Control Engineering Practice 121 (2022) 105046

Contents lists available at ScienceDirect

Control Engineering Practice

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

Deep reinforcement learning with shallow controllers: An experimental application to PID tuning

Nathan P. Lawrence [a,*], Michael G. Forbes [c], Philip D. Loewen [a], Daniel G. McClement [b], Johan U. Backström [d], R. Bhushan Gopaluni [b]

[a] Department of Mathematics, University of British Columbia, Vancouver BC, Canada
[b] Department of Chemical and Biological Engineering, University of British Columbia, Vancouver, BC, Canada
[c] Honeywell Process Solutions, North Vancouver, BC, Canada
[d] Backstrom Systems Engineering Ltd., Canada

ARTICLE INFO

Keywords:
Reinforcement learning
Deep learning
PID control
Process control
Process systems engineering

ABSTRACT

Deep reinforcement learning (RL) is an optimization-driven framework for producing control strategies for general dynamical systems without explicit reliance on process models. Good results have been reported in simulation. Here we demonstrate the challenges in implementing a state of the art deep RL algorithm on a real physical system. Aspects include the interplay between software and existing hardware; experiment design and sample efficiency; training subject to input constraints; and interpretability of the algorithm and control law. At the core of our approach is the use of a PID controller as the trainable RL policy. In addition to its simplicity, this approach has several appealing features: No additional hardware needs to be added to the control system, since a PID controller can easily be implemented through a standard programmable logic controller; the control law can easily be initialized in a "safe" region of the parameter space; and the final product—a well-tuned PID controller—has a form that practitioners can reason about and deploy with confidence.

1. Introduction

Reinforcement learning (RL) is a branch of machine learning that adaptively formulates a "policy" through interactions with an environment (Sutton & Barto, 2018). Such a general framework has led to the burgeoning interest in RL for process control applications (Nian, Liu, & Huang, 2020; Shin, Badgwell, Liu, & Lee, 2019). However, real-world applications in the field remain sparse. Systems integration and software development, alongside more fundamental algorithmic issues concerning closed-loop stability and sample efficiency are all formidable challenges one faces.

We take a pragmatic approach to implementing RL. Proportional–integral–derivative (PID) controllers are well-understood and ubiquitous in practice. This makes the problem of tuning a PID a reasonable starting point for real-world applications of RL. Moreover, their simple structure and stabilizing properties are neatly compatible with policy design in RL. Further, the prevalence of PID controllers in current industrial plants makes it natural to seek RL applications that operate effectively with them and thus take full advantage of existing hardware and expertise. Therefore, this framework provides a fruitful testbed for evaluating RL algorithms through the lens of industrially-accepted auto-tuning methods.

This paper extends the previous work of Lawrence et al. (2020) to a physical system; the key idea is to interpret a PID controller as the RL policy. We embrace the fact that performance is not the only metric of interest when evaluating a new approach to controller design (Åström & Hägglund, 1984; Forbes, Patwardhan, Hamadah, & Gopaluni, 2015). Other important considerations include *ease of use*, *maintainability*, and *robustness*. We convey the technical challenges involved in implementing a RL algorithm in our lab: this includes special consideration of the interplay between software development and existing hardware. Moreover, we propose various metrics for evaluating RL algorithms with existing auto-tuners serving as a baseline. We aim to show that RL can be deployed on a physical system in an interpretable and modular fashion, and to weigh the merits of RL against standard tuning techniques. Ultimately, the purpose of this work is to develop a guide for real-world applications of RL in process control.

1.1. Related work

We review some related work at the intersection of RL and process control. For a more thorough overview the reader is referred to the survey papers by Lee, Shin, and Realff (2018) and Shin et al. (2019), or the tutorial-style papers by Nian et al. (2020) and Spielberg, Tulsyan,

* Corresponding author.
E-mail addresses: lawrence@math.ubc.ca (N.P. Lawrence), bhushan.gopaluni@ubc.ca (R.B. Gopaluni).

https://doi.org/10.1016/j.conengprac.2021.105046
Received 2 August 2021; Received in revised form 11 November 2021; Accepted 16 December 2021
Available online 12 January 2022

Lawrence, Loewen, and Bhushan Gopaluni (2019). Moreover, since PID control is such a large field, we limit our survey of PID control to RL-related methods; some data-driven and optimization-based approaches can be found in the works of Berner, Soltesz, Hägglund, and Åström (2018) and Wakitani, Yamamoto, and Gopaluni (2019), and others they cite.

One of the first studies of reinforcement learning for process control applications is due to Hoskins and Himmelblau (1992). Later, in the early 2000s, several successful implementations of RL methods in process control were developed. For example, Kaisare, Lee, and Lee (2003) and Lee and Lee (2008) utilize approximate dynamic programming methods for optimal control of discrete-time nonlinear systems. These early works demonstrate the applicability of RL in process control through applications such as scheduling problems or control of a microbial cell reactor when a simulation model is available.

More recently, there has been significant interest in deep RL methods for process control (Cui, Zhu, Fujisaki, Kanokogi, & Matsubara, 2018; Dogru, Wieczorek, Velswamy, Ibrahim, & Huang, 2021; Ge, Li, & Chang, 2018; Ma, Zhu, Benton, & Romagnoli, 2019; Noel & Pandian, 2014; Pandian & Noel, 2018; Syafiie, Tadeo, Martinez, & Alvarez, 2011). Spielberg et al. (2019) adapted the popular model-free deep deterministic policy gradient (DDPG) algorithm for setpoint tracking problems. Meanwhile, Wang, Velswamy, and Huang (2018) developed a deep RL algorithm based on proximal policy optimization algorithm (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). Petsagkourakis, Sandoval, Bradford, Zhang, and del Rio-Chanona (2020) use transfer learning to adapt a policy developed in simulation to novel systems. Variations of DDPG, such as twin-delayed DDPG (TD3) (Fujimoto, Hoof, & Meger, 2018) or a Monte-Carlo based strategy have also shown promising results in complex control tasks (Joshi, Makker, Kodamana, & Kandath, 2021; Yoo, Kim, Kim, & Lee, 2021). Other approaches utilize meta-learning and apprenticeship learning to quickly adapt trained RL models to new processes (McClement et al., 2021; Mowbray, Smith, Del Rio-Chanona, & Zhang, 2021). In the model-based setting, Kim et al. (2020) incorporate deep neural networks (DNNs) as value function approximators into the globalized dual heuristic programming algorithm. Predictive models have also been augmented with popular DRL algorithms, such as DDPG or TD3, to improve the policy gradient estimation (Bao, Zhu, & Qian, 2021).

Other approaches to RL-based control postulate a fixed control structure such as PID (Carlucho, De Paula, Villar, & Acosta, 2017; Sedighizadeh & Rezazadeh, 2008; Shipman & Coetzee, 2019). Brujeni, Lee, and Shah (2010) develop a model-free algorithm to dynamically select the PID gains from a pre-defined collection derived from Internal Model Control (IMC). Their approach is adapted to a physical continuously stirred tank heater after pre-training in simulation. Berger and da Fonseca Neto (2013) dynamically tune a PID controller in continuous parameter space using the actor–critic method; their approach is based on dual heuristic dynamic programming, where an identified model is assumed to be available. The actor–critic method is also employed by Sedighizadeh and Rezazadeh (2008), where the action at each time-step is to revise the PID gains.

Only a handful of these studies on RL for process control contain real-world validation (Brujeni et al., 2010; Dogru et al., 2021; Nian et al., 2020; Pandian & Noel, 2018; Syafiie et al., 2011). This work is focused on evaluating the merits of deep RL as a model-free auto-tuning strategy. This entails a couple significant differences from previous work: We do not use a simulation model or offline datasets for pre-training the RL agent; We outline criteria for any auto-tuning strategy and evaluate our deep RL algorithm through this lens. Crucially, we provide a detailed account of the hardware, software, and algorithmic details involved in implementing the RL agent.

2. Methodology

2.1. Reinforcement learning

A brief overview of deep RL will serve to fix our notation, which is largely standard. For more background, see Sutton and Barto (2018); tutorial-style treatments are given by Nian et al. (2020) and Spielberg et al. (2019).

The system of interest has states s that evolve in some set $\mathcal{S}$. At each instant t (an integer), the agent observes the state s_t and responds by applying some action a_t, chosen from a given set $\mathcal{A}$. The system dynamics then produce a new state s_{t+1}, and the agent receives a scalar reward, r_t. As time marches on, a history $h = (s_0, a_0, s_1, a_1, \ldots)$ is produced. The present value of the agent's total accumulated rewards is

$$\sum_{t=1}^{\infty} \gamma^{t-1} r_t,$$

where $\gamma \in (0, 1]$ is a fixed discount factor.

Randomness complicates the situation. Given the current state s_t and action a_t, the new state s_{t+1} is a random variable distributed according to some density $p\left(\cdot \middle| s_t, a_t\right)$. (Typically p, which encodes the system's nonlinear stochastic dynamics, is taken as completely unknown.) More formally, the system is assumed to be a Markov Decision Process (MDP). The agent's reward emerges from a fixed function r, according to $r_t = r(s_t, a_t)$. The agent's actions are determined by selecting a "policy" π, which is a state-dependent conditional probability density on $\mathcal{A}$, so that the agent's actions obey $a_t \sim \pi\left(\cdot \middle| s_t\right)$.

By fixing a policy π, the agent completes the specification of a Markov process and establishes a probability density $p^{\pi}(\cdot)$ on the set of trajectories $h = (s_0, a_0, s_1, a_1, \ldots)$. The agent's goal is to maximize expected long-term reward.[1] by choosing the best possible policy π, that is,

$$\begin{aligned} &\text{maximize} \quad J(\pi) = \mathbb{E}_{h\sim p^{\pi}(\cdot)}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t)) \middle| s_0\right] \\ &\text{over all} \quad \text{policies } \pi : \mathcal{S} \to \mathcal{P}(\mathcal{A}), \end{aligned} \tag{1}$$

where $\mathcal{P}(\mathcal{A})$ denotes the set of probability measures on $\mathcal{A}$.

The broad subject of reinforcement learning concerns iterative methods for choosing a desirable policy π (this is the "learning"), guided in some fundamental way by the agent's observations of the rewards from past state–action pairs (this provides the "reinforcement").

