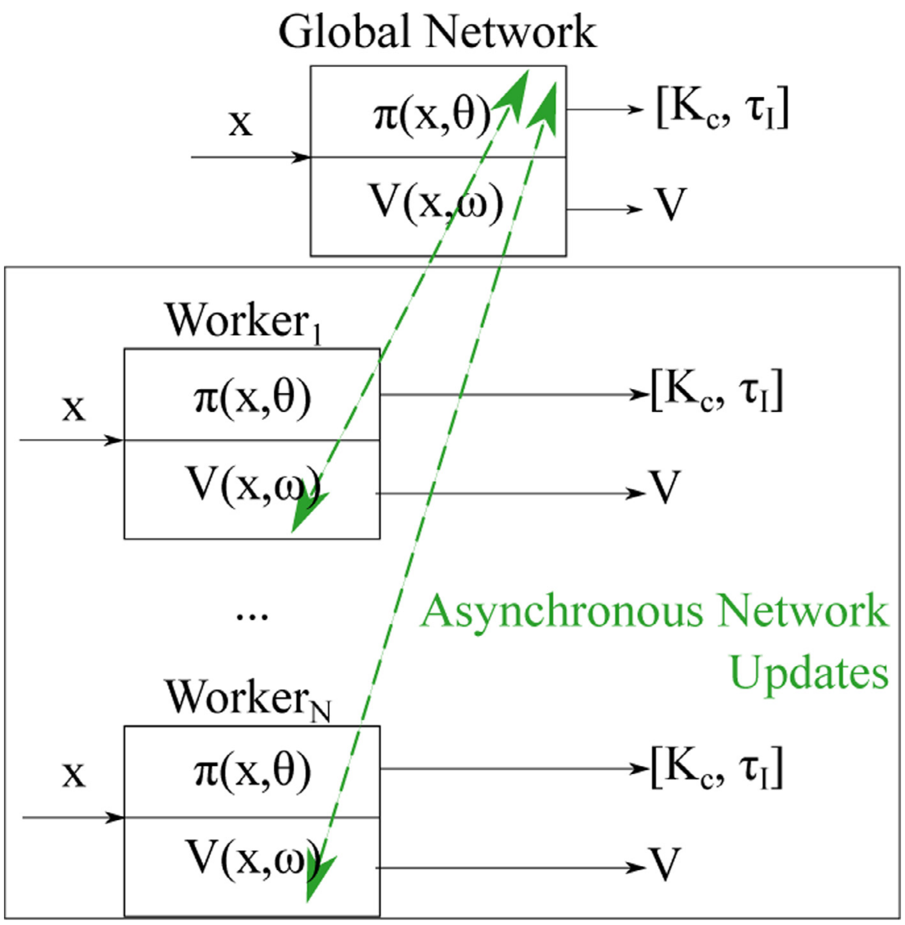

Fig. 2. The asynchronous learning scheme that promotes data diversity and increases off-line training speed. *N*-workers simultaneously interact with their environment, predict the value function, and update a global network's and their parameters according to Eqs. (13) and (14). During on-line implementation, only the global network can be used.

States: $x = SP$ or final CV. The agent observes the operating point of the process output as the state. Owing to the PI control, the operating point can also be considered as the setpoint of the PI control, which is given by a user. Note that the agent has the knowledge of the SP (which is equal to CV at steady state), which makes the agent adaptive to different setpoints and/or operational conditions.

Actions: $u = [K_c, \tau_I]^T$. The agent adjusts the PI parameters, K_c, τ_I, given observations, x.

The goal of the agent is to find the best PI parameters given a setpoint. In addition, several process constraints should be satisfied in order to address the safety concerns. To achieve this, a constrained value function optimization for the policy can be defined as follows:

$$\begin{aligned} v^*_\pi(x) &= \max_\pi v_\pi(x), \forall x \in \mathcal{X} \\ &\quad s.t. \;\; V_{\pi,C} \le \xi \end{aligned} \tag{6}$$

where v_π is the value function for π_θ, $V_{\pi,C} = \mathbb{E}_\pi[C(x)]$ is the expected constraint C, and ξ is a penalty threshold. A solution to this problem is obtained by Lagrangian relaxation as shown in Eq. (7).

$$v^*_\pi(x) = \min_{\Lambda \ge 0} \max_\theta \left[v_{\pi_\theta}(x) - \Lambda\left(V_{\pi_\theta,C} - \xi\right)\right], \forall x \tag{7}$$

where Λ is a dynamic Lagrangian coefficient with an initial value of zero and θ represents the policy parameters. Note that the goal of this method is to learn a policy quickly while learning Λ slowly to guarantee convergence. The learning goal is to find a feasible saddle point with respect to Λ and θ (Tessler et al., 2018). The value function estimation with constraints can be re-defined as:

$$\begin{aligned} V_\pi(x,\Lambda) &= \mathbb{E}_\pi[G_t|X_0 = x] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} R(X_t,U_t) - \Lambda C(X_t,U_t) | X_0 = x\right] \\ &= V - \Lambda V_{\pi,C} \end{aligned} \tag{8}$$

Note that the constrained value function ($V_\pi(x,\Lambda)$) can be estimated by using the reward function and the constraint violations only, which makes this method convenient for data-driven constraint imposition (Dogru et al., 2021c). If V ($\forall\pi$) is bounded, every local minima of $V_{\pi,C}$ is feasible, and Eq. (9) is satisfied; then, $V_\pi(x,\Lambda)$ converges to a feasible solution almost surely. Convergence analysis of the method and related proofs have been provided in Tessler et al. (2018); Borkar (2009).

$$\sum_{t=0}^{\infty}\alpha_{co,t} = \sum_{t=0}^{\infty}\alpha_{a,t} = \infty, \; \sum_{t=0}^{\infty}(\alpha^2_{co,t} + \alpha^2_{a,t}) < \infty, \; \frac{\alpha^2_{co,t}}{\alpha^2_{a,t}} \to 0 \tag{9}$$

where α_{co} and α_a are the learning rates for the Lagrangian coefficient and for the policy, respectively. Eq. (9) shows that the constraints should be learned at a lower rate compared to the policy for the value function to converge to a feasible solution. To ensure safe *exploration* of the PI controller parameters, the constraints, C, are imposed on the values of the PI parameters (K_c and τ_I), and on the process variables CV and MV as shown in Eq. (10).

$$C_i = \begin{cases} 0, & \text{if } o_{\min} \le CC \le o_{\max} \\ |CC - o_{\min}|, & \text{if } o_{\min} > CC \\ |CC - o_{\max}|, & \text{if } o_{\max} < CC \end{cases} \tag{10}$$

for $i \in \{1, 2, 3, 4\}$, with

$$o_{\min} = \begin{bmatrix} K_{c,ref} - 1 \\ \tau_{I,ref} - 1 \\ 0 \\ 0 \end{bmatrix}, CC = \begin{bmatrix} K_c \\ \tau_I \\ CV \\ MV \end{bmatrix}, o_{\max} = \begin{bmatrix} K_{c,ref} + 1 \\ \tau_{I,ref} + 1 \\ 1.1 \times SP \\ 1.1 \times \overline{MV} \end{bmatrix} \tag{11}$$

where o represents the upper and lower boundaries of exploration regions as will be discussed later in this section. $K_{c,ref}$ and $\tau_{I,ref}$ are the initial reference PI parameter values such as those calculated from certain PI tuning rules based on step-response models or values retrieved from an existing PI controller. $\overline{MV}$ is the steady state MV value at the corresponding setpoint of the output. The value 1.1 is set in Eq. (11) for illustration purpose and it can be adjusted according to the actual constraint requirements. Similarly, the value 1 is set for the PI parameter constraints and can be adjusted.

$o(1)$ and $o(2)$ constrain the PI parameters, whereas $o(3)$ and $o(4)$ regulate CV (regarding SP) and MV (according to $\overline{MV}$) respectively. Note that these are not hard constraints but they can be tightened by tuning the corresponding weights. They regularize the agent as an addition to the reward function and can provide a safer operation (Dogru et al., 2021c).

Return: G_t. Considering minimization of the deviation from the setpoint and satisfying the constraints, the sum of rewards and constraints represents the return, which can be defined as:

$$G_t = -\sum_{UI} [(CV - SP)^2 + W \begin{bmatrix} C_1, & C_2, & C_3, & C_4 \end{bmatrix}^T \Lambda] \tag{12}$$

where UI is the update interval, and W represents the weight coefficients. These values are selected empirically and can be tuned depending on the control task to obtain the desired behaviour. Although there is no state transitions in the contextual bandit setting, time step, t, is used to clearly define the parameter update rules. In an episode, the RL-level x is static, but the lower-level process variables (MV, CV) are dynamic. In this study, W will be increased during learning to provide safer operation. In Eq. (12), the first term is the integral squared error (ISE, with a fixed time step) that indicates the reward, R. The second term represents the constraints, where C_1 and C_2 keep the PI parameters within a predefined exploration region. This region is shown geometrically in Fig. 3. C_3 and C_4 constrain CV and MV respectively. Violating C results in lower returns. When converging of learning and respecting of these constraints, the final return value will consist only of ISE.

After defining the RL elements and formulating the PI tuning problem as an RL task, a major challenge is to train the agent. Data-driven models may produce general solutions, however training them may require humongous number of samples. Moreover, initial policies may not be safe or robust enough to be deployed in the real process. A solution to this challenge can be using a system model for preliminary training. However, sufficiently accurate models may not be available in practice. As an alternative to such models, less accurate step-response models can be obtained and are commonly available through a simple test, which can be used to train an initial policy. After the policy is obtained through off-line trainings, it can be further tuned on-line and improved by interacting with the real process. This methodology will reduce the training time significantly and provide effective policies without risking of damaging the equipment. The flowchart of the proposed method is illustrated in Fig. 4.

The proposed method consists of four steps. In the first step, depending on the nonlinearity of the process, two or more step-response models can be obtained, which may only be valid within

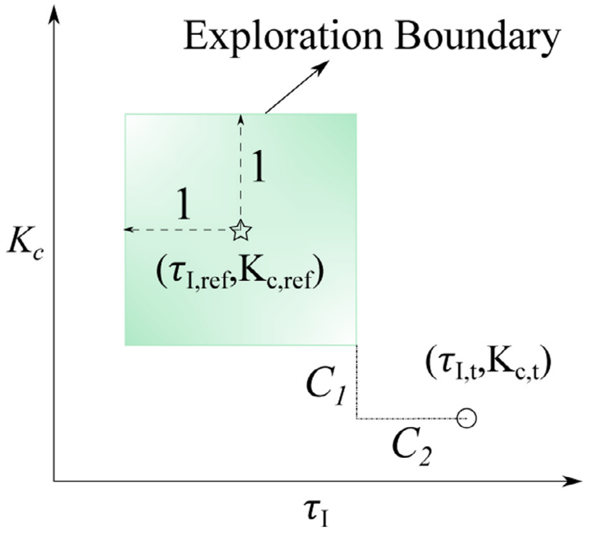

Fig. 3. The proposed safe exploration of the PI parameters. $K_{c,ref}$ and $\tau_{I,ref}$ are the pre-determined reference PI parameters. These parameters can be obtained by using a step-response model. At any time t, the agent is penalized if the PI parameters are outside the unit exploration boundary.

a limited operational range. Industrial processes are typically nonlinear or multi-modal. This is why we consider two or more of the locally approximate linear step-response models or the models corresponding to two or more of the modes for the multi-modal process. These models can also be used to obtain rough knowledge (such as a nominal control action value) about the system.