We next outline the class of algorithms aimed at solving Problem (1). Not knowing p (or, consequently, p^{π}) is a key limitation. We focus exclusively on methods that produce *deterministic* policies μ, which can be considered as simple $\mathcal{A}$-valued mappings defined on the state space. Henceforth we use the notation μ for a policy; in the context of the above formulation one may set $\pi = \mu$.

Common approaches to solving Problem (1) involve Q-learning (value-based methods) and the policy gradient theorem (policy-based methods) (Sutton & Barto, 2018). These methods form the basis for deep RL algorithms, that is, algorithms for solving RL tasks with the aid of deep neural networks. Deep neural networks act as flexible function approximators, well-suited for expressing complex control laws. Moreover, function approximation methods make RL problems tractable in continuous state and action spaces (Lillicrap et al., 2015; Silver et al., 2014; Sutton, McAllester, Singh, & Mansour, 1999). Without them, discretization of the state and action spaces is necessary, accentuating the "curse of dimensionality".

[1] Standard RL terminology calls for *maximizing a reward* The problem can be recast as *minimizing a loss* by changing the sign of the objective. Thus $-r_t$ has a role analogous to the stage cost in MPC; when we discuss a reward function with reference to a cost function, this change of sign is understood implicitly.

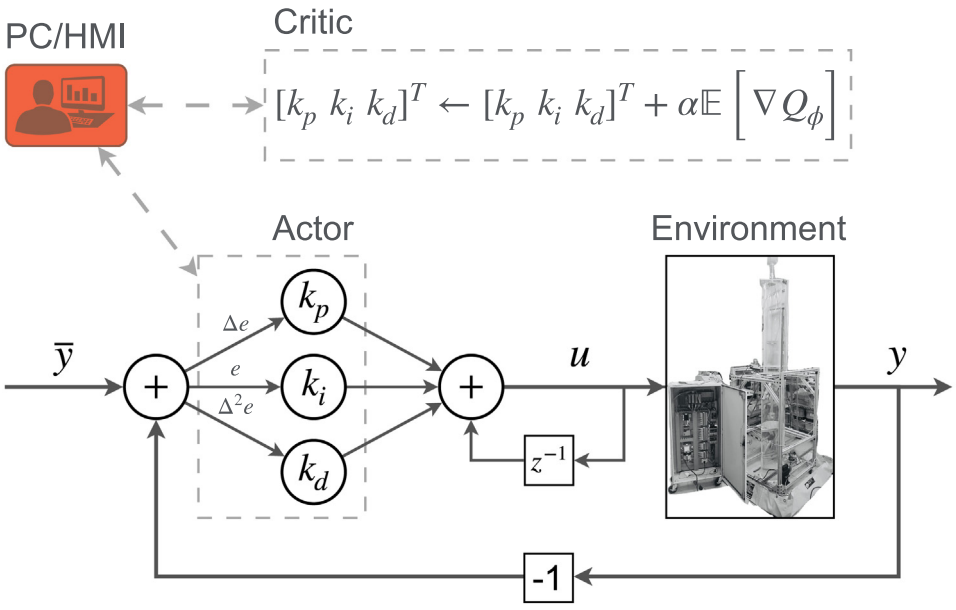

Fig. 1. A conceptual diagram of the proposed method. The actor network is a PID controller, which interacts with a physical two-tank system. The PC/HMI collect process data, which is used by the RL algorithm to train a critic network. The critic network then updates the PID parameters, which are read by the HMI and sent back to the system. Dashed lines indicate a separation of time scales between the RL algorithm and the physical system.

In the space of all possible policies, the optimization is performed over a subset parameterized by some vector θ. For example, in some applications, θ denotes the set of all weights in a deep neural network. In this work, we take θ to be the gains in a PID controller. The policies considered are denoted μ_θ, and we simplify Problem (1) by writing $J(\theta)$ instead of $J(\mu_\theta)$ and p^θ instead of p^{μ_θ}.

A standard approach to solving Problem (1) uses gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta), \tag{2}$$

where $\alpha > 0$ is a step-size parameter. Analytic expressions for $\nabla J(\theta)$ exist for both stochastic and deterministic policies (Silver et al., 2014; Sutton & Barto, 2018). Crucially, these formulas rely on the state–action value function:

$$Q(s_t, a_t) = \mathbb{E}_{h\sim p^\theta(\cdot)}\left[\sum_{k=t}^{\infty} \gamma^{k-1} r(s_k, \mu_\theta(s_k)) \middle| s_t, a_t\right]. \tag{3}$$

Although Q is not known precisely, as it depends on both the dynamics and the policy, it can be estimated with a deep neural network (Mnih et al., 2015). Writing ϕ for the vector of parameters in this network, and Q_ϕ for the approximation to Q that it defines, ϕ is chosen to minimize the temporal difference error across N observations indexed by i (or variations of this, as given in the references cited below):

$$\mathcal{L}_{\text{critic}}(\phi) = \frac{1}{N}\sum_{i=1}^{N}\left(Q^{(i)}_{\text{target}} - Q_\phi(s^{(i)}, a^{(i)})\right)^2. \tag{4}$$

For example, $Q^{(i)}_{\text{target}} = r(s^{(i)}, a^{(i)}) + \gamma Q_\phi(s'^{(i)}, \mu_\theta(s'^{(i)}))$, and s' represents the next state in the trajectory following policy μ_θ. With an up-to-date critic network, we then define the loss for the actor network as follows:

$$\mathcal{L}_{\text{actor}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} Q_\phi(s^{(i)}, \mu_\theta(s^{(i)})). \tag{5}$$

$\nabla\mathcal{L}_{\text{actor}}$ serves as a tractable approximation of ∇J, and is therefore used in the nominal policy update shown in Eq. (2) (Silver et al., 2014). These ideas are the basis of popular deep RL algorithms such as DDPG, TD3, SAC (Fujimoto et al., 2018; Haarnoja, Zhou, Abbeel, & Levine, 2018; Lillicrap et al., 2015). More generally, they fall into the class of *actor–critic* methods (Konda & Tsitsiklis, 2000), as they learn both a policy μ ($\approx\mu_\theta$) and a state–action value function Q ($\approx Q_\phi$). We use a modified version of TD3, the twin-delayed DDPG algorithm (Fujimoto et al., 2018), a refinement of DDPG (Lillicrap et al., 2015). For completeness, we present the basic algorithm in Algorithm 1 and give an overview of its main differences from DDPG in Appendix A.

2.2. PID in the RL framework

We now apply the general formulation given in the previous section to the problem of PID tuning. The proposed framework is illustrated in Fig. 1. We use the variable $\bar{y}$ to denote a reference signal; we generally omit the time index with the understanding that the initial time $t = 0$ indicates a step change in the reference signal, which is then constant. We then write the output error signal as $e^{(y)}_t = \bar{y} - y_t$. We use the operator Δ to denote a first-order difference between time steps of a signal; for example, $\Delta e^{(y)}_t = e^{(y)}_t - e^{(y)}_{t-1}$. We use Δt to denote the sampling time. When implementing a PID controller in the incremental form, the derivative term requires second-order output information. We use a superscript f to denote a signal resulting from a low-pass filter; since the only instances in which we use this convention are to approximate derivatives, we include the sampling time in the definition. The filtered first-order difference of the output $\Delta y^{(f)}_t$ is used to compute a second-order difference, based on the following recursion:

$$\Delta y^{(f)}_t = T_f \Delta y^{(f)}_{t-1} + (1 - T_f)\frac{y_t - y_{t-1}}{\Delta t} \tag{6}$$

$$\Delta^2 y^{(f)}_t = \frac{\Delta y^{(f)}_t - \Delta y^{(f)}_{t-1}}{\Delta t}, \tag{7}$$

where $T_f \in [0, 1]$ is a fixed parameter. An anti-windup component[2] is incorporated through the variable $e^{(u)}_{t-1} = \hat{u}_{t-1} - u_{t-1}$, where $\hat{u}$ is the signal proposed by a PID controller before being saturated and $u = \text{sat}(\hat{u})$ denotes the actual input signal after saturation.

[2] Although we are working with the incremental form of PID controller, which automatically resets the controller at its saturation limits, we still use the terminology "anti-windup" to refer to this component of the controller.

With the variables $\Delta e_t^{(y)}$, $e_t^{(y)}$, $\Delta^2 y_t^{(f)}$, $e_{t-1}^{(u)}$, u_{t-1}, a PID controller can then be written as follows:

$$\hat{u}_t = \begin{bmatrix} k_p & k_i & k_d & k_\tau \end{bmatrix} \begin{bmatrix} \Delta e_t^{(y)} \\ \Delta t e_t^{(y)} \\ -\Delta^2 y_t^{(f)} \\ \Delta t e_{t-1}^{(u)} \end{bmatrix} + u_{t-1}. \tag{8}$$

The scalar $\hat{u}_t$ given by Eq. (8) is clipped to produce the physically admissible input signal $u_t = \mathrm{sat}(\hat{u}_t)$, which is sent to the plant. Since Eq. (8) contains first and second-order output information, and the system usually contains time delay, the RL state is taken to be a history of the observations $o_t = \left[\Delta e_t^{(y)}, \Delta t e_t^{(y)}, -\Delta^2 y_t^{(f)}, \Delta t e_{t-1}^{(u)}, u_{t-1}\right]$:

$$s_t = \left[o_{t-d}, \ldots, o_t\right] \tag{9}$$

where d is a nonnegative integer. In the context of the RL formulation in Section 2.1, the "actions" a_t are interchangeable with the control inputs u_t.

Eq. (9) contains the necessary information for implementing a PID controller in discrete time steps. Eq. (8) parameterizes the PID controller. We therefore take Eq. (8) to be a shallow neural network, where $[k_p\ k_i\ k_d\ k_\tau]$ is a vector of trainable weights. In light of Eqs. (2), (5) and (8), the PID update equation takes the following explicit form:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta) \tag{10}$$

$$\approx \theta + \alpha \nabla \left(\tfrac{1}{N} \textstyle\sum_{i=1}^{N} Q_\phi(s^{(i)}, \mu_\theta(s^{(i)})) \right) \tag{11}$$

$$= [k_p\ k_i\ k_d\ k_\tau]^T + \cdots$$
$$\alpha \tfrac{1}{N} \textstyle\sum_{i=1}^{N} \left(\left[\Delta e^{(y)}, \Delta t e^{(y)}, -\Delta^2 y^{(f)}, \Delta t e^{(u)}\right]^T \right)^{(i)} \nabla_a Q_\phi(s^{(i)}, a)\Big|_{a=\mu_\theta(s^{(i)})}. \tag{12}$$

Eq. (12) follows by the chain rule and can be computed automatically in deep learning frameworks, such as PyTorch, based on the computation of $\mathcal{L}_{\text{actor}}$ and $\mathcal{L}_{\text{critic}}$. Later in Section 5.3 we explain strategies for training the weights subject to the saturation nonlinearity.

3. Scorecard for RL algorithms in process control

We propose the following criteria for evaluating RL based tuning methods.

1. *Nominal performance*: How much does the RL agent improve the performance of the closed-loop system compared to its initialization?
2. *Stability*: Is the trained closed-loop system stable? Did it become unstable during training?
3. *Perturbation to the system*: How much perturbation to the process is required to achieve satisfactory performance?
4. *Initialization*: To what extent does the performance of the RL agent depend on the initial PID parameters?
5. *Hyperparameters*: Do the algorithm hyperparameters need to be adjusted between experiments? Which hyperparameters influence the learning process most strongly?
6. *Training duration*: How many episodes are required to achieve satisfactory performance?
7. *Practicality and specialization*: What hardware is required in order to deploy the RL algorithm on the physical process? What level of user expertise is required in order to implement the algorithm?

Many of these criteria have natural counterparts in current tuning methods, which provide useful points of reference in our evaluation. These are discussed further in Section 6.4. To quantify our assessment, we consider the integral absolute error (IAE), integral squared error (ISE), total variation (TV), overshoot (OS), and settling time (ST):

Fig. 2. Our two-tank system in a lab at Honeywell for testing PID tuning algorithms. Each tank is roughly 12 cm (radius) by 122 cm (height).

$$\text{IAE: } \int_0^\infty |e(t)|dt \qquad \text{ISE: } \int_0^\infty e(t)^2 dt$$
$$\text{TV: } \sum_{t=0}^{\infty} |y(t+1) - y(t)| \qquad \text{OS: } \max_{0\le t<\infty} \left\{ |e(t)| \mathbb{1}_{\{t\,:\,e(0)\cdot e(t)<0\}}(t) \right\} \tag{13}$$
$$\text{ST: } \min_{0\le t<\infty} t \mathbb{1}_{\{t\,:\,|e(\tau)|\le \epsilon\ \forall \tau \ge t\}}(t) \qquad \text{where } \mathbb{1}_\Omega(t) = \begin{cases} 1 & \text{if } t \in \Omega \\ 0 & \text{otherwise.} \end{cases}$$

Of course, these are approximated as summations using discrete-time samples from the system. To compute percent overshoot (% OS), the OS is multiplied by $100/|\Delta y_{\text{sp}}(0)|$. We will also be interested in the total variation of the input signal, which is denoted by TV_u.

These criteria address much more than simply the performance of the trained RL agent. We are interested in the "path" the RL agent takes to reach its final form, the perturbations it makes to the system, its usability, and the overall software/hardware requirements.

4. Lab setup and software structure

In this section, we describe the dynamics and instrumentation of our two-tank system. We also give an overview of the software used to implement a RL agent on the physical system with standard hardware.

4.1. Description of the two-tank system

We consider the problem of controlling the liquid level in a tank, using the physical apparatus shown in Fig. 2. The tank of interest is positioned vertically above a second tank that serves as a reservoir. Water drains from the tank into the reservoir through an outflow pipe, and is replenished by water from the reservoir delivered by a pump whose flow rate is our manipulated variable. More precisely, two PID controllers are in action: For a desired level, one PID controller outputs the desired flow rate based on level tracking error. This flow rate is then used as a reference signal for the second PID controller, whose output

is the pump speed. The first is referred to as the "level controller" and the second as the "flow controller".

Significant physical dimensions (with units of length) include r_{tank}, the radius of the top tank; r_{pipe}, the radius of the pipe for the outflow of the top tank; and ℓ, the distance from the base of the top tank up to the water surface. (We later refer to m, a filtered counterpart of ℓ.)

Key flow parameters, with units of volume/time, are the outflow f_{out}; the inflow f_{in}; an empirical coefficient f_c; and $f_{\max}$, the maximum flow the pump can deliver.

The pump delivers water to the top tank at the rate $(p/100)f_{\max}$, where p is a dimensionless percentage in $[0, 100]$.

We write $\bar{p}$ for the commanded pump speed, typically a piecewise-constant setpoint function.

The system dynamics are based on Bernoulli's equation, $f_{\text{out}} \approx f_c\sqrt{2g\ell}$, and the conservation of fluid volume in the upper tank:

$$\frac{d}{dt}\left(\pi r_{\text{tank}}^2 \ell\right) = \pi r_{\text{tank}}^2 \dot{\ell} = f_{\text{in}} - f_{\text{out}}. \tag{14}$$

(We use a dot for the time derivative; g is the gravitational constant.)

Our system model combines the principles above with some simple filters to make our mathematical description physically realizable. A first-order filter with time constant $\tau \geq 0$ transforms an input signal $\hat{y}$ into the output signal y defined by

$$\tau\dot{y} + y = \hat{y}, \qquad y(0) = 0. \tag{15}$$

Note that setting $\tau = 0$ gives $y(t) = \hat{y}(t)$, while if $\hat{y}$ is constant, one has $y(t) = (1 - e^{-t/\tau})\hat{y}$. Our application involves four filtered signals, with time constants τ_p for the pump, τ_{in} for changes in the inflow, τ_{out} for the outflow, and τ_m for the measured level dynamics.

We therefore have the following system of differential equations describing the pump, flows, level, and measured level:

$$\tau_p\dot{p} + p = \bar{p} \tag{16}$$

$$\tau_{\text{in}}\dot{f}_{\text{in}} + f_{\text{in}} = f_{\max}\left(\frac{p}{100}\right) \tag{17}$$

$$\tau_{\text{out}}\dot{f}_{\text{out}} + f_{\text{out}} = \pi r_{\text{pipe}}^2 f_c\sqrt{2\,g\ell} \tag{18}$$

$$\pi r_{\text{tank}}^2\dot{\ell} = f_{\text{in}} - f_{\text{out}} \tag{19}$$

$$\tau_m\dot{m} + m = \ell. \tag{20}$$

To track a desired level $\bar{\ell}$, we can employ level and flow controllers by including the following equations:

$$\bar{p} = \text{PID}_{\text{flow}}(\bar{f}_{\text{in}} - f_{\text{in}}) \tag{21}$$

$$\bar{f}_{\text{in}} = \text{PID}_{\text{level}}(\bar{\ell} - m). \tag{22}$$

Eqs. (21)–(22) use shorthand for PID controllers taking the error signals $\bar{f}_{\text{in}} - f_{\text{in}}$ and $\bar{\ell} - m$, respectively. For our purposes, PID_{flow} is fixed and a part of the environment, while $\text{PID}_{\text{level}}$ is the tunable controller.

This mathematical description is given to provide intuition for our control system. Moreover, it was used for the offline development of our interface for interacting with the physical system.

4.2. Human–machine interface

We developed a human–machine interface (HMI) in Matlab App Designer to interact with the tank system. The interface is shown in Fig. 3a. From the HMI, we can adjust the PID parameters for controlling the pump speed, flow rate, and level of the top tank. More precisely, the PID controllers for the pump and flow produce setpoints which the physical pump must attain. This is illustrated in Fig. 3b. As a Matlab application, it is easy to build in a lot of additional functionality into the HMI. Although we refer to it as the HMI, it also includes implementations of PID controllers, automated and on-demand saving of data to CSV files, and logic and algorithms to supervise and run process experiments. The HMI also includes an embedded simulation of the system based on the dynamics given in Section 4.1. This setup allows easy switching between simulation and reality, providing a unified interface for conceptual development and laboratory experiments. We note that the simulator was *not* used to pre-train the RL agent.

4.3. Software and hardware for training RL agents

(Software) Our starting point for developing a RL agent was the open source repository Spinning Up (Achiam, 2018). In principle, deep RL libraries like Spinning Up can help streamline the process of testing new algorithms on novel simulated environments through the Open AI Gym (Brockman et al., 2016). However, our project called for a number of fundamental changes. Most importantly, the RL agent does *not* directly actuate the system (as in Gym environments). Rather, it is one element of a modular design comprising the RL code, the HMI, and the instrumentation of the two-tank system.

In light of this, it is worth noting that there are two PID representations of Eq. (22) at play: The actor network in the RL code, and the PID controller acting on the system through a hardware programmable logic controller (PLC). The actor network representation is implemented in PyTorch and used for the purposes of training. This setup requires inter-module communication to transmit the results of training to the PLC for implementation.

In order to accommodate this distinction, we made the following changes to the Spinning Up implementation of the TD3 algorithm:

- The standard actor network in Spinning Up is a feedforward neural network. A PID is a linear function followed by the identity activation function, given by Eq. (8). We provided a custom initialization of the PID parameters (rather than the typical random initialization) and constrained the parameters to be positive through the use of a smooth and invertible approximation of ReLU ($x \mapsto \max\{0, x\}$) called Softplus ($x \mapsto \ln(1 + \exp(x))$). For example, if one wishes to initialize the proportional gain as $k_p = 4.0$, then the corresponding weight in the RL code would be initialized as $\theta_{k_p} = \ln(\exp(4.0) - 1)$. A similar strategy constrains the anti-windup term to the interval $(0, 1)$, by using Sigmoid ($x \mapsto 1/(1 + \exp(-x))$) instead of Softplus. Therefore, instead of taking $\theta = [k_p\ k_i\ k_d\ k_\tau]$ as in Section 2.2, the vector of inverted PID parameters is used as the trainable weights $\theta = [\theta_{k_p}\ \theta_{k_i}\ \theta_{k_d}\ \theta_{k_\tau}]$.
- We removed all linkages to a Gym-style environment. Instead, new data from the tank is processed in batches. Our HMI records process data and periodically saves it to a directory accessible to the RL agent. The RL code processes the new data then updates its internal representation of the PID controller according to Algorithm 1. More specifically, the RL code is responsible for reading these data files and constructing its replay buffer consisting of state transition tuples (s, a, s', r). Once the weights θ are updated, they are translated back to "PID form" through the Softplus or Sigmoid functions as described above, then saved and read by the HMI.
- As a consequence of the above changes, we also removed all parameters in the code that would characterize an episode in the Gym environment. These include the number of time steps per episode, the number of samples to collect before training begins, and how many time steps pass between parameter (both actor and critic) updates. The effect of these parameters is still present, as the HMI includes these specifications; the main difference is that the RL code is only used to process new data. We can easily change design parameters in the HMI online, while the RL code is running in the background.