In Step 2, a PI tuner or certain tuning rules can be used to obtain a reference range for the PI parameters. Alternatively, this step may be substituted by an expert's knowledge on the process for the initial PI parameters. This study determines $[K_c, \tau_I]_{ref}$ according to one of the estimated step-response models following the IMC tuning rule (Seborg et al., 2010).

In the third step, the agent uses the reference PI parameters and the step-response models to train itself through the off-line simulations to achieve a satisfactory simulation performance. This step can be repeated until the agent passes the test criteria, which greatly saves subsequent on-line tuning time and prevents the potential risk of equipment damage due to extensive training duration.

After the policy meets the desired specifications in Step 3, it can be deployed on-line on the real physical system of interest. Since the proposed method is on-line, the agent can tune the PI controller further in real-time while updating itself to adapt to the real process dynamics. On-line tuning can occur at once if the real process is time invariant or continuously if the process has varying operating conditions that cause change of its dynamics. In this study, the agent will be on-line fine-tuned for multiple operating conditions to demonstrate the performance improvement effectively. To improve safety, the constraints term can be weighted more by increasing W at the beginning of this step. After the on-line tuning is completed, the agent will be capable to provide a suitable set of PI parameters for any setpoint given by users, as will be discussed in Sections 4.7 and 4.8.

Based on Eqs. (5) and (8), and the entropy-based exploration scheme proposed in Mnih et al. (2016), the parameter update for the proposed method is given as:

$$\omega_{t+1} \leftarrow \omega_t + \alpha_c \nabla_\omega \delta^C(\cdot|\omega)^2 \tag{13}$$

$$\theta_{t+1} \leftarrow \theta_t + \alpha_a \delta^C(\cdot|\omega) \nabla_\theta \ln \pi(\cdot, \theta) + \beta \pi(\cdot) \ln \pi(\cdot) \tag{14}$$

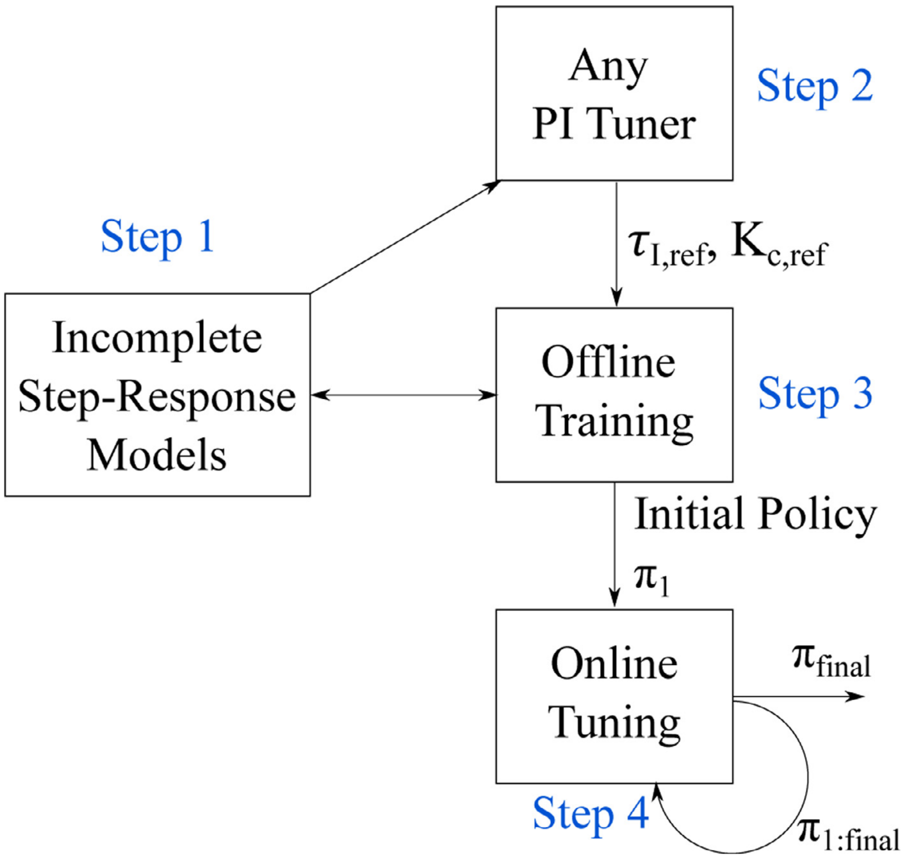

Fig. 4. The proposed training and deployment methodology. **Step 1:** Two initial, less accurate system models are obtained by using step-response tests. **Step 2:** A PI tuning method is used to obtain the initial PI parameters. **Step 3:** Off-line training through simulation is carried out to obtain the initial policy, π_1, until it satisfies the desired properties at Step 3. At **Step 4**, π_1 can be tuned on-line to learn the real system dynamics.

with $\delta^C = G_t - V(X_t|\omega) = (R_{t+1} - WC^T\Lambda - V(X_t, \omega))$ (15)

$$\Lambda_{t+1} \leftarrow \Lambda_t + \alpha_{co}(\mathbb{E}_{\pi,\Lambda_t}[C] - \xi) \tag{16}$$

where δ^C is the constrained temporal difference (TD) error for a static environment, ω represents the critic parameters. $\alpha_c = 1 \times 10^{-3}$, $\alpha_a = 1 \times 10^{-4}$ and $\alpha_{co} = 1 \times 10^{-6}$ are the learning rates of the critic, policy, and the Lagrangian coefficient respectively. $\alpha_c > \alpha_a > \alpha_{co}$ is necessary for the feasibility of the proposed solution (Tessler et al., 2018). δ is the modified TD-error, which determines the direction of the parameter update. β is the entropy coefficient that adjusts the exploration extent (the randomness degree of the policy distribution). In this study, it will be decreased at the end of off-line training to improve the stability of learning. In theory, the policy $(\pi(\cdot, \theta))$ and critic $(V(\cdot, \omega))$ can be parameterized by using various types of approximate functions (Sutton and Barto, 2018; Bishop, 2006). In this study, neural networks are utilized to capture relationships between the setpoint, value function and PI parameters that are considered to be nonlinear. As mentioned earlier, multiple workers can speed up learning; and in this study, two workers will be used during off-line training. During on-line implementation, only the global agent can be used in real control applications.

4. Results and discussion

As described in the earlier sections, the proposed method identifies step response models, obtains reference PI parameters and operational knowledge, offline trains a policy using these approximate models, and online tunes the policy in the real process. Therefore, the hyperparameters listed in Algorithm 1 (denoted as inputs) should be defined by the user before using the proposed algorithm. A user friendly guideline about their selection is provided in Table 1. The proposed algorithm tunes the PI parameters while minimizing the deviation of the response from the setpoint and respecting the operational constraints. After describing the processes in the case studies, implementation details, the network, learning results, sensitivity analysis, and simulated and experimental test results under process uncertainty and disturbances are presented in this section. To demonstrate the generality and effectiveness of the proposed method, a simulated and two experimental case studies are used in this section. These systems show multi-modal behaviour according to the setpoint. The experimental studies showcase that given reference simple models, the proposed method can be used to tune PI controllers in different settings in real-time.

4.1. *Simulated case study: a parameter varying system (PVS)*

A first-order parameter-varying system, $P(s)$, is considered and shown in Eq. (17).

$$P(s) = \frac{A_1 \sin(\Omega SP) + B_1}{(A_2 \sin(\Omega SP) + B_2)s + 1} \tag{17}$$

where $A_1 = 8$, $A_2 = 11$, $\Omega = 0.3$, $B_1 = 2$, $B_2 = 6$. The gain and the time constant of this system are a function of the setpoint, thus varying during the experiments. These variations make the system suitable to test the proposed adaptive method. The above-

Algorithm 1: PI tuning by using the proposed algorithm.

```
    input : $\alpha_a, \alpha_c, \alpha_{co}, T_{\max}, SP$ (or $SPs$), $UI$, $[K_c, \tau_I]_{ref}$, $\beta$, $W$, $\xi$
1   // Global network parameter vectors $\omega^G$, $\theta^G$, $\Lambda^G$, and global shared time-step $T \leftarrow 0$;
2   // Worker-specific parameter vectors $\omega^L$, $\theta^L$, $\Lambda^L$, and worker time-step $t \leftarrow 1$;
3   Initialize $\Lambda^G = 0$, Randomly initialize $\omega^G$ and $\theta^G$;
4   Initialize the PI parameters and the Environment ($Env.$) arbitrarily;
5   repeat
6       Reset gradients $d\omega \leftarrow 0$, $d\theta \leftarrow 0$, $d\Lambda \leftarrow 0$;
7       Sync. workers' parameters $\theta^L = \theta^G$, $\omega^L = \omega^G$, $\Lambda^L = \Lambda^G$;
8       $t_{start} = t$;
9       Uniformly sample a random setpoint, $SP \sim SPs$;
10      Observe the state $x_t = SP$;
11      Sample an action from the policy $u_t = [K_c, \tau_I]_t \sim \pi(x_t, \theta_L)$;
12      Update the controller parameters $UpdateController(u_t)$;
13      for $k = 0$ to $UI$ do
14          $\Delta MV_k = Controller()$, Controller outputs $\Delta$MV;
15          $MV_k = MV_{k-1} + \Delta MV_k$, MV is updated;
16          $Env(MV_k)$, $MV$ controls the system
17      end
18      $G_t$ = Eqn. (**??**), Receive the return.;
19      Calculate, $\delta^C$ = Eqn. (**??**) ;
20      Accumulate policy gradients wrt. $\theta^L : d\theta \leftarrow d\theta + \nabla_{\theta^L}(\log \pi(u_t|x_t, \theta^L)\delta^C + \beta\pi(\cdot)\log \pi(\cdot))$;
21      Accumulate critic gradients wrt. $\omega^L : d\omega \leftarrow d\omega + \nabla_{\omega^L}(\delta^C)^2$ ;
22  until $T > T_{\max}$;
23  Accumulate constraint difference $d\Lambda \leftarrow -(WC^T - \xi)$;
24  Asynchronously update: $\theta^G \leftarrow \theta^G + \alpha_a d\theta$, $\Lambda^G \leftarrow \Lambda^G + \alpha_{co} d\Lambda$, and $\omega^G \leftarrow \omega^G + \alpha_c d\omega$.;
25  Update $t \leftarrow t + 1$, and $T \leftarrow T + 1$output: $\pi$
```

mentioned values can be tuned to mimic the nonlinearity or change of the gain and the time constant.

Before off-line training *(Step 1)*: Two step-response models were developed by conducting step-response tests at two setpoints, $SP \in SPs_1 = \{1, 2\}$. Some gain mismatch was deliberately introduced to represent the inaccuracy of the identified step-response models. *(Step 2):* Then, the initial $[K_c, \tau_I]_{ref}$ were determined according to one of the step-response models following the IMC tuning rule (Seborg et al., 2010).