(Instrumentation and System Integration) The key field measurement devices and actuators included in the lab apparatus are a guided wave radar level measurement device to measure the level in the upper tank; a differential pressure flow measurement device to measure the flow of water into the upper tank; and a variable speed pump to pump water from the lower tank, through the flow measurement device, and into the upper tank. These are wired into an HC900 process and logic controller which allows communication with Matlab, via Modbus/TCP, to support a HMI and controls. Two UDC2500 loop controllers are also connected to the HC900 so that these elements may be used optionally for flow and/or level control. Logic programmed in the HC900 is used to switch between flow and level control by Matlab

Fig. 3. (a) The HMI control screen for the two-tank system. (b) Real-time plots show tracking performance and PID parameters; dashed lines are setpoints and solid lines are measured process variables.

PID implementations, by the UDC2500 controllers, or by manual pump speed adjustments.

(Computing) For our purposes, it was sufficient to run the HMI and train the RL agent on a desktop computer in the lab. The PC used in the lab has an Intel Xeon CPU which runs at 3.5 GHz, and 8 GB of RAM. (This is a good but unremarkable PC.) Due to our modular setup, it would also be possible to train the RL agent through cloud services.

5. Experiment design and case studies

5.1. Experiments for learning

An episode is characterized by a step change in the process; a timer specifies how much time will elapse between step changes. To ensure safe operation even when unsupervised, the HMI has access to a set of PID parameters that are known to stabilize the system. These may be poorly tuned parameters, such as ones used before the tuning procedure. The HMI is able to switch to these parameters if the tracking provided by the RL-tuned parameters is too poor, guaranteeing that the next step change starts from steady state. Therefore, the timer parameter may be set online based on closed-loop experiments, serving as an upper bound for when to trigger the "safe" PID parameters unless the system is brought to steady-state beforehand. For simplicity and consistency in our experiments, we keep the timer fixed during experiments.

Measured data are stored, processed, then used to train the RL agent during operations. The RL agent generates new PID parameters, which are then loaded into the HMI to update the Matlab PID control implementations. The outputs from these PID controllers are sent to the HC900 process and logic controller which passes them to the

appropriate field devices. This process of sending new data to the RL agent and updating the PID parameters can be done during an experiment or set to occur at the end.

We train using a pre-defined set of setpoints; an "episode cycle" refers to one pass over all of these setpoints. In the context of training, an "experiment" refers to all the episodes leading to the final tuning parameters. Later on we refer to an "evaluation experiment" or "robustness experiment" in the context of some final set of tuning parameters that we wish to evaluate in various conditions.

5.2. Reward function selection

A crucial component of any RL agent is the reward function. This function must accurately capture the goals of the system designer. Therefore, we consider costs (negative rewards) that depend only on the tracking error e_t and the change in control variable $\Delta u_t = u_t - u_{t-1}$, in the form

$$l(s_t, u_t) = |e_t|^p + \lambda |\Delta u_t|^q, \tag{23}$$

where p, q are fixed integers (namely, 1 or 2) and $\lambda \geq 0$ is a fixed penalty term. For a system with multiple inputs and outputs, one can generalize Eq. (23) by using the ℓ_1 or squared ℓ_2 norms. Although the cost given by Eq. (23) is commonly used in the case of $p = q = 2$, such as in applications of model predictive control, the user is left with the tuning parameter λ. A reasonable initial choice is $\lambda = 1/|\Delta u|_{\max}$, where $|\Delta u|_{\max}$ is an approximation of the maximum absolute value of Δu observed within the operating region. This is simply a normalization step: one can rewrite the penalty term as $\lambda \frac{|\Delta u_t|^q}{|\Delta u|_{\max}}$ and adjust λ based on relative weight given to the tracking error term of Eq. (23). For our purposes, we use the penalty term $0.1(\Delta u_t)^2$.

The above reward function structure is flexible in terms of the range of behaviors it can incentivize. For example, setting $\lambda = 0$ selects the absolute or squared error reward function. The squared error places a significant amount of weight on states with large errors; in comparison, the absolute error puts more emphasis on small errors. Therefore, it is reasonable to prefer the absolute error for a slower response but with better attenuation of overshoot and oscillations. Conversely, a fast response may be desirable. One of the appeals of RL is the flexibility in the choice of the reward function. Therefore, we also test a cost function that has the desirable components of both the absolute and squared errors:

$$l(s_t, u_t) = \begin{cases} |e_t| & \text{if } |e_t| < 1 \\ \frac{1}{2}(e_t^2 + 1) & \text{otherwise.} \end{cases} \tag{24}$$

Eq. (24) is the absolute value function around the origin and smoothly transitions to a parabola. From now on, we use l exclusively for Eq. (24), and refer to it as the "hybrid cost" (or reward). One may set $r = -l$ or use a reward based on Eq. (24) with an input penalty.

5.3. Tuning subject to input constraints

One difficulty in applying RL algorithms to tune a PID controller is the inherent presence of input constraints in a physical system. We follow the methods proposed by Hausknecht and Stone (2016): The actor update scheme in Eq. (11) is modified to steer its actions to within a pre-defined range $[u_{\min}, u_{\max}]$. Formally, since we have

$$\nabla_\theta Q_{\phi_i}(s,u)|_{u=\mu_\theta(s)} = \nabla_\theta \mu_\theta(s) \nabla_u Q_{\phi_i}(s,u)|_{u=\mu_\theta(s)} \tag{25}$$

by the chain rule, the components of $\nabla_u Q_{\phi_i}(s,u)|_{u=\mu_\theta(s)}$ are scaled as follows:

$$\frac{\partial Q_{\phi_i}}{\partial u}(s,u) \leftarrow \frac{\partial Q_{\phi_i}}{\partial u}(s,u) \cdot \begin{cases} \frac{u_{\max} - u}{u_{\max} - u_{\min}} & \text{if } \frac{\partial Q_{\phi_i}}{\partial u}(s,u) > 0 \\ \frac{u - u_{\min}}{u_{\max} - u_{\min}} & \text{otherwise.} \end{cases} \tag{26}$$

Without making the RL agent "aware" of the input constraints it may continue to propose infeasible actions. Intuitively, Eq. (26) reverses the direction of the parameter update if the critic "reinforces" an infeasible action toward the optimum. This update scheme also puts less weight on a parameter update when an action is close to the constraints.

Another option for updating the actor parameters subject to constraints is to use an output activation on the actor such that the actions are automatically forced inside the range $[u_{\min}, u_{\max}]$ (Hausknecht & Stone, 2016). Options include using the saturation function or a smooth approximation such as tanh. However, this approach diminishes the gradient in Eq. (25) at the constraints, essentially ignoring the value of such state–action pairs. We emphasize that the PID controller on a physical system will still obey the constraints $[u_{\min}, u_{\max}]$, whether by physical limitations or by modeling them directly with the saturation function; the update scheme in Eq. (26) is solely for the purpose of updating the actor and any constraint violations performed in the update scheme are not reflected on the physical system.

5.4. Standard tuning methods

To provide an overall evaluation of our RL algorithm, we compare our results to baseline tuning methods in the context of the criteria put forth in Section 3. In particular, we compare the RL results to various tuning parameters given by the Skogestad IMC (SIMC) tuning method (Skogestad, 2003) and Honeywell's Accutune III algorithm (Berner et al., 2018; Honeywell, 2007). SIMC provides PI parameters based on a first-order plus dead time (FOPDT) model of the plant; the closed loop time constant T_c is the only tuning parameter for this method. Accutune III is a relay autotuning algorithm; its user inputs are the input range for the relay signal and a switch to select either "fast" or "slow" tuning. Note that these methods do not share the same underlying objective of RL, so the purpose of including them is simply for baseline data of what reasonable performance or robustness look like in our setting. Moreover, we can evaluate the three methods (RL, SIMC, Accutune) through the lens of our "scorecard" items in Section 3.

6. Lab results

We report our experimental results in two sections. This section reports some key results and high-level conclusions. Appendix B provides all the data supporting our findings. Table A.3 lists the hyperparameters used in this work. We emphasize that all the experiments presented here are performed directly on the physical two-tank system without prior pre-training, for example, in simulation or with offline datasets.

6.1. Experimental setup

We run three different RL experiments, all using this reward function:

$$-r(s_t, u_t) = |e_t| + 0.1\Delta u_t^2. \tag{27}$$

They differ based on the initial PID tuning parameters for training as well as the operating conditions. The first two experiments are "unconstrained experiments"; more precisely, we run Algorithm 1 in an operating region where the agent is unlikely to hit the physical constraints of the pump. Our algorithm is designed to improve the initial tuning parameters; therefore, we test the algorithm from different starting parameters. The initial tuning parameters are given according to the following dependence on k_p:

$$k_i = k_p/60, \;\; k_d = 0.01 k_p, \tag{28}$$

where the first experiment sets $k_p = 4.0$ and for the second $k_p = 2.0$. Since the RL agent operates in closed loop, the input constraints may not always be avoidable, which is why they must be accounted for during training. We next run a "constrained experiment" with $k_p = 4.0$. That is, we run Algorithm 1 with a strategy for dealing with input constraints, as described in Section 5.3. This experiment switches between the setpoints 60 cm and 65 cm as well 60 cm and 63 cm; this configuration gives the agent training data for setpoint tracking both with and without input constraints.

Table 1
The nominal performance of RL experiments and baseline tuning methods. Each cell shows the average normalized performance over the sequence of step changes.

	IAE	ISE	TV	TV_u	% OS	ST	M_s
RL	39.34 ± 3.44	27.56 ± 2.25	1.49 ± 0.08	9.37 ± 0.42	6.39 ± 1.21	127.00 ± 38.80	1.32 ± 0.06
SIMC	47.74	32.31	1.50	9.14	7.81	215.25	1.25
Accutune	54.06	34.06	1.75	583.78	9.14	159.75	1.41

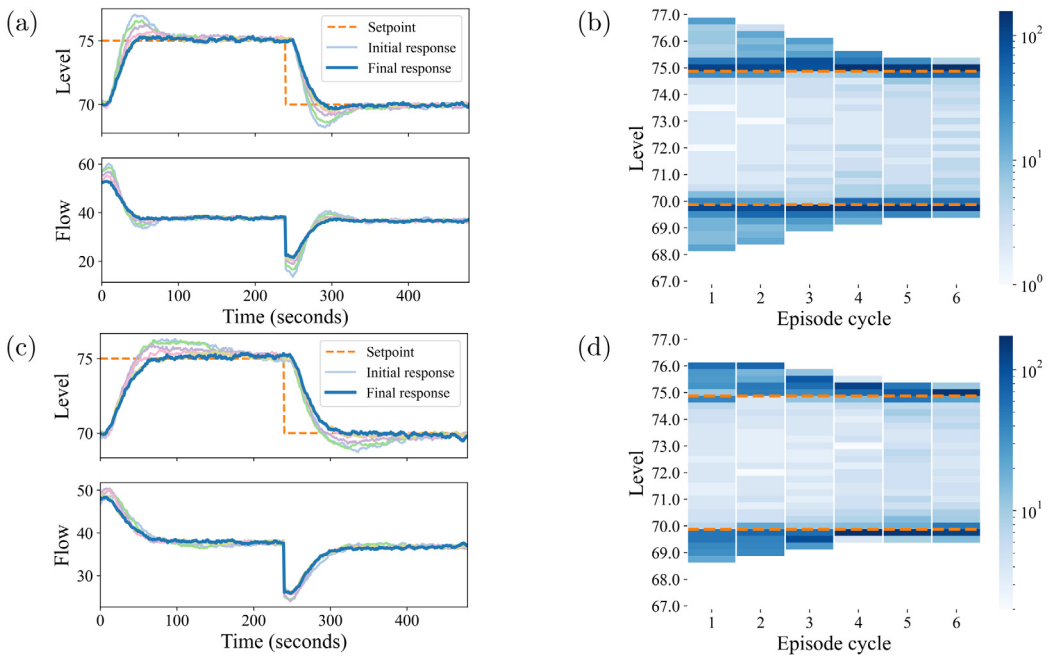

Fig. 4. Performance evolution under two different reward functions: (a) Tracking in experiment for RL-2; (b) Tracking heatmap for RL-2: x-axis is progression of episode cycles (2 step changes each) for training and the y-axis is the level; (c) Tracking in experiment for RL-3; (d) Tracking heatmap for RL-3.

6.2. Evaluation procedure

For each set of tuning parameters (five in total), we cycle through each setpoint for four minutes each. The setpoint sequence is 60 cm, 65 cm, 60 cm, 63 cm. Input constraints are inactive for the evaluation. The reason for this, even for the results from the constrained experiment, is to evaluate the quality of the PID parameters across all methods.

For each of these evaluation experiments, we calculate the average normalized integral absolute error (IAE), integral squared error (ISE), total variation (TV), total variation in the input variable (TV_u), percent overshoot (% OS), and settling time (ST). *Normalized* performance means the IAE, ISE, TV use the error signal divided by the change in the setpoint for that step change in the calculation; TV_u is calculated by dividing by the initial Δu value instead.

To perform the SIMC tuning method we first derive a FOPDT model of the flow setpoint to level dynamics of our system. Based on data from stepping the flow setpoint up and down to steady state, we obtain the model $G(s) = \frac{3.44}{301.19s+1}e^{-9.21s}$. We consider the values $T_c = 9.21, 15, 20, 25, 30$. The first value of T_c comes from the "default" configuration of setting T_c equal to the process time delay; however, since G is nearly an integrating processes, this setting may not be desirable. In this section, we only use $T_c = 20$; for completeness, the other values are evaluated in Appendix B. Finally, we use a UDC2500 to run the Accutune III procedure and to control the flow for the ensuing evaluation steps.

6.3. Summary of results

Table 1 shows the nominal performance of the final tuning parameters across the RL, SIMC, and Accutune evaluation experiments. We also include the maximum sensitivity $M_s = \max_{0\le\omega<\infty} (1 + C(i\omega)G(i\omega))^{-1}$ based on the model G used for SIMC tuning. The RL row summarizes the performance the three aforementioned experiments. We report the average of each of these statistics across the three experiments, plus or minus the standard deviation. All three experiments perform better than Accutune III across all metrics, with the one exception of ST for the constrained RL experiment. RL and SIMC achieved similar TV, TV_u and OS, with RL performing better in terms of IAE, ISE, and ST. Ultimately, the result of the RL tuning across these different operating conditions is smooth and efficient tracking. Moreover, as we will see next, each RL result was obtained in around 40 min of operation.

Fig. 4 shows the evolution of the training process for the two unconstrained experiments. We show two visualizations for each experiment ($k_p = 4.0$ and $k_p = 2.0$, respectively). Time-series plots are given in Figs. 4(a) and 4(c). The initial step performances are characterized by a fast rise with significant overshoot (roughly 40%) or a slow settling time (roughly 3.5 min). We see that in all the experiments the performance uniformly plateaus at around 10 episodes. Each episode is roughly four minutes. Even though the experiments run for varying lengths of time, all of them reach their peak performance in less than 40 min of operation.

Figs. 4b and 4d are respective heatmaps of the same output data. The time-series and heatmaps are shown side-by-side to convey the intuition for the heatmap, which is more heavily used in Appendix B because it is a compact way of showing many different experiments together. The darker shades mean the process variable spent more time in that region of the y-axis than lighter regions. For example, the heatmap captures overshoot with the presence of shaded regions above the setpoint, and conveys settling time based on the distribution of shading around the setpoint. We see as training progresses (that is, as the number of episodes increases) the darker shading is more concentrated around the setpoints and the overshoot decreases. We also

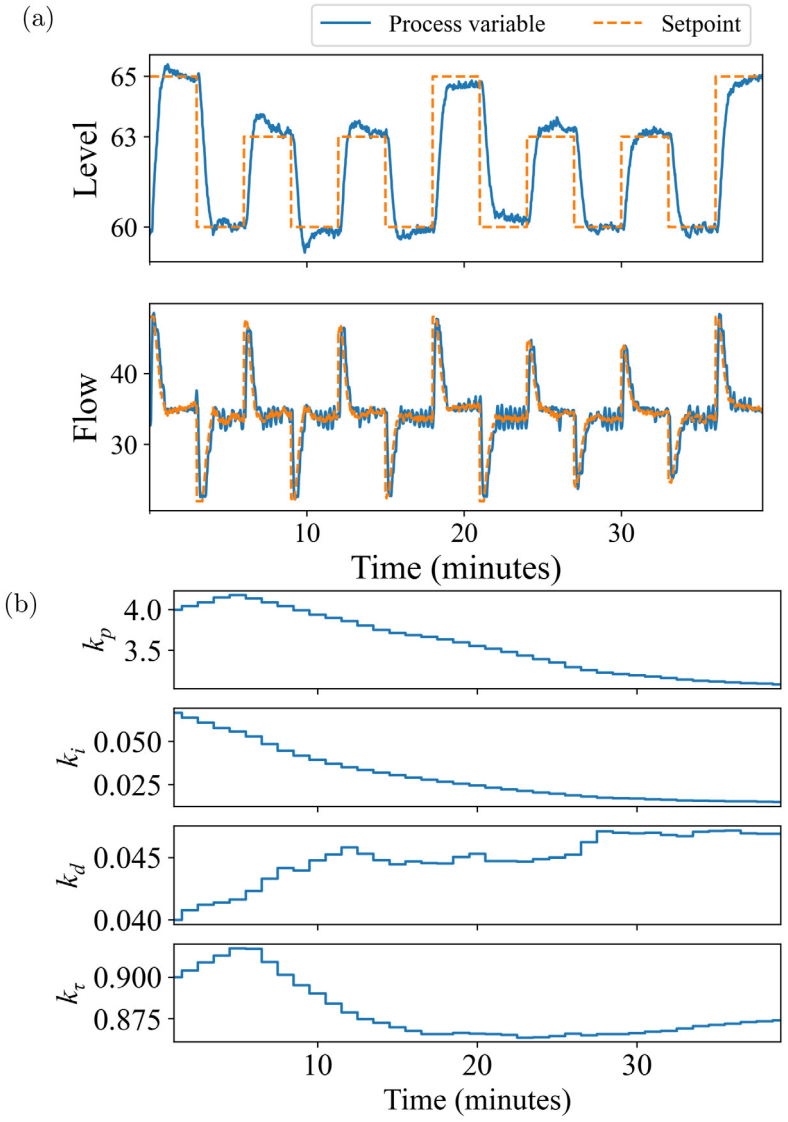

Fig. 5. The progression from initial to final performance of RL-8. (a) A time-series of the entire training procedure; (b) A temporally-aligned plot of the tuning parameters during training.

Table 2
A qualitative evaluation of deep RL, SIMC, and Accutune III. We use ✓ to indicate "excellent"; × to indicate "poor"; and – to indicate "neutral".

	Nominal performance	Robustness	Time required for tuning	Disturbance to process	Hardware requirements	Ease of use	Input constraints
SIMC	–	–	–	×	✓	–	×
Accutune III	×	×	✓	✓	✓	✓	×
Deep RL	✓	✓	×	✓	–	✓	✓

see a slightly longer rise time based on how dark the region is *between* setpoints.

The two RL experiments corresponding to $k_p = 4.0$, one unconstrained and one constrained, achieved similar evaluation performance with the exception of ST, where the unconstrained experiment achieved roughly 40% better settling time. However, the latter was trained subject to more realistic conditions. Further, the qualitative end result of the constrained experiment, as shown in Fig. 5, is still excellent. Note that this time-series is with the input constraints in place and is also temporally-aligned with the tuning parameters during training.