During off-line training *(Step 3)*: A random setpoint was sampled from a uniform distribution, $SP \sim U(SPs_1)$, and the agent was trained on two distinct step-response models for the agent to be trained in the environment of varying operating conditions. These models were interpolated from the two step-response tests.

This step ensures that the agent gains sufficient experience before it is deployed in real-time for on-line tuning without risking the process.

On-line tuning *(Step 4)*: During on-line tuning, a random setpoint was sampled from another uniform distribution, $SP \sim U(SPs_2 = \{4, 5\})$, to demonstrate the adaption to the real process by the proposed scheme. The gains used in this study are geometrically shown in Fig. 5. In addition, the penalty coefficient, W, was increased from 1000 to 2000 to improve safety.

4.2. Experimental study: a multi-modal, nonlinear tank system (TS)

A tank system (TS) with three modes (Dogru et al., 2021c) is shown in Fig. 6. The storage tank contains water that can be moved into Tank 1 by means of a pump. Valves V1–V2 are used to move the water into Tank 2, which changes the system dynamics according to the water level. As the level increases, the process gain decreases. This multi-modal behaviour makes the controller design more challenging. Valves V3 and V5 were kept open to avoid overflowing. Valve V4 was opened for 20 s to introduce disturbance during the test.

Before off-line training *(Step 1)*: Two step-response models were developed by conducting step-response tests with the first set of setpoints ($SP \in SPs_3 = \{2, 10\}$). These tests were deliberately chosen to mimic the industrial cases where only inaccurate models that cannot represent the complete process dynamics are available through step tests. *(Step 2):* Then, $[K_c, \tau_I]_{ref}$ were determined according to one of these models following the IMC tuning rule (Seborg et al., 2010). Note that these calculated PI parameters were used to provide an initial tuning and can be obtained by using any other PI tuning rule.

During off-line training *(Step 3)*: The inaccurate step-response models (that were interpolated from the two step-response tests) were used to train the agent within the first set of setpoints ($SP \in SPs_4 = [2, 10]$) in the off-line simulations. During off-line training, the agent learns the relationship between the switching setpoints and the PI parameters.

Generally, the agent parameters of the approximated function (e.g., the neural network) are randomly initialized and may not be sufficient to control the system of interest. This step, as proposed in this work, ensures the neural network parameters are improved before the real-time implementation. Additionally, this step can utilize multiple workers, saves training time, and prevents the equipment from a risk of getting damaged due to the initial random explorations of the agent.

On-line tuning *(Step 4)*: If the agent (π_1) is ready for the real-time deployment, the multiple worker scheme should be converted into a single one in the actual implementation as there is only one real environment available. Then, π_1 can be on-line tuned to learn the multi-modal behaviour as well as the uncertainties of the real physical system.

Table 1
User-defined parameters for the proposed algorithm.

Parameter	Definition	Data type	Guide
$\alpha_c, \alpha_a, \alpha_{co}$	Learning rates $\alpha_c > \alpha_a > \alpha_{co}$.	Floating Point (Scalars)	Smaller values mean faster convergence. Recommended values: 10^{-3}, 10^{-4}, 10^{-6}, respectively.
β	Entropy coefficient.	Floating Point (Scalar)	A larger β implies more exploration.
$T_{\max}$	Maximum number of episodes.	Integer (Scalar)	A larger $T_{\max}$ implies more training time.
UI	Number of steps in an episode.	Integer (Scalar)	Preferably close to process settling time.
$o_{\min}$, $o_{\max}$	Lower and upper limits for the constraints.	Floating Point (Vectors)	Given variable constraint.
W	Constraint weights.	Floating Point (Vector)	A larger W implies more penalty on the corresponding variable if violating its constraint.
ξ	Constraint threshold.	Floating Point (Scalar)	Tolerance to constraint violation.
SPs	Setpoint sets.	Floating Point (Vector)	Desired setpoint.
$[K_c, \tau_I]_{ref}$	Reference PI values.	Floating Point (Vector)	Existing PI controller parameters or initial tuning according to any PI tuning rule.

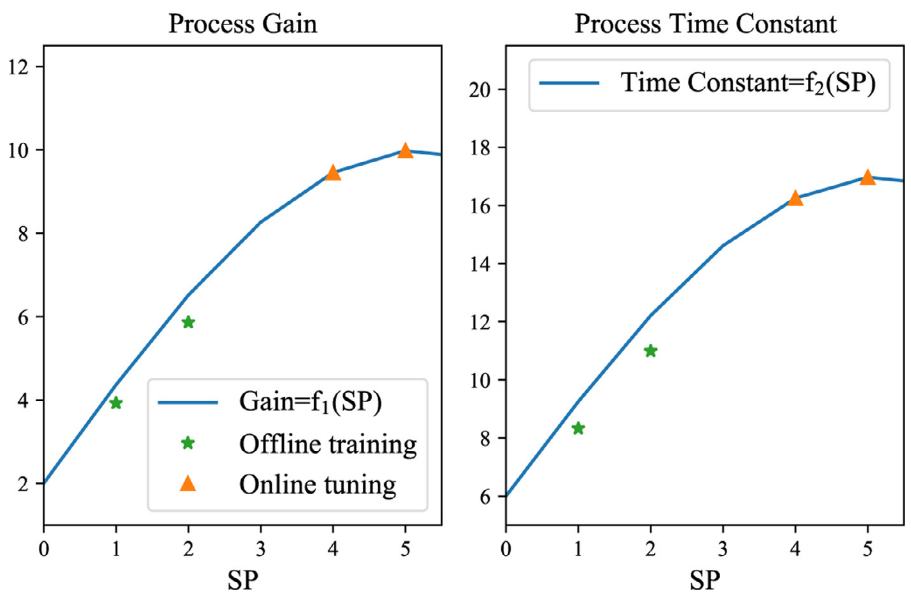

Fig. 5. Gain and time constant of PVS that is shown in Eq. (17) with respect to setpoint. The green stars and the orange triangles indicate the gains used during off-line training and on-line tuning phases respectively. Note the deviation of the stars from the true gain and time constants reflects the model mismatch between the off-line training and on-line tuning phases.

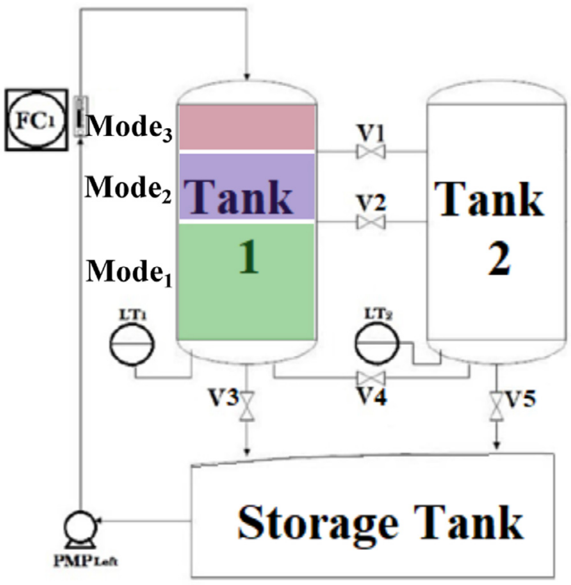

Fig. 6. P&ID of the tank system. Storage tank contains a water that is pumped into Tank 1 by using a pump. Tank 1 has three modes due to operating valves V1 and V2 that remove the water from Tank 1 into Tank 2. $\text{Mode}_1 \in [0, 15.3]$cm, $\text{Mode}_2 \in [15.3, 30.6]$cm, and $\text{Mode}_3 \in [30.6, 41.3]$cm, where each mode increase reduces the process gain. Valves V3 and V5 were kept open to create a continuous process. Valve V4 was opened to introduce disturbance during the test phase.

In this study, the agent was online-tuned by using setpoints, $SP \in SPs_5 = \{11, 12, 20\}$. Note that there is a valve below Mode_2 that reduces the process gain as shown in Fig. 6. Therefore, the system behaviour is setpoint-dependent (multi-modal) and the agent learns this multi-modality during on-line training. After on-line tuning the agent for about one day of real-time experimentation, its PI parameter suggestions were used in the test phase. The on-line tuning duration can differ (e.g., vary from 1 to 24 h) depending on the complexity of the process and uncertainties as well as how much mismatch between the step-response model and the real process and how tightly the PID is to be tuned. For the sake of diversity and to demonstrate how general the proposed method is, two agents will be trained in the TS.

4.2.1. Agent with socket connection

This agent tuned a velocity-form PI controller that was connected to the experiment directly. This implementation showcases how the agent adapts to the changes in the simplest controlling scheme as shown in Fig. 7, where the agent directly tunes the parameters of a level PI controller parameters (PI_{level}). For simplicity, the weight coefficient (W) was kept constant during online tuning while reducing β to limit the exploration. V4 was opened for 20 s to introduce disturbance during the test. This implementation will be called TS-a.

4.2.2. Agent with DeltaV connection

In contrast to the previous case, cascaded controllers are preferred in various industries to improve safety, avoid wearing of the equipment, reduction of disturbances etc. A related example is shown in Fig. 7, where the process is indirectly controlled by using a flow PI controller (PI_{flow}), which receives a reference signal from PI_{level}. In this case the 'optimum' PI_{level} parameters depend on those of PI_{flow}, therefore they will be different from those of TS-a for the same system. Although there is a broad literature about cascade controller tuning (Wang et al., 1995; Lee et al., 1998; Song et al., 2003; Jeng and Lee, 2012; Çakıroğlu et al., 2015; Manh et al., 2016; Khosravi et al., 2020; Jung et al., 2020), this work fixes the PI_{flow} parameters and tunes only PI_{level}. Therefore, the proposed method is still considered noncascade controller tuning. In addition, the controller can be in the position form. Since the agent is model-free, it can tune a PI controller in the position form as shown in Eq. (18).

$$MV_t = \overline{MV} + K_c\left[\varepsilon_t + \frac{\Delta t}{\tau_I}\sum_{j=1}^{t}\varepsilon_j\right] \tag{18}$$

As will be discussed in Section 4.3, the RL-based algorithm is used to tune a PI controller that is implemented in an industry standard

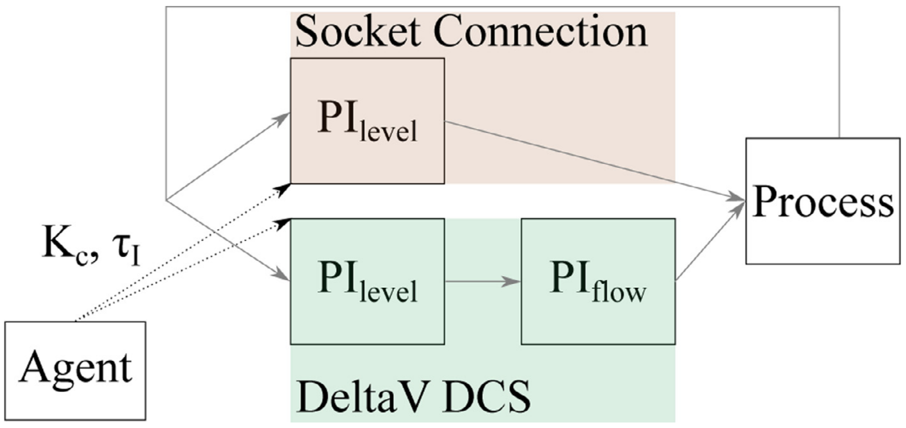

Fig. 7. PI tuning in different configurations for level control of a process. Top: Socket connection uses a velocity form PI controller. Bottom: level controller is cascaded with a flow controller to enhance safety. The cascade controlling scheme and DeltaV implementation mimics an industrial setup. The agent in both cases tune only the level controller, and these configurations demonstrate the effectiveness of the proposed method.

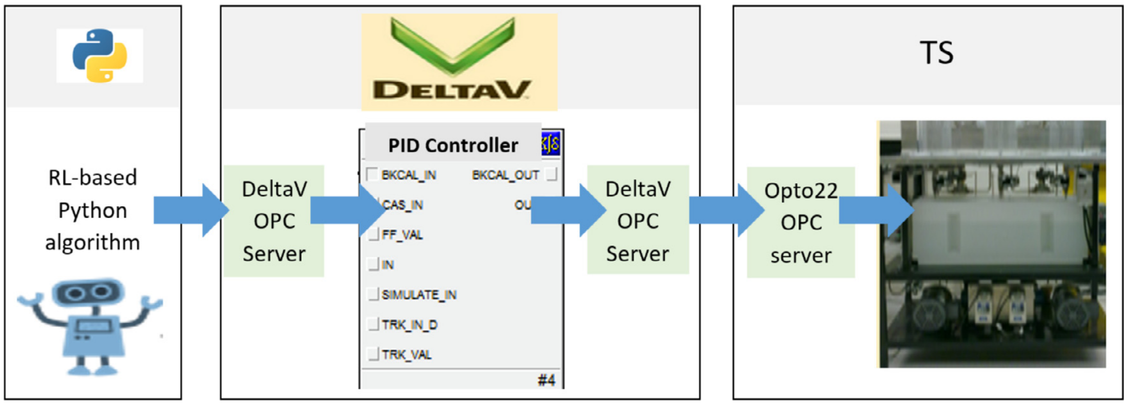

Fig. 8. The Python RL-based algorithm tunes the DeltaV-PI parameters via the DeltaV OPC server that also enables the PI to manipulate TS-b via the Opto22 OPC server.

DCS known as the DeltaV system (Yibin et al., 2005). Similar to the simulated case, W will be increased from 5000 to 10,000 while reducing β in this case study. This implementation will be called TS-b. Overall, these implementations demonstrate that the agent can tune PI controllers in various configurations, starting from a simple step-response model.

4.3. Implementation details for both simulated and experimental systems

Pseudocode of the proposed algorithm is provided in Algorithm 1.[1] As shown in Fig. 2, the algorithm consists of $N = 2$-workers that interact with their own environments during off-line training. The number of workers is limited by the available computational resources. At the on-line implementation phase, only a single worker can be used. These workers can be any differentiable function that can approximate complex functions. In this case study, the workers were feed-forward neural networks with their own setpoints to improve data diversity and exploration. The reference PI parameters, $[K_c, \tau_I]_{ref}$, can be obtained by using any method as described in Section 2.2. During off-line training, an episode consisted of $UI = 1000$ steps. The sampling time for the PI controllers of PVS and TS was one and five seconds, respectively.

[1] The source code is given in https://github.com/oguzhan-dogru/RL_PID_Tuning. The user-defined hyperparameters are summarized in the algorithm's caption and the README.md file of the source code.

After off-line training the agents in the simulated environments, the agents were on-line tuned for around a day. The on-line tuning time can vary depending on the quality of the step-response models used for the off-line training. Note that the on-line tuning was conducted on the actual process and resulted in improvements in the policy as shown in the result section.

The proposed RL-based tuning scheme is applicable to different types of industrial DCS. This study uses DeltaV as an example to illustrate the connection due to its broad applicability in the industry. The main advantages of using such DCS are that it can continuously run an entire plant at peak performance, synchronize control strategies, inputs, and outputs safely and securely while optimizing production. The proposed method can be directly plugged into the existing control schemes without major changes. The DeltaV Application station is equipped with a DeltaV Open Platform Communications (OPC) server that, on the one hand, connects the RL-based tuning algorithm to the DeltaV-PID controller and tunes its parameter via a Client-Server OPC communication protocol, as shown in Fig. 8. A Python package known as OpenOPC is used to connect the code to the DeltaV OPC server, and a Matrikon OPC UA Tunneller is used to bridge between the 32-bit DeltaV OPC server and the 64-bit Python code. On the other hand, the DeltaV OPC server enables the smartly tuned DeltaV-PID to manipulate the pumps of TS-b via another OPC server (Opto22). This connection is known as Server-to-Server OPC communication.

Real-time communication between the RL agent and the TS was directly established using a socket connection to an Opto22 server

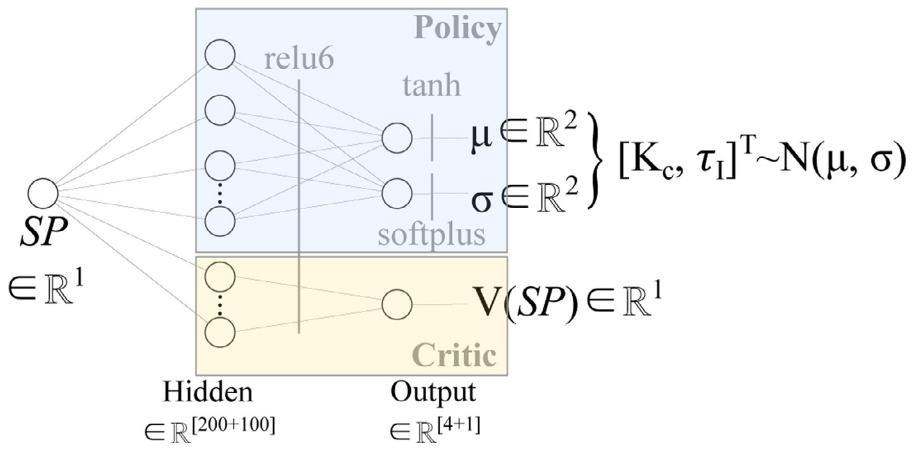
Policy
relu6
tanh
$\mu \in \mathbb{R}^2$
$\sigma \in \mathbb{R}^2$
$[K_c, \tau_I]^T \sim N(\mu, \sigma)$
softplus
$SP \in \mathbb{R}^1$
$V(SP) \in \mathbb{R}^1$
Critic
Hidden $\in \mathbb{R}^{[200+100]}$
Output $\in \mathbb{R}^{[4+1]}$

Fig. 9. A neural network structure that represents the agent. The input (*SP*) is fed into the policy and critic networks that consist of 200 and 100 neurons respectively. The outputs are fed into nonlinear ReLU6 (Sheng et al., 2018) activation function. The policy outputs a mean (that comes from tanh function) and a variance (that comes from softplus function). PI parameters are sampled from a Gaussian distribution, which is parameterized by using these mean and variance values. Critic, on the other hand, outputs a scalar value function with linear activation.

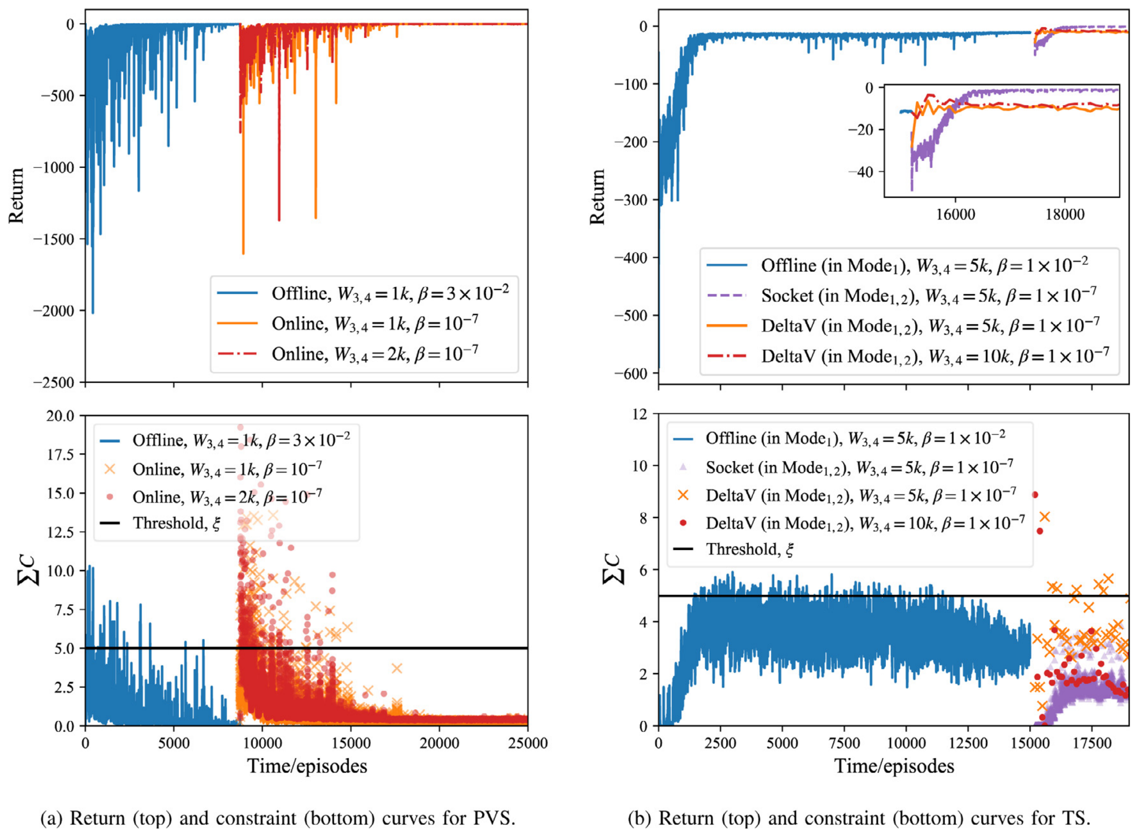

Fig. 10. Learning and constraint curves for (a) PVS and (b) TS. The entropy coefficient was decreased during learning to improve the consistency of the policy. After the off-line training phase, the entropy coefficient was set to $\beta = 10^{-7}$. Then, the agent was tuned on-line to adapt to the true system dynamics. The figures show convergence for both the cases. Note that both the case studies had different setpoints during the off-line training and on-line tuning phases that affected the system dynamics. This change in the dynamics caused reduction in the returns for both the cases.

and to the DeltaV DCS through Open Platform Communications (OPC) servers. A computer with Intel Core i5-4590 CPU (4 CPU cores) at 3.30 GHz, 8 GB RAM at 1600 MHz under 64-bit Windows 7 operating system was used for the on-line tuning phase. The offline learning phase was conducted on a Lambda deep learning workstation with an Intel Core i9-9820X CPU (10 CPU cores) at 3.30 GHz, 64 GB DDR4 RAM at 2666 MHz under 64-bit Windows 10 operating system. More information about the experimental setup can be found in Dogru et al. (2021c).

4.4. Details of the agent network

In this study, the agents consisted of deep neural networks, whose results may depend on the hyperparameters. This subsection presents the network details as well as the hyperparameters that have been used throughout the study.