Despite the differences among RL experiments, all but two of them, which we show in Appendix B, performed better than SIMC in terms of IAE and ISE. Two instances of SIMC showed an advantage over RL in terms of TV, TV_u, and %OS, but at a significant cost in IAE and ISE. Accutune-1 performs reasonably well in terms of TV, %OS, and ST, but achieves, by far, the highest values of IAE, ISE, and TV_u across the RL and SIMC experiments. Finally, the absolute value based reward is good for "smooth" tracking and overall performs well across the metrics in Table 1. In Appendix B, rewards based on the squared error or hybrid function may be better for "fast" tracking but suffer in terms of TV, TV_u, and %OS as well as in experimental robustness. We consider the experiments based on the reward in Eq. (27) to be the overall best option.

6.4. Overall evaluation

As mentioned in Section 3, performance and robustness are not the only factors to consider when evaluating a tuning method. Table 2 provides a summary of our qualitative assessment of deep RL with SIMC and Accutune III included as points of reference. Unsurprisingly, deep RL achieved excellent nominal performance across different initializations, reward functions, and subject to input constraints. Moreover, the tuning of these results is robust to changes in the two-tank system (see Appendix B) . Although these result were achieved in a reasonable amount of time ($30-50$ min), commercial auto-tuners such as Accutune III provide tuning parameters in roughly 10 min (excluding additional time to evaluate the performance). SIMC recommends setting $T_c = \theta$, but manipulating T_c from there requires additional experiments. The next thing to note is that SIMC and Accutune III are open loop tuning methods, whereas deep RL operates in closed loop; therefore, despite longer training time, we consider the disturbance to the process to be more practical than SIMC or Accutune III, *if* one can provide an acceptable range of setpoints. Accutune III is also desirable in this regard because it switches the relay signal based on deviations of the process variable from the current setpoint. An advantage of SIMC and Accutune III is that they are simple enough such that no specialized hardware is required. For a single control loop, we were able to use a standard desktop for training the RL agent. Finally, taken in their final form, both deep RL and Accutune III can be simple to use: in the case of RL, a user may input some rudimentary information, such as acceptable setpoints, while Accutune III relies on an interval of admissible inputs.

7. Discussion and conclusion

To conclude this study, we contrast our findings with some common themes that circulate in the deep RL literature and highlight promising areas for future work.

The primary concerns at the prospect of applying deep RL in the process industries pertain to stability, interpretability, sample efficiency, and practicality (Nian et al., 2020; Shin et al., 2019). In some ways these are complementary concepts: In the broad landscape of RL, the policy is often represented by a DNN, which makes it difficult to rigorously explain its behavior. This contributes to the difficulty surrounding sample efficiency, interpretability, and stability due to the nonlinear structure of a DNN operating in a closed-loop system. From a practical implementation perspective, it is of course possible to deploy these policies in an industrial control system. However, there already exists extensive investment and research into deploying control architectures such as MPC and PID. Given their prevalence and track record, the most promising starting point for RL applications in the process industries is through some hybrid approach.

In this work, we have directly parameterized the policy as a PID controller and configured it by solving the RL objective using a model-free deep RL algorithm. Other approaches, mentioned in Section 1.1, train the RL policy to output PID parameters as "actions", but the policy itself is a DNN or derived from value-based approaches. Consequently, we have orders of magnitude fewer parameters to tune. Crucially, we also exploit the fact that a PID controller is standard in universal digital controllers, and therefore only need to send new parameters to the controller, rather than install new hardware in order to run the RL policy.

In our experimental results we are encouraged by the monotonic improvement in IAE and ISE, ultimately leading to a well-tuned PID controller within $30-50$ min. We attribute this to the small number of parameters in a PID controller and the fact that setpoint tracking is a "primitive" of its design; in other words, it does not need to "learn" how to track setpoints, rather improve upon its existing performance. Note that these results are obtained by training the critic network from scratch, that is, without any pre-training either from a simulation or historical process data. The fact that we could deploy the algorithm without adding any specialized hardware to the system, while operating in closed-loop, and performing the computation on a local desktop computer is encouraging. Ultimately, these are promising results on which to build.

7.1. Opportunities in deep RL

Despite some promising results presented here, a looming question persists: How can one systematically configure the RL agent to a novel environment when it has failed to learn? The central problem at play is that of training RL agents on a case-by-case basis. Despite being dubbed "model-free", actor–critic aims to capture the system dynamics by directly modeling the value function, which is used to update the controller. More generalized, offline approaches are a promising avenue forward aim to achieve this over a range of similar systems or with historical datasets alone. Several frameworks have been proposed: *offline RL* (Levine, Kumar, Tucker, & Fu, 2020) aims to train the RL agent using *only* historical data from the system of interest. These methods are concerned with training a predictive model or value function that accounts for the uncertainty between the data samples and environment. On the other hand, *meta-RL* (McClement et al., 2021) trains an agent to learn from a distribution of similar dynamics and objectives; that is, the RL agent is not only trained to achieve optimal control, but also to learn an encoding of its environments, enabling it to generalize its policy to new systems. Latent representations of the system dynamics have been shown to be critical elements achieving real-world generalization from training in simulation in robotics applications (Lee, Hwangbo, Wellhausen, Koltun, & Hutter, 2020).

The ability to train a generalized RL agent and utilize historical data from individual systems is a significant opportunity for the process industries. We believe the primary benefits of these approaches are twofold: Increased scalability of RL algorithms and safety of the training procedure. As mentioned earlier, an outstanding issue with RL algorithms is the ability to reliably choose hyperparameters in the event of poor training performance. Therefore, training offline is a safety precaution. Moreover, algorithms that can effectively distill information from a range of different systems into a single agent will increase the scalability of RL by decreasing the cost of calibrating the agent to novel environments.

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.

Acknowledgments

We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and Honeywell Connected Plant. We would like to thank Jude Abu Namous for helping us obtain lab results by running countless experiments at Honeywell in North Vancouver. We would also like to thank, from Honeywell, Shadi Radwan for designing and assembling the two-tank system, and Stephen Chu for his help with instrumentation and system integration.

Appendix A. Further algorithmic and implementation details

The optimization of J in line (2) relies on knowledge of the Q-function (3). The critic network Q_{ϕ_i} is iteratively approximated using a deep neural network with training data from replay memory (RM). RM is a fixed-size collection of tuples of the form (s, u, s', r). The PID controller (actor "network") is given by (8) and denoted as μ, and the critic is Q_{ϕ_i}. More concretely, we utilize a combination of a feedforward

Algorithm 1: Deep Reinforcement Learning PID Controller

Output: Optimal PID controller $\mu_\theta(\cdot)$
Initialize: Actor tuning parameters θ, critic weights ϕ_1, ϕ_2, measurement dataset $\mathcal{D}_{\text{process}}$, replay memory $\mathcal{D}_{\text{RM}}$, step sizes α_a, α_c

1 Set target parameters equal to actor/critic parameters $\tilde{\theta} \leftarrow \theta$ and $\tilde{\phi}_1 \leftarrow \phi_1$, $\tilde{\phi}_2 \leftarrow \phi_2$
2 **for** each episode **do**
▷ In HMI:
3 Set $\bar{y} \leftarrow$ setpoint
4 **repeat**
▷ In HMI:
5 Load current PID parameters θ
6 Observe state s from environment
7 Execute control action $u \leftarrow \mu_{\text{PLC}}(s)$ ▷ μ_{PLC} is a PLC implementation of the actor ''network'' μ_θ.
8 Observe next state s' from environment
9 Store process data in $\mathcal{D}_{\text{process}}$
▷ Execute in Python:
10 **if** it is time to update **then**
11 Store transition tuples (s, u, r, s') in $\mathcal{D}_{\text{RM}}$ ▷ These structured tuples are derived from $\mathcal{D}_{\text{process}}$.
12 **for** each update step j **do**
13 Randomly sample a batch of transitions $\mathcal{B}$ from $\mathcal{D}$
14 Compute target actions $a' = \text{sat}(\mu_{\tilde{\theta}}(s') + \text{sat}(\epsilon))$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$
15 Compute targets $q = r + \gamma \min_{i=1,2} Q_{\tilde{\phi}_i}(s', a')$
16 Update critic weights as follows for $i = 1, 2$:
17 $\phi_i \leftarrow \phi_i - \alpha_c \nabla_{\phi_i} \left(\frac{1}{|\mathcal{B}|} \sum_{(s,u,r,s') \in \mathcal{B}} (q - Q_{\phi_i}(s, a))^2 \right)$
18 **if** $j \mod$ policy_delay $= 0$ **then**
19 Update policy weights ▷ Optional: Use Eq. (26).
20 $\theta \leftarrow \theta + \alpha \nabla_\theta \left(\frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} Q_{\phi_1}(s, \mu_\theta(s)) \right)$
21 Update target weights
$\tilde{\phi}_i \leftarrow \rho \tilde{\phi}_i + (1 - \rho)\phi_i$ for $i = 1, 2$
$\tilde{\theta} \leftarrow \rho \tilde{\theta} + (1 - \rho)\theta$
22
23 Save current PID parameters θ
24 **until** next step change or system reaches steady state

neural network and a recurrent neural network (RNN) for the critic. As discussed in Section 2.2, the state may contain past output information. The state vector passes through a specialized RNN called a "gated recurrent unit" (GRU) (Cho et al., 2014); the hidden state of the GRU is then passed as an input with the action through a DNN.

Algorithm 1 summarizes the training procedure and Table A.3 gives the hyperparameter names, symbols, and settings. The reader is referred to Fujimoto et al. (2018) and the references therein for more details and background on the TD3 algorithm. The primary hyperparameters we modified were the learning rates (for the actor and critic), the critic network, and the policy update delay. The other hyperparameters were set to the default values suggested by Achiam (2018). The learning rates and policy update delay did not deviate much from their original values either. The main design choice was with the critic network.

In Algorithm 1, note the distinctions between the measurement dataset $\mathcal{D}_{\text{process}}$ and replay memory $\mathcal{D}_{\text{RM}}$, and μ_θ and μ_{PLC}. $\mathcal{D}_{\text{process}}$ is

Table A.3
The recommended hyperparameter settings from this work.