Fig. 9 demonstrates the details of the network. The agent consisted of 300 hidden and 5 output neurons. The PI parameters were sampled from a Gaussian distribution that is parameterized

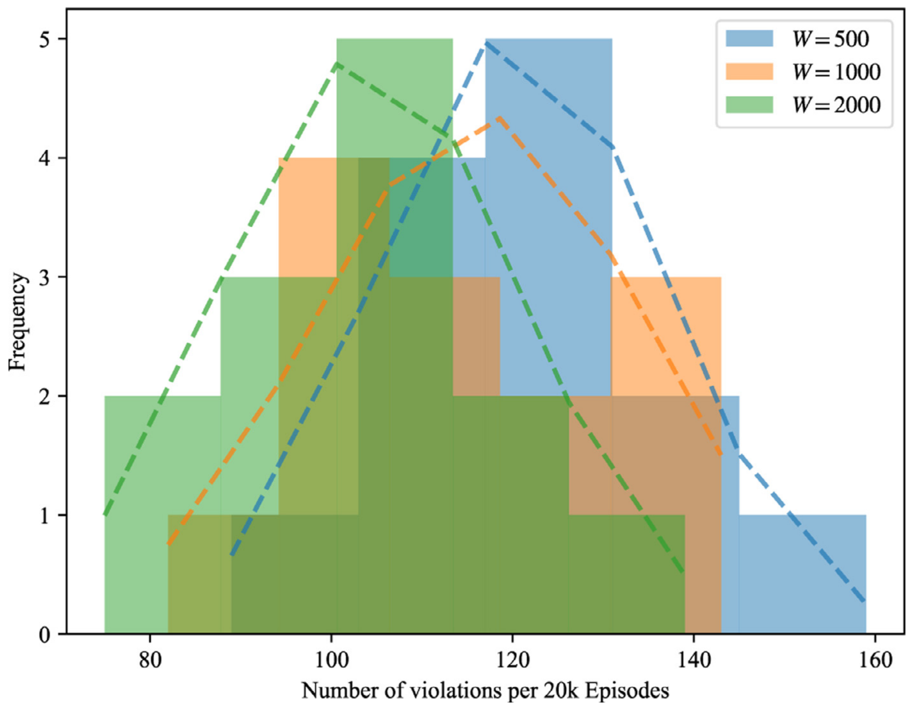

Fig. 11. Distribution of constraint violations as a function of W over twelve trials. The mean values for the constraint violation were 120, 116 and 104 for $W = 500$, 1000 and 2000 respectively. The dashed lines demonstrate the up-scaled distributions. The shift in the distribution indicates that increasing W can improve safety in terms of constraint satisfaction.

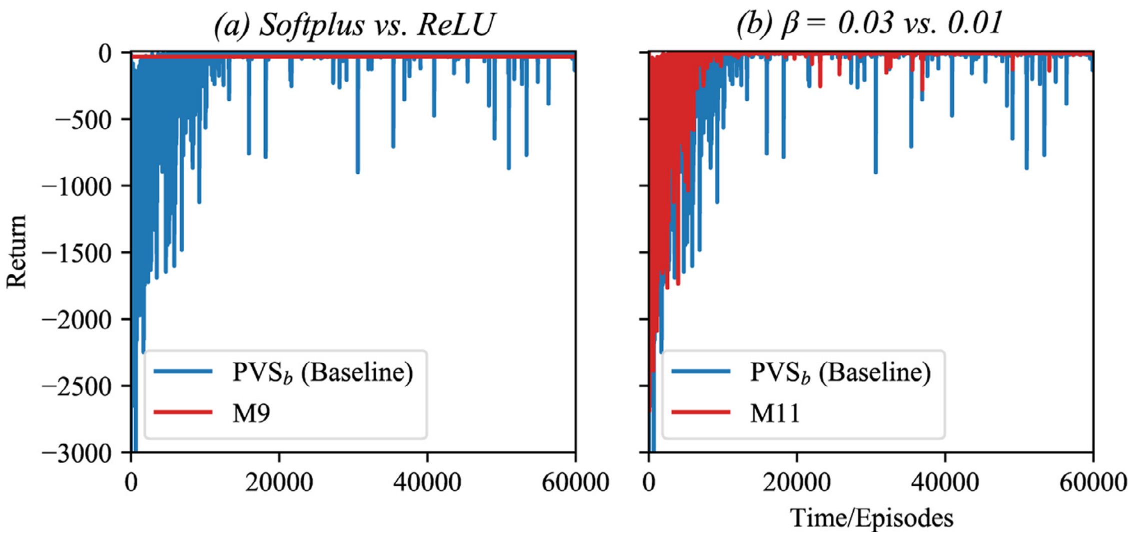

Fig. 12. Comparisons of PVS_b with (a) M9 and (b) M11. Note that the ReLU function prevented the agent from learning. On the other hand, return became more consistent as the entropy was decreased from 0.03 to 0.01. Therefore, an entropy annealing scheme was chosen during training.

by the policy outputs. The variance of this distribution decreased over time, which made the PI parameters more consistent over the time it is trained. The critic outputs a scalar that estimates the value of the setpoint (i.e., $V(SP|\omega)$). To train the agent, a modified policy gradient method (Mnih et al., 2016) was used. Alternative policy gradient based methods (e.g., deep deterministic policy gradient Lillicrap et al., 2015 or proximal policy optimization Schulman et al., 2017) can be used to obtain the PI parameters. Note that these methods may suffer from longer training time or learning instability. Because it promotes efficient learning, RMSprop (Hinton et al., 2012) algorithm was used to optimize the agent's parameters (as shown in Eqs. (13) and (14)). The above-mentioned structure and hyperparameters resulted in consistent results over a variety of trials in both PVS and TS, as will be shown in Section 4.6. Nonetheless, they can be adjusted further if the system of interest is significantly different.

4.5. Results: learning

During the learning, the entropy coefficient, β (shown in Eq. (14)), was decreased to reduce the randomness of the policy and to improve the stability of learning. As shown in Fig. 10, β was reduced at the end of off-line training, which could alternatively be reduced continuously through learning.

4.5.1. Simulated case

For the simulated system, namely, PVS, two sets of setpoints (SPs_1 and SPs_2) were used during off-line training and on-line tun-

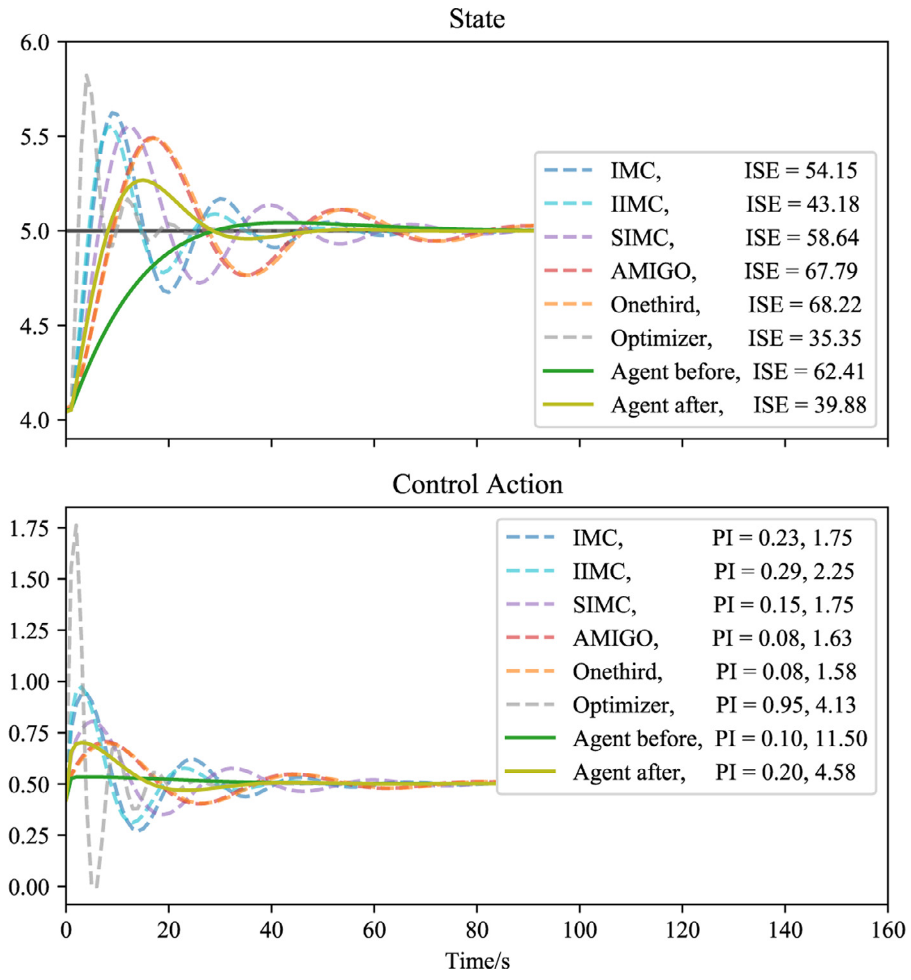

Fig. 13. Simulated step response tests for PVS at $SP = 5$ (which was used during on-line tuning). The dashed lines indicate alternative PI tuning methods, the solid lines indicate the agent. Note that the tuning parameters other than the ones that were suggested by the on-line tuned agent were obtained at $SP = 1$. The change in the setpoint caused significant oscillations for other tuning methods whereas the agent proposed PI parameters that yielded safer and smoother transitions with lower error. Note that the optimizer results in lower ISE, however, it overshoots the SP more than 10%, which is not desired. The agent suggested slower and more aggressive controller parameters, before and after on-line tuning respectively.

ing to improve the setpoint diversity. Note that the system gain and time constant were increased during on-line tuning to simulate varying system behaviour. As shown in Fig. 10a, off-line training ended at episode 8750 with a high return value. Then, the agent started interacting with the new environment (the actual process) that initially yielded lower returns owing to the mismatch between the step-response model and the actual process. After some initial exploration, the agent converged to an optimum point. At the initial phases of on-line tuning, the agent violated the constraint threshold, ξ, since the PI parameters suggested at SPs_1 were not best fit for SPs_2. After increasing W from 1000 to 2000 at the beginning of the on-line tuning phase, the violations decreased from 113 to 109. Nevertheless, the violations decreased over time for both cases as the agent adapted to the new dynamics.

As shown in Fig. 11, the mean values for the constraint violation were 120, 116 and 104 for $W = 500$, 1000 and 2000 respectively, over twelve simulations. The results show that increasing W can provide safer operation by reducing constraint violation.

4.5.2. Experimental case

For the experimental system, namely, TS, sets of setpoints SPs_4 and SPs_5 were used during off-line training and on-line tuning respectively since they correspond to different modes in the tank. As shown in Fig. 10b, off-line training ended at around episode 15,000 with a high return value. Then, the agent started interacting with the real environment that yielded lower returns (similar to the previous case). After realizing the changes in the system, the agent gradually improved its performance by finding better PI parameters while respecting its constraint threshold. Since the socket implementation (TS-a) and the offline training utilized the same control structure, the agent did not violate the constraints, however it started from lower return values compared to TS-b. On the other hand, the initial PI parameters resulted in the DeltaV implementation (TS-b), causing oscillations and constraint violations. After adapting to the process changes, the agent avoided oscillations, resulting in safer and more efficient control. Similar to the PVS, increasing W resulted in a decrease in the constraint violations from seven to two for $W = 5000$ and $W = 10,000$ respectively. This is because the cascade control structure changes the optimum PI parameters. Increasing W made the agent *regret* its aggressive actions. Note that in both the cases the agent improved its performance by respecting the constraint threshold, ξ.