Hyperparameter	Symbol	Nominal value
Actor learning rate	α_c	0.002
Critic learning rate	α_c	0.002
Discount factor	γ	0.99
Target network update rate	ρ	0.995
Policy update delay	policy_delay	2
Batch size	$\lvert\mathcal{B}\rvert$	256
Replay buffer size	$\lvert\mathcal{D}_{\text{RM}}\rvert$	10,000
Target noise	σ	0.2

a dataset storing measurements from the process, such as, setpoints, process variables, control variables. When it is time to update the PID parameters, new measurements in $\mathcal{D}_{\text{process}}$ are converted to the structured transition tuples (s, u, s', r) the RL algorithm expects, and store them in $\mathcal{D}_{\text{RM}}$ for training. μ_θ is defined as the actor "network" in our RL code, whereas μ_{PLC} uses the same parameters as μ_θ, but actually interacts with the system through a PLC. For the optimization of the actor and critic parameters, we use the Adam optimizer (Kingma & Ba, 2014) to implement the nominal update steps shown in Line 17 and Line 20.

Appendix B. Further experiments and discussion

We provide additional experiments and analysis. All the lab results are visualized in Fig. B.6 and performance statistics are reported in Table B.5. The experimental specifications corresponding to the labels given in these figures and tables are listed in Table B.4. RL-2, RL-3, RL-8, SIMC-3, Accutune-1, shown in boldface, refer to the five experiments shown in Section 6. The difference between RL-9 and RL-10 is the transient training data: RL-9 only switches between the setpoints 60 cm and 65 cm, where it hits the constraints at each step change.

We evaluate the performance on a variety of different reward functions based on the discussion in Section 5.2. In addition to Eq. (27) and the hybrid cost function l given in Eq. (24), we choose the reward functions:

$$-r(s_t, u_t) = e_t^2 + 0.1\Delta u_t^2 \tag{B.1}$$

$$-r(s_t, u_t) = l(s_t, u_t) + 0.1(\Delta u_t)^2. \tag{B.2}$$

The rationale for comparing the end training results of four different reward functions is to demonstrate their relative performance according to the metrics in Eq. (13).

Fig. B.6 shows a heatmap of the output performance of all the final tuning parameters from our experiments. Darker shades correspond to more time spent by the process variable at a value on the y-axis. This is a compact way of visualizing many experiments side-by-side while qualitatively capturing useful information such as overshoot or rise/settling time. Compared to the RL experiments involving the squared error based reward function (RL-1,RL-4), the absolute value based reward performs better and is more consistent between training experiments. The hybrid reward function (RL-5, RL-6, RL-7, yields similar, and sometimes superior, nominal performance to the absolute value based reward.

B.1. Experimental robustness

So far we have reported on the promising nominal performance of the RL based tuning. We also evaluate the empirical robustness of these results. With each set of final tuning parameters from our experiments, we perform the same sequence of step changes reported in Section 6 (65 cm, 60 cm, 63 cm, 60 cm), but with the following independent changes to the two-tank system: First, we tighten the outflow from the top tank to 50% on its valve (whereas before it was at maximum outflow); then, with the outflow back at its original state, we detune

Table B.4
Labels and specifications for RL, SIMC, and Accutune III experiments. In this paper we refer to them, for example, as "RL-1" for the first experiment in the RL category. The initialization refers to Eq. (28). Labels in boldface refer to the experiments shown in Section 6.

	RL (Unconstrained)							RL (Constrained)			SIMC					Accutune
	1	**2**	**3**	4	5	6	7	**8**	9	10	1	2	**3**	4	5	**1**
Reward	Eq. (B.1)	(27)	(27)	(B.1)	(B.2)	(24)	(24)	(27)	(24)	(24)	–	–	–	–	–	–
Initialization	$k_p = 4.0$	4.0	2.0	2.0	4.0	4.0	2.0	4.0	4.0	4.0	–	–	–	–	–	–
Setting	–	–	–	–	–	–	–	–	–	–	$T_c = 9.21$	15	20	25	30	Mode = "slow"

Table B.5
The overall performance of each experiment. Each cell shows the average statistic for its column plus or minus the standard deviation; the average is computed over the nominal performance and that of two robustness experiments where we independently adjusted the outflow of the tank and the flow tuning parameters.

	IAE	ISE	TV	TV_u	% OS	ST	M_s
RL-1	44.41 ± 9.57	28.33 ± 4.89	1.88 ± 0.23	11.14 ± 1.08	22.94 ± 10.77	134.67 ± 25.51	1.55
RL-2	43.76 ± 7.59	30.18 ± 3.85	1.56 ± 0.12	9.63 ± 0.50	10.95 ± 8.26	129.08 ± 34.22	1.35
RL-3	48.75 ± 4.55	34.00 ± 3.40	1.42 ± 0.05	8.90 ± 0.20	7.46 ± 2.03	153.17 ± 26.34	1.25
RL-4	67.96 ± 15.42	40.09 ± 10.96	2.32 ± 0.40	12.81 ± 1.92	48.20 ± 14.38	186.58 ± 28.46	1.42
RL-5	42.72 ± 8.03	27.87 ± 4.16	1.78 ± 0.21	10.67 ± 0.91	20.43 ± 9.74	128.75 ± 15.36	1.48
RL-6	40.85 ± 7.63	27.00 ± 3.81	1.69 ± 0.23	10.32 ± 1.14	17.33 ± 9.84	145.42 ± 41.48	1.50
RL-7	54.92 ± 6.65	34.65 ± 2.94	1.50 ± 0.09	9.06 ± 0.47	14.87 ± 4.67	190.58 ± 34.84	1.27
RL-8	42.75 ± 5.87	28.89 ± 3.75	1.56 ± 0.09	9.62 ± 0.41	11.38 ± 6.48	166.42 ± 20.03	1.36
RL-9	56.47 ± 13.25	33.90 ± 7.48	2.06 ± 0.25	11.45 ± 1.25	37.84 ± 9.84	163.08 ± 21.80	1.51
RL-10	40.18 ± 6.59	27.84 ± 3.61	1.58 ± 0.13	9.81 ± 0.52	11.08 ± 7.57	133.17 ± 5.63	1.41
SIMC-1	61.64 ± 17.36	36.73 ± 10.19	2.68 ± 0.72	14.29 ± 3.40	53.55 ± 14.48	162.08 ± 52.19	1.72
SIMC-2	56.16 ± 10.89	34.36 ± 6.05	1.73 ± 0.11	10.07 ± 0.49	24.11 ± 7.83	168.17 ± 29.17	1.36
SIMC-3	54.74 ± 6.49	35.47 ± 3.20	1.52 ± 0.02	9.17 ± 0.05	13.26 ± 4.97	193.75 ± 19.16	1.25
SIMC-4	57.61 ± 5.08	39.12 ± 3.50	1.38 ± 0.02	8.49 ± 0.11	8.38 ± 2.57	167.17 ± 47.65	1.19
SIMC-5	57.33 ± 12.18	38.77 ± 9.85	1.33 ± 0.06	8.31 ± 0.34	4.86 ± 1.79	141.67 ± 13.73	1.16
Accutune-1	70.09 ± 24.46	46.33 ± 16.94	1.59 ± 0.24	351.62 ± 201.09	8.98 ± 0.46	139.33 ± 65.07	1.41

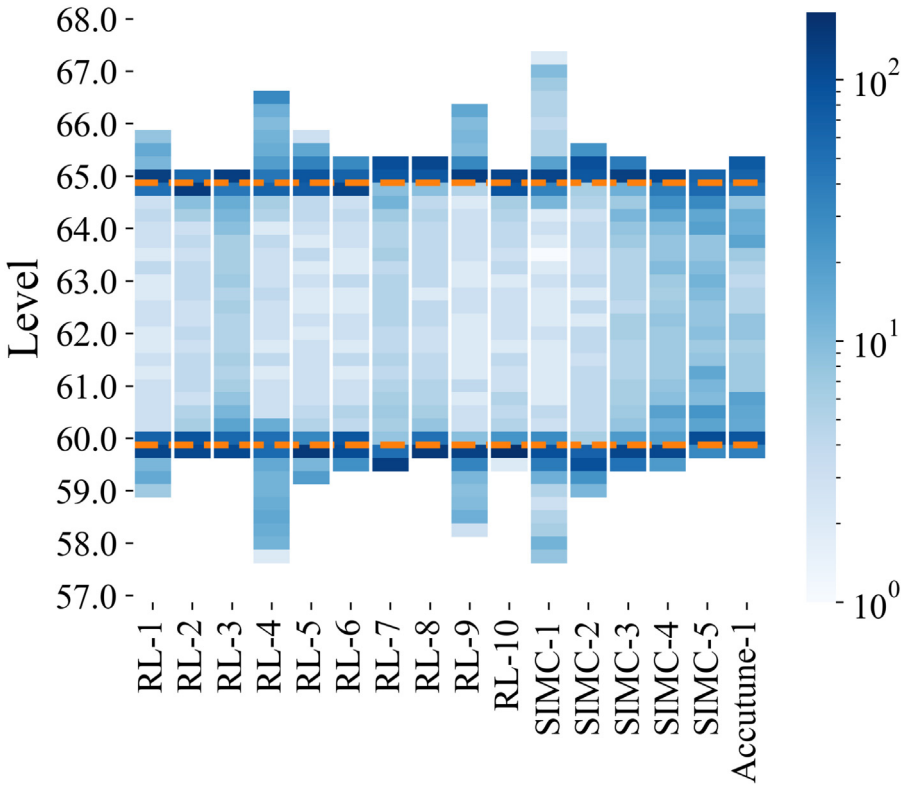

Fig. B.6. Heatmap of final performances across many different experiments. Darker colors indicate more time spent by the process variable at that location on the y-axis. Dashed lines at 60 cm and 65 cm indicate setpoints. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the flow controller. The flow controller parameters were originally set to $k_p = 0.2$, $k_i = k_p/3$, $k_d = 0.67k_p$, $T_f = 0.1$; we cut k_p in half for the second experiment.

Results for these experiments are reported in Table B.5. It encompasses the nominal performance and the performance of the two aforementioned robustness experiments. For each statistic, performance is measured based on the average across the four step changes. Each cell shows the average performance across these three experiments plus or minus the standard deviation.

Out of RL-1 – RL-4, RL-2 and RL-3 remain the most promising results. With the exception of ISE between RL-1 and RL-2, these two experiments based on the reward function from Eq. (27) are superior to RL-1 and RL-4, respectively, across all categories, both in terms of their average and standard deviation.