Overall, the results showed that annealing entropy coefficient, β, improved the convergence. Moreover, in industrial applications, a short on-line tuning time is desired. As shown in Fig. 10, the policies converge to their optima in around a day of on-line tuning because they had been trained on the step-response model. As such, even utilizing inaccurate models still helps the agent learn the system dynamics faster, which is an appealing solution for in-

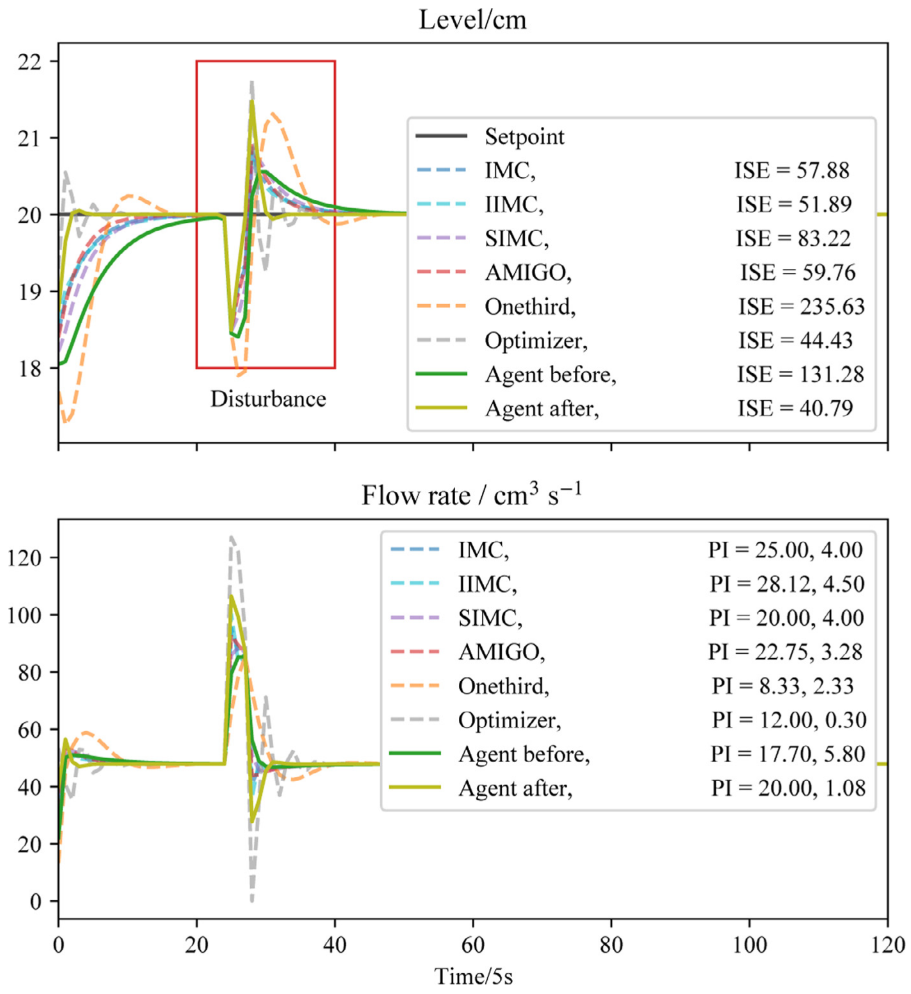

Fig. 14. Experimental step response tests for TS-a at $SP = 20$ (which corresponds to Mode$_2$ and was used during on-line tuning). The red boxes highlight the disturbance that has been introduced by using V4. The dashed lines indicate alternative PI tuning methods, the solid lines indicate the agent. Note that when the setpoint increases, the process gain and the time constant increase. Note that the compared methods and the agent before on-line tuning resulted in slower responses. The changes in the system dynamics make the agent choose higher gains and lower time constants during on-line tuning to minimize the error. C_3 helps minimizing the ISE while C_4 preventing extremely aggressive actions at $SP = 20$. Note that the agent-tuned controllers can handle the disturbance more quickly without oscillations.

dustrial applications in the sense of reducing the on-line tuning time.

4.6. Sensitivity analysis

Properties of a stochastic policy may depend on the hyperparameters of the neural networks. An experimental sensitivity analysis was performed on PVS (at the off-line training stage) to find the optimal strategy and hyperparameters to obtain a consistent policy. After creating a baseline (PVS$_b$), some of the hyperparameters were deliberately changed. This analysis was performed prior to using the agent in the on-line phase and in TS. As shown in Table 2, most of the hyperparameters did not affect the consistency significantly. Using the rectified linear unit (ReLU) function for the standard deviation of the policy prevented the agent from learning at all. On the other hand, the policy was found to be sensitive to the entropy coefficient (β). In particular, lower β values resulted in more consistent policies. This is because the entropy term regularizes the exploration of the policy. When the entropy is higher, the policy tends to take more diverse actions. For the sake of brevity, only comparisons among PVS$_b$, M9 and M11 are shown graphically. As demonstrated in Fig. 12, changing the activation function from softplus to ReLU prevented the agent from learning due to the non-smoothness of ReLU. Moreover, the agent became more consistent over time after reducing the entropy coefficient. Note that the results may vary depending on the system of concern.

4.7. Simulation case: PVS step test

Before and after the on-line tuning phase, a setpoint was fed into the agent to obtain its PI parameter suggestions. To demonstrate the effectiveness of the proposed method, a step test in the setpoint for the closed-loop process was conducted at $SP = 5$. The true system model was used during the test to mimic a practical scenario (note that inaccurate step-response models at SPs_1 were used during off-line training). During the tests, the step magnitude was standardized as one for the sake of comparability.

As shown in Fig. 13, a major result is that the agent's performance is better than the methods available in the literature (Bequette, 2003; Hägglund, 2019) in terms of error and step-response behaviour. This is because the compared methods provided PI parameters that were determined based on SPs_1. These PI parameters caused significant overshoots and oscillations at $SP = 5$. Since the agent was offline trained in the same model, its PI parameter suggestions before on-line tuning yielded a similar ISE value as the compared methods. However, the controller performance was improved after on-line tuning since the agent adapted to the changes. After this adaptation, the agent suggested PI pa-

Table 2
Simulated sensitivity analysis for PVS with varying parameters. ‘Result’ indicates whether the related hyperparameters affected the results positively (+), negatively (−), or not significantly (NS).

Agent name	Network update interval	Fixed entropy coef., β	Activation function (for σ)	Setpoint scale (SS), SP = SP/SS	Number of workers	Result
PVS_b (Baseline)	Every 10 Episodes	3×10^{-2}	softplus	10	2	
M1	10	3×10^{-2}	softplus	10	**4**	NS
M2	10	3×10^{-2}	softplus	10	**8**	NS
M3	**5**	3×10^{-2}	softplus	10	2	NS
M4	**20**	3×10^{-2}	softplus	10	2	NS
M5	10	$\mathbf{4} \times 10^{-2}$	softplus	10	2	NS
M6	10	$\mathbf{5} \times 10^{-2}$	softplus	10	2	−
M7	10	$\mathbf{1} \times 10^{-1}$	softplus	10	2	−
M8	10	3×10^{-2}	**sigmoid**	10	2	NS
M9	10	3×10^{-2}	**ReLU**	10	2	Did not learn
M10	10	$\mathbf{2} \times 10^{-2}$	softplus	10	2	+
M11	10	$\mathbf{1} \times 10^{-2}$	softplus	10	2	+
M12	10	3×10^{-2}	softplus	**5**	2	NS
M13	10	3×10^{-2}	softplus	**20**	2	NS
M14	10	$\mathbf{5} \times 10^{-3}$	softplus	10	2	+

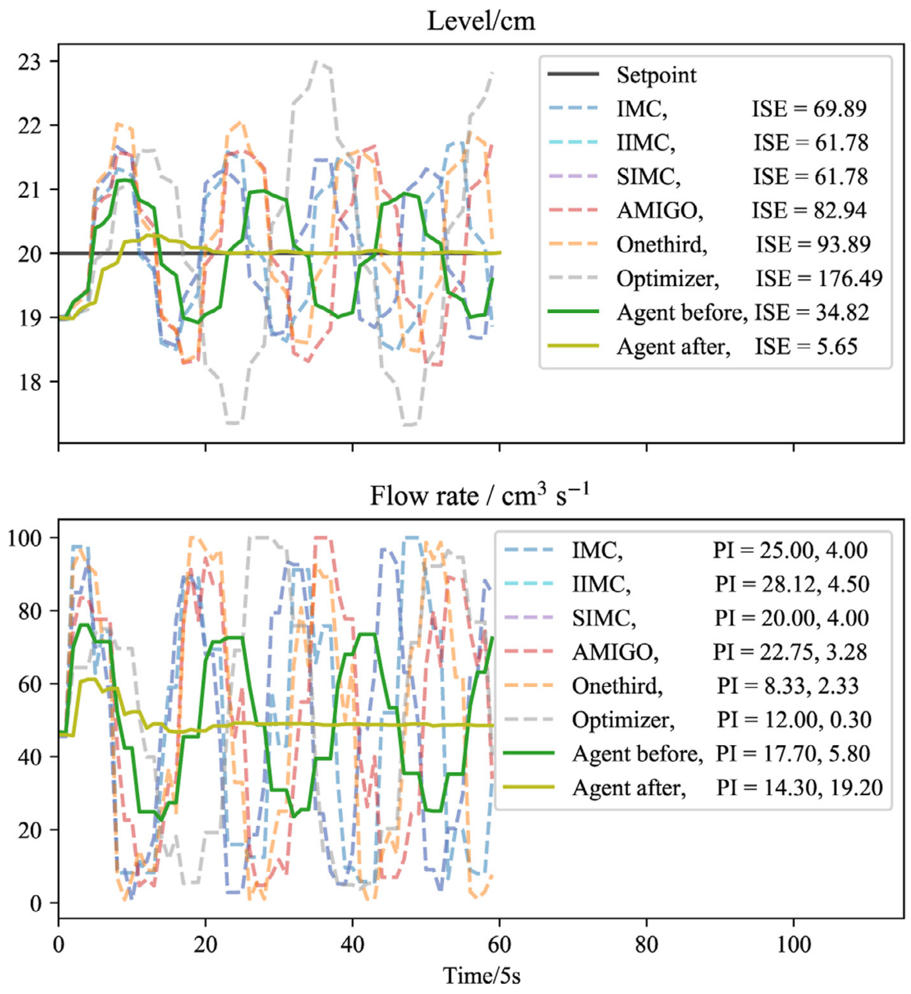

Fig. 15. Experimental step response tests for TS-b at $SP = 20$ (which corresponds to $Mode_2$ and was used during on-line tuning). The dashed lines indicate alternative PI tuning methods, the solid lines indicate the agent. Note that when the setpoint increases, the process gain and the time constant increase. Additionally these parameters were tuned for the direct control through PI_{level}, which caused oscillations when PI_{flow} was introduced. Owing to the constraints introduced during learning, the agent avoids oscillations after six hours of training.

rameters that resulted in lower error values while respecting its constraints. Additionally, the optimizer-based PI parameters resulted in lower ISE, but higher overshoots that are undesired. This is because the optimizer greedily minimizes the ISE without taking the soft constraints into consideration.

4.8. Experiment case: TS step test

In the testing phase, we mimic an actual industrial application scenario in the sense that the agent has been trained as a mature tuning expert. At this phase, one only needs to specify a desired setpoint value since the agent is able to provide appropriate PI parameters without further trial and error. Thus, during this experiment, a setpoint was given to the agent to obtain the corresponding PI parameters. The procedure was repeated before and after the on-line tuning phase to showcase the adaptiveness of the proposed method. $SP = 20$ was selected for the test phase. Besides, V4 was opened to introduce disturbance to TS-a.

As shown in Fig. 14, the agent before on-line tuning suggested PI parameters similar to the ones suggested by the compared methods. The optimization-based tuning method resulted in two oscillations due to its aggressiveness. AMIGO and IMC methods yielded similar results, whereas the one-third method resulted in a delayed overshoot. Note that during on-line tuning (in Mode1 and Mode$_2$), the decrease in the process gain results in a slower response, which increases the error if the PI parameters are not adjusted. After on-line tuning, the agent recommended more aggressive PI parameters to reduce the error at higher setpoints while respecting the constraints. This recommendation reduced the rise time significantly when the setpoint increased. In addition, C_3 helped minimizing ISE while C_4 helped avoiding aggressive actions, providing a better overall performance. Additionally, agent-tuned controllers handled the disturbance more quickly.

In contrast, as shown in Fig. 15, the initial parameters suggested by the agent and other tuning methods resulted in oscillatory behaviour because they were obtained for direct control through PI$_{\text{level}}$. Despite this difference, the agent was able to adapt to the differences introduced by PI$_{\text{flow}}$ in TS-b after six hours of on-line tuning. The results emphasize the versatility of the proposed method that can adapt to changes in the control strategy starting from simple step-response tests.

As a result of the proposed constrained PI tuning scheme, the agent improved its overall performance. These results also show that the agent adapts to process changes, and thus the proposed method can be beneficial for tuning PID for multi-modal or varying dynamic processes.

5. Conclusion

Despite ever-advancing control technologies, PID controllers are an indispensable element in industrial applications due to their practicality and simplicity. Although they provide robust and stable solutions, their parameters may require frequent tuning due to uncertainties and operation condition changes. Classical methods rely on system models or various rules to tune PID controllers, where autonomous tuning is a major challenge. This study has proposed an end-to-end methodology, starting from inaccurate step-response models, which may be the only available model for many complex processes. Following the simple tuning procedure, the initial or reference PI parameters were obtained by using an off-line tuning rule to provide a baseline for the RL agent. The proposed off-line tuning step can be convenient for many industries where the on-line tuning may be costly. At the on-line tuning phase, the agent can safely improve its performance within a shorter period by interacting with the real environments that involve sensory uncertainties. Additionally, the proposed scheme has been tested successfully on a pilot-scale experiment by using industry standard connectivity schemes with a DCS in the loop, to highlight its applicability potential. Consequently, this scheme can be beneficial for industries where continuous identification of complex processes is challenging, human tunings are expensive, and on-line learning from scratch is costly or risky. This work imposes soft constraints on the variables in the reward function. One of the future research directions is a consideration of hard constraints through a direct optimization of lower-level process variables.

Declaration of Competing Interest

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

Appendix A. PI tuning methods

As mentioned in Section 2.2, there are various PI tuning methods available in the literature. These vary from model-based methods to the data-driven ones, including heuristic and rule-based methods. In this study, six methods are used as a benchmark to compare the proposed method. These methods are easy to implement and do not require the systems to be pushed until their unstable states (unlike the methods like Ziegler–Nichols Ziegler et al., 1942; Ziegler, 1975). On the other hand, they require a system model around the operating points, which make them less attractive in the case of multi-modal systems.

The following tuning rules can be used after conducting a step-response test and obtaining the system gain ($K = \Delta y / \Delta u$), time delay (L), and the time constant ($T = \arg 0.63 y_{final}$ or 63% of the final value of the process output). A detailed information about this subsection and further references can be found in Hägglund (2019).

A1. IMC/Lambda tuning method

As one of the most commonly used methods, IMC method relies on the 63% step-response test (Dahlin et al., 1968; Higham, 1968). The rule is given as:

$$K_c = \frac{1}{K}\frac{T}{\lambda + L} \tag{19}$$

$$\tau_I = T \tag{20}$$

where λ is a tuning parameter that adjusts the smoothness of the controller. In this study, this method is used to obtain $[K_c, \tau_I]_{ref}$ by using $\lambda = T, L = 1$.

A2. IIMC

Improved IMC or IIMC method (Bequette, 2003) uses a zero-order Padé approximation for the time delay. This results in:

$$K_c = \frac{T + 0.5L}{K\lambda} \tag{21}$$

$$\tau_I = T + 0.5L \tag{22}$$

A3. SIMC

Skogestad's IMC or SIMC method (Skogestad, 2003) modifies the time constant of the IMC method as follows:

$$K_c = \frac{1}{K}\frac{T}{\lambda + L} \tag{23}$$

$$\tau_I = \min(T, 4(\lambda + L)) \tag{24}$$

Moreover, it is recommended to use $\lambda = L$ rather than $\lambda = T$ (Hägglund, 2019). Note that the above-mentioned IMC methods are similar to each other with minor adjustments to the IMC method. These adjustments have been performed to approximate the system behaviour better.

A4. AMIGO

This method minimizes the integral error by considering the 63% of the system output (Åström et al., 2006). The rule is given by:

$$K_c = \frac{1}{K}\left(0.15 + 0.35\frac{T}{L} - \frac{T^2}{(L+T)^2}\right) \tag{25}$$

$$\tau_I = 0.35L + \frac{13LT^2}{T^2 + 12LT + 7L^2} \tag{26}$$

A5. One-third

Unlike the previously discussed methods, the one-third method does not rely on the 63% response of the system. It, instead, uses 95% of the system output to provide acceptable performance in different processes (Hägglund, 2019). The controller parameters are calculated as:

$$K_c = \frac{1}{3K} \tag{27}$$

$$\tau_I = \frac{L}{3} + T \tag{28}$$

A6. Optimization-based tuning method

In addition to the above-mentioned rule-based methods, a Monte Carlo simulation is used to find the optimum PI parameters. This method is selected because it is easy to use and demonstrates an extreme case, where only ISE is minimized. The optimization-based tuning method uses the inaccurate step-response model that was used in the off-line training phase (without any constraints). Therefore, this method was expected to yield PI parameters that are significantly more aggressive. Particle-based methods can be used to find the optimum PI parameters. In this example, a brute force optimizer is designed to maximize Eq. (29).

$$J(K_c, \tau_I) = -\sum_{UI}(CV(K_c, \tau_I) - SP)^2 \tag{29}$$

Note that CV depends on the controller that uses the PI parameters, which are suggested by the optimization-based tuning method. In this study, the range for the PI parameters are chosen as $PI = [K_c, \tau_I]^T \in [\mathbf{0}, 2PI_{ref}]^T$, where $PI_{ref} = [K_c, \tau_I]_{ref}$ are the reference PI parameters obtained by the IMC rule.

CRediT authorship contribution statement

Oguzhan Dogru: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Kirubakaran Velswamy:** Investigation, Software, Writing – review & editing. **Fadi Ibrahim:** Investigation, Software, Validation, Writing – original draft, Writing – review & editing. **Yuqi Wu:** Software. **Arun Senthil Sundaramoorthy:** Writing – review & editing. **Biao Huang:** Conceptualization, Formal analysis, Funding acquisition, Methodology, Resources, Supervision, Writing – review & editing. **Shu Xu:** Formal analysis, Investigation, Writing – review & editing. **Mark Nixon:** Formal analysis, Investigation, Writing – review & editing. **Noel Bell:** Formal analysis, Investigation, Writing – review & editing.

References

Åström, K. J., 2002. Control system design.
Åström, K.J., Hägglund, T., 1984. Automatic tuning of simple regulators with specifications on phase and amplitude margins. Automatica 20 (5), 645–651.
Åström, K.J., Hägglund, T., 2004. Revisiting the Ziegler–Nichols step response method for PID control. J. Process Control 14 (6), 635–650.
Åström, K.J., Hägglund, T., Astrom, K.J., 2006. Advanced PID Control, vol. 461. Research Triangle Park, NC: ISA-The Instrumentation, Systems, and Automation Society.
Altman, E., 1999. Constrained Markov Decision Processes, vol. 7. CRC Press.
Bao, Y., Zhu, Y., Qian, F., 2021. A deep reinforcement learning approach to improve the learning performance in process control. Ind. Eng. Chem. Res. 60 (15), 5504–5515.
Barto, A.G., Sutton, R.S., Anderson, C.W., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans. Syst., Man, Cybern. (5) 834–846. Barto et al.: SMC-3.
Bequette, B.W., 2003. Process Control: Modeling, Design, and Simulation. Prentice Hall Professional.
Berner, J., Soltesz, K., Hägglund, T., Åström, K.J., 2018. An experimental comparison of PID autotuners. Control Eng. Pract. 73, 124–133.
Bertsekas, D.P., 2019. Reinforcement Learning and Optimal Control. Athena Scientific.
Bharat, S., Ganguly, A., Chatterjee, R., Basak, B., Sheet, D.K., Ganguly, A., 2019. A review on tuning methods for PID controller. Asian J. Converg. Technol. (AJCT). ISSN-2350-1146.
Bishop, C.M., 2006. Pattern Recognition and Machine Learning. Springer, New York.
Blevins, T., Wojsznis, W.K., Nixon, M., 2013. Advanced Control Foundation: Tools, Techniques and Applications. International Society of Automation (ISA).
Borase, R.P., Maghade, D.K., Sondkar, S.Y., Pawar, S.N., 2021. A review of PID control, tuning methods and applications. Int. J. Dyn. Control 9, 818–827. doi:10.1007/s40435-020-00665-4.
Borkar, V.S., 2009. Stochastic Approximation: A Dynamical Systems Viewpoint, vol. 48. Springer.
Brown, D. S., Goo, W., Nagarajan, P., Niekum, S., 2019. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. arXiv preprint arXiv:1904.06387
Brujeni, L.A., Lee, J.M., Shah, S.L., 2010. Dynamic Tuning of PI-Controllers Based on Model-free Reinforcement Learning Methods. IEEE.
Carlucho, I., De Paula, M., Villar, S.A., Acosta, G.G., 2017. Incremental q-learning strategy for adaptive PID control of mobile robots. Expert Syst. Appl. 80, 183–199.
Çakıroğlu, O., Güzelkaya, M., Eksin, I., 2015. Improved cascade controller design methodology based on outer-loop decomposition. Trans. Inst. Meas. Control 37 (5), 623–635.
Chang, S.J., Lee, J.Y., Park, J.B., Choi, Y.H., 2015. An online fault tolerant actor-critic neuro-control for a class of nonlinear systems using neural network HJB approach. Int. J. Control Autom. Syst. 13 (2), 311–318.
Chien, I.-L., Fruehauf, P.S., 1990. Consider IMC tuning to improve controller performance. Chem. Eng. Prog. 86 (10), 33–41.
Cohen, G.H., Coon, G.A., 1953. Theoretical consideration of retarded control. Trans. ASME 75, 827–834.
Dahlin, E.B., et al., 1968. Designing and tuning digital controllers. Inst. Control Syst. 41 (6), 77–83.
Dev, M.P., Jain, S., Kumar, H., Tripathi, B.N., Khan, S.A., 2020. Various tuning and optimization techniques employed in PID controller: a review. In: Proceedings of International Conference in Mechanical and Energy Technology. Springer, pp. 797–805.
Dogru, O., Chiplunkar, R., Huang, B., 2021. Reinforcement learning with constrained uncertain reward function through particle filtering. IEEE Trans. Ind. Electron. 69 (7), 7491–7499. doi:10.1109/TIE.2021.3099234.
Dogru, O., Velswamy, K., Huang, B., 2021. Actor-critic reinforcement learning and application in developing computer-vision-based interface tracking. Engineering 7 (9), 1248–1261.
Dogru, O., Wieczorek, N., Velswamy, K., Ibrahim, F., Huang, B., 2021. Online reinforcement learning for a continuous space system with experimental validation. J. Process Control 104, 86–100.
El Hakim, A., Hindersah, H., Rijanto, E., 2013. Application of reinforcement learning on self-tuning PID controller for soccer robot multi-agent system. In: 2013 Joint International Conference on rural Information & Communication Technology and Electric-Vehicle Technology (rICT & ICeV-T). IEEE, pp. 1–6.
Fujimoto, S., Hoof, H., Meger, D., 2018. Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning. PMLR, pp. 1587–1596.
Ge, Z., Song, Z., Ding, S.X., Huang, B., 2017. Data mining and analytics in the process industry: the role of machine learning. IEEE Access 5, 20590–20616.
Hägglund, T., 2019. The one-third rule for PI controller tuning. Comput. Chem. Eng. 127, 25–30.
Higham, J.D., 1968. Single-term control of first-and second-order processes with dead time. Control 12 (116), 136.
Hinton, G., Srivastava, N., Swersky, K., 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on vol. 14, 8.
Huba, M., Vrancic, D., Bistak, P., 2020. PID control with higher order derivative degrees for IPDT plant models. IEEE Access 9, 2478–2495.
Irshad, M., Ali, A., 2017. A review on PID tuning rules for SOPTD inverse response processes. In: 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT). IEEE, pp. 17–22.
Jeng, J.-C., Lee, M.-W., 2012. Identification and controller tuning of cascade control systems based on closed-loop step responses. IFAC Proc. Vol. 45 (15), 414–419.
Jung, H., Jeon, K., Kang, J.-G., Oh, S., 2020. Iterative feedback tuning of cascade control of two-inertia system. IEEE Control Syst. Lett. 5 (3), 785–790.

Khosravi, M., Behrunani, V., Smith, R.S., Rupenyan, A., Lygeros, J., 2020. Cascade control: data-driven tuning approach based on Bayesian optimization. IFAC-PapersOnLine 53 (2), 382–387.
Kofinas, P., Dounis, A.I., 2019. Online tuning of a PID controller with a fuzzy reinforcement learning MAS for flow rate control of a desalination unit. Electronics 8 (2), 231.
Lambert, N. O., Drew, D. S., Yaconelli, J., Calandra, R., Levine, S., Pister, K. S. J., 2019. Low level control of a quadrotor with deep model-based reinforcement learning. CoRR abs/1901.03737
Lawrence, N. P., Forbes, M. G., Loewen, P. D., McClement, D. G., Backstrom, J. U., Gopaluni, R. B., 2021. Deep reinforcement learning with shallow controllers: an experimental application to PID tuning. arXiv preprint arXiv:2111.07171
Lawrence, N.P., Stewart, G.E., Loewen, P.D., Forbes, M.G., Backstrom, J.U., Gopaluni, R.B., 2020. Reinforcement learning based design of linear fixed structure controllers. IFAC-PapersOnLine 53 (2), 230–235.
Lee, Y., Park, S., Lee, M., 1998. PID controller tuning to obtain desired closed loop responses for cascade control systems. Ind. Eng. Chem. Res. 37 (5), 1859–1865.
Levine, S., Kumar, A., Tucker, G., Fu, J., 2020. Offline reinforcement learning: tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643
Li, L., Chu, W., Langford, J., Schapire, R.E., 2010. A contextual-bandit approach to personalized news article recommendation. In: Proceedings of the 19th International Conference on World Wide Web, pp. 661–670.
Li, Y., Gao, W., Yan, W., Huang, S., Wang, R., Gevorgian, V., Gao, D.W., 2021. Data–driven optimal control strategy for virtual synchronous generator via deep reinforcement learning approach. J. Mod Power Syst. Clean Energy 9 (4), 919–929.
Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971
Ma, Y., Zhu, W., Benton, M.G., Romagnoli, J., 2019. Continuous control of a polymerization system with deep reinforcement learning. J. Process Control 75, 40–47.
Madhuranthakam, C.R., Elkamel, A., Budman, H., 2008. Optimal tuning of PID controllers for FOPTD, SOPTD and SOPTD with lead processes. Chem. Eng. Process. 47 (2), 251–264.
Manh, N.V., Diep, V.T., Hung, P.D., 2016. A synthesis method of robust cascade control system. J. Autom. Control Eng. 4 (2), 111–116.
McMillan, G.K., 1999. Process/Industrial Instruments and Controls Handbook. McGraw-Hill Education.
Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937.
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529.
Nath, U.M., Dey, C., Mudi, R.K., 2021. Review on IMC-based PID controller design approach with experimental validations. IETE J. Res. 1–21.
Nian, R., Liu, J., Huang, B., 2020. A review on reinforcement learning: introduction and applications in industrial process control. Comput. Chem. Eng. 139, 106886.
O'dwyer, A., 2009. Handbook of PI and PID Controller Tuning Rules. World Scientific.
Pandey, I., Panda, A., Bhowmick, P., 2021. Kalman filter and its application on tuning PI controller parameters. In: 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, pp. 1551–1556.
Pandian, B.J., Noel, M.M., 2018. Control of a bioreactor using a new partially supervised reinforcement learning algorithm. J. Process Control 69, 16–29.
Pi, C.-H., Hu, K.-C., Cheng, S., Wu, I.-C., 2020. Low-level autonomous control and tracking of quadrotor using reinforcement learning. Control Eng. Pract. 95, 104222.
Pongfai, J., Angeli, C., Shi, P., Su, X., Assawinchaichote, W., 2021. Optimal PID controller autotuning design for MIMO nonlinear systems based on the adaptive SLP algorithm. Int. J. Control, Autom. Syst. 19 (1), 392–403.
Powell, K.M., Machalek, D., Quah, T., 2020. Real-time optimization using reinforcement learning. Comput. Chem. Eng. 143, 107077.
Rivera, D.E., Morari, M., Skogestad, S., 1986. Internal model control: PID controller design. Ind. Eng. Chem. Process Des. Dev. 25 (1), 252–265.
Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347
Seborg, D.E., Mellichamp, D.A., Edgar, T.F., Doyle Rezazadeh, F.J., 2010. Process Dynamics and Control. John Wiley & Sons.
Sedighizadeh, M., Rezazadeh, A., 2008. Adaptive PID controller based on reinforcement learning for wind turbine control. In: Proceedings of World Academy of Science, Engineering and Technology, vol. 27. Citeseer, pp. 257–262.
Shafi, H., Velswamy, K., Ibrahim, F., Huang, B., 2020. A hierarchical constrained reinforcement learning for optimization of bitumen recovery rate in a primary separation vessel. Comput. Chem. Eng. 140, 106939.
Sheng, T., Feng, C., Zhuo, S., Zhang, X., Shen, L., Aleksic, M., 2018. A quantization-friendly separable convolution for mobilenets. In: 2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2). IEEE, pp. 14–18.
Shipman, W.J., Coetzee, L.C., 2019. Reinforcement learning and deep neural networks for PI controller tuning. IFAC-PapersOnLine 52 (14), 111–116.
Skogestad, S., 2003. Simple analytic rules for model reduction and PID controller tuning. J. Process Control 13 (4), 291–309.
Somefun, O.A., Akingbade, K., Dahunsi, F., 2021. The dilemma of PID tuning. Annu. Rev. Control 52, 65–74.
Song, S., Cai, W., Wang, Y.-G., 2003. Auto-tuning of cascade control systems. ISA Trans. 42 (1), 63–72.
Sun, Q., Du, C., Duan, Y., Ren, H., Li, H., 2019. Design and application of adaptive PID controller based on asynchronous advantage actor–critic learning method. Wirel. Netw. 27 (5), 3537–3547.
Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.
Tessler, C., Mankowitz, D. J., Mannor, S., 2018. Reward constrained policy optimization. arXiv preprint arXiv:1805.11074
Tjokro, S., Shah, S.L., 1985. Adaptive PID control. In: 1985 American Control Conference. IEEE, pp. 1528–1534.
Ulusoy, S., Nigdeli, S.M., Bekdaş, G., 2021. Novel metaheuristic-based tuning of PID controllers for seismic structures and verification of robustness. J. Build. Eng. 33, 101647.
Wang, F.-S., Juang, W.-S., Chan, C.-T., 1995. Optimal tuning of PID controllers for single and cascade control loops. Chem. Eng. Commun. 132 (1), 15–34.
Wang, L., 2020. PID Control System Design and Automatic Tuning Using MATLAB/Simulink. John Wiley & Sons.
Wang, Y., Velswamy, K., Huang, B., 2017. A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems. Processes 5 (3), 46.
Xu, Z., Pan, L., Shen, T., 2021. Model-free reinforcement learning approach to optimal speed control of combustion engines in start-up mode. Control Eng. Pract. 111, 104791.
Yibin, J., Xinan, S., Ming, Z., Hongyu, B., 2005. EAST cryogenic supervisory and control system based on Delta-V DCS *Plasma Science and Technology*. Plasma Sci. Tech. 7 (5), 3013.
Yu, Z., Wang, J., Huang, B., Bi, Z., 2011. Performance assessment of PID control loops subject to setpoint changes. J. Process Control 21 (8), 1164–1171.
Zheng, L., Ratliff, L. J., 2020. Constrained upper confidence reinforcement learning. arXiv preprint arXiv:2001.09377
Zhu, L., Cui, Y., Takami, G., Kanokogi, H., Matsubara, T., 2020. Scalable reinforcement learning for plant-wide control of vinyl acetate monomer process. Control Eng. Pract. 97, 104331.
Ziegler, J.G., 1975. Those magnificent men and their controlling machines. J. Dyn. Syst. Meas. Control-Trans. ASME 97, 279–280.
Ziegler, J.G., Nichols, N.B., et al., 1942. Optimum settings for automatic controllers. Trans. ASME 64 (11), 759–768.