For the hybrid reward function, given in Eq. (24), it is apparent that RL-6 gives the best IAE and ISE, but with worse overshoot than RL-2. Moreover, RL-7 indicates a wider detrimental change in IAE and ISE based on the initial tuning than that of RL-2 and RL-3. For the constrained experiments, RL-8 and RL-10 achieved similar scores for TV, TV_u, and %OS, with RL-10 slightly superior in terms of IAE, ISE, and ST. Despite this, we consider the experiments based on the reward in Eq. (27) to be the overall best option; this is in terms of performance, generalization between training experiments, and robustness.

References

Achiam, J. (2018). Spinning up in deep reinforcement learning. URL: https://github.com/openai/spinningup.

Åström, K., & Hägglund, T. (1984). Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica*, *20*(5), 645–651. http://dx.doi.org/10.1016/0005-1098(84)90014-1.

Bao, Y., Zhu, Y., & Qian, F. (2021). A deep reinforcement learning approach to improve the learning performance in process control. *Industrial and Engineering Chemistry Research*, http://dx.doi.org/10.1021/acs.iecr.0c05678, acs.iecr.0c05678.

Berger, M. A., & da Fonseca Neto, J. V. (2013). Neurodynamic programming approach for the PID controller adaptation. *IFAC Proceedings Volumes*, *46*(11), 534–539. http://dx.doi.org/10.3182/20130703-3-FR-4038.00129.

Berner, J., Soltesz, K., Hägglund, T., & Åström, K. J. (2018). An experimental comparison of PID autotuners. *Control Engineering Practice*, *73*, 124–133. http://dx.doi.org/10.1016/j.conengprac.2018.01.006.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. (2016). OpenAI Gym. URL: https://github.com/openai/gym.

Brujeni, L. A., Lee, J. M., & Shah, S. L. (2010). Dynamic tuning of PI-controllers based on model-free reinforcement learning methods. In *ICCAS 2010* (pp. 453–458). Gyeonggi-do: IEEE, http://dx.doi.org/10.1109/ICCAS.2010.5669655.

Carlucho, I., De Paula, M., Villar, S. A., & Acosta, G. G. (2017). Incremental Q-learning strategy for adaptive PID control of mobile robots. *Expert Systems with Applications*, *80*, 183–199. http://dx.doi.org/10.1016/j.eswa.2017.03.002.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., et al. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078 [cs, stat]. URL: http://arxiv.org/abs/1406.1078.

Cui, Y., Zhu, L., Fujisaki, M., Kanokogi, H., & Matsubara, T. (2018). Factorial kernel dynamic policy programming for vinyl acetate monomer plant model control. In *2018 IEEE 14th international conference on automation science and engineering (CASE)* (pp. 304–309). Munich: IEEE, http://dx.doi.org/10.1109/COASE.2018.8560593.

Dogru, O., Wieczorek, N., Velswamy, K., Ibrahim, F., & Huang, B. (2021). Online reinforcement learning for a continuous space system with experimental validation. *Journal of Process Control*, *104*, 86–100. http://dx.doi.org/10.1016/j.jprocont.2021.06.004.

Forbes, M. G., Patwardhan, R. S., Hamadah, H., & Gopaluni, R. B. (2015). Model predictive control in industry: Challenges and opportunities. *IFAC-PapersOnLine*, *48*(8), 531–538. http://dx.doi.org/10.1016/j.ifacol.2015.09.022.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning* (pp. 1587–1596). PMLR.

Ge, Y., Li, S., & Chang, P. (2018). An approximate dynamic programming method for the optimal control of Alkai-Surfactant-Polymer flooding. *Journal of Process Control*, *64*, 15–26. http://dx.doi.org/10.1016/j.jprocont.2018.01.010.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv:1801.01290 [cs, stat]. URL: http://arxiv.org/abs/1801.01290.

Hausknecht, M., & Stone, P. (2016). Deep reinforcement learning in parameterized action space. arXiv:1511.04143 [cs]. URL: http://arxiv.org/abs/1511.04143.

Honeywell (2007). *UDC2500 universal digital controller product manual*. Honeywell.

Hoskins, J., & Himmelblau, D. (1992). Process control via artificial neural networks and reinforcement learning. *Computers & Chemical Engineering*, *16*(4), 241–251. http://dx.doi.org/10.1016/0098-1354(92)80045-B.

Joshi, T., Makker, S., Kodamana, H., & Kandath, H. (2021). Application of twin delayed deep deterministic policy gradient learning for the control of transesterification process. arXiv:2102.13012 [cs, eess]. URL: http://arxiv.org/abs/2102.13012.

Kaisare, N. S., Lee, J. M., & Lee, J. H. (2003). Simulation based strategy for nonlinear optimal control: Application to a microbial cell reactor. *International Journal of Robust and Nonlinear Control*, *13*(3–4), 347–363. http://dx.doi.org/10.1002/rnc.822.

Kim, J. W., Park, B. J., Yoo, H., Oh, T. H., Lee, J. H., & Lee, J. M. (2020). A model-based deep reinforcement learning method applied to finite-horizon optimal control of nonlinear control-affine system. *Journal of Process Control*, *87*, 166–178. http://dx.doi.org/10.1016/j.jprocont.2020.02.003.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. arXiv:1412.6980, Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems* (pp. 1008–1014). Citeseer.

Lawrence, N. P., Stewart, G. E., Loewen, P. D., Forbes, M. G., Backstrom, J. U., & Gopaluni, R. B. (2020). Optimal PID and antiwindup control design as a reinforcement learning problem. *IFAC-PapersOnLine*, *53*, 236–241. http://dx.doi.org/10.1016/j.ifacol.2020.12.129.

Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., & Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, *5*(47), eabc5986. http://dx.doi.org/10.1126/scirobotics.abc5986.

Lee, J. M., & Lee, J. H. (2008). Value function-based approach to the scheduling of multiple controllers. *Journal of Process Control*, *18*(6), 533–542. http://dx.doi.org/10.1016/j.jprocont.2007.10.016.

Lee, J. H., Shin, J., & Realff, M. J. (2018). Machine learning: Overview of the recent progresses and implications for the process systems engineering field. *Computers & Chemical Engineering*, *114*, 111–121. http://dx.doi.org/10.1016/j.compchemeng.2017.10.008.

Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv:2005.01643 [cs, stat]. URL: http://arxiv.org/abs/2005.01643.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (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. *Journal of Process Control*, *75*, 40–47. http://dx.doi.org/10.1016/j.jprocont.2018.11.004.

McClement, D. G., Lawrence, N. P., Loewen, P. D., Forbes, M. G., Backström, J. U., & Gopaluni, R. B. (2021). A meta-reinforcement learning approach to process control. *IFAC-PapersOnLine*, *54*, 685–692. http://dx.doi.org/10.1016/j.ifacol.2021.08.321.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. http://dx.doi.org/10.1038/nature14236.

Mowbray, M. R., Smith, R., Del Rio-Chanona, E. A., & Zhang, D. (2021). Using process data to generate an optimal control policy via apprenticeship and reinforcement learning. *AIChE Journal*, http://dx.doi.org/10.1002/aic.17306.

Nian, R., Liu, J., & Huang, B. (2020). A review On reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering*, *139*, Article 106886. http://dx.doi.org/10.1016/j.compchemeng.2020.106886.

Noel, M. M., & Pandian, B. J. (2014). Control of a nonlinear liquid level system using a new artificial neural network based reinforcement learning approach. *Applied Soft Computing*, *23*, 444–451. http://dx.doi.org/10.1016/j.asoc.2014.06.037.

Pandian, B. J., & Noel, M. M. (2018). Control of a bioreactor using a new partially supervised reinforcement learning algorithm. *Journal of Process Control*, *69*, 16–29. http://dx.doi.org/10.1016/j.jprocont.2018.07.013.

Petsagkourakis, P., Sandoval, I., Bradford, E., Zhang, D., & del Rio-Chanona, E. (2020). Reinforcement learning for batch bioprocess optimization. *Computers & Chemical Engineering*, *133*, Article 106649. http://dx.doi.org/10.1016/j.compchemeng.2019.106649.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv:1707.06347 [cs]. URL: http://arxiv.org/abs/1707.06347.

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* (pp. 257–262). Citeseer.

Shin, J., Badgwell, T. A., Liu, K.-H., & Lee, J. H. (2019). Reinforcement Learning – Overview of recent progress and implications for process control. *Computers & Chemical Engineering*, *127*, 282–294. http://dx.doi.org/10.1016/j.compchemeng.2019.05.029.

Shipman, W. J., & Coetzee, L. C. (2019). Reinforcement learning and deep neural networks for pi controller tuning. *IFAC-PapersOnLine*, *52*(14), 111–116. http://dx.doi.org/10.1016/j.ifacol.2019.09.173.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning* (pp. 387–395). PMLR.

Skogestad, S. (2003). Simple analytic rules for model reduction and PID controller tuning. *Journal of Process Control*, *13*(4), 291–309. http://dx.doi.org/10.1016/S0959-1524(02)00062-8.

Spielberg, S., Tulsyan, A., Lawrence, N. P., Loewen, P. D., & Bhushan Gopaluni, R. (2019). Toward self-driving processes: A deep reinforcement learning approach to control. *AIChE Journal*, *65*, http://dx.doi.org/10.1002/aic.16689.

Sutton, R. S., & Barto, A. G. (2018). *Adaptive computation and machine learning series*, *Reinforcement learning: An introduction* (2nd ed.). Cambridge, Massachusetts: The MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPs, Vol. 99* (pp. 1057–1063). Citeseer.

Syafiie, S., Tadeo, F., Martinez, E., & Alvarez, T. (2011). Model-free control based on reinforcement learning for a wastewater treatment problem. *Applied Soft Computing*, *11*(1), 73–82. http://dx.doi.org/10.1016/j.asoc.2009.10.018.

Wakitani, S., Yamamoto, T., & Gopaluni, B. (2019). Design and application of a database-driven PID controller with data-driven updating algorithm. *Industrial and Engineering Chemistry Research*, *58*(26), 11419–11429. http://dx.doi.org/10.1021/acs.iecr.9b00704.

Wang, Y., Velswamy, K., & Huang, B. (2018). A novel approach to feedback control with deep reinforcement learning. *IFAC-PapersOnLine*, *51*(18), 31–36. http://dx.doi.org/10.1016/j.ifacol.2018.09.241.

Yoo, H., Kim, B., Kim, J. W., & Lee, J. H. (2021). Reinforcement learning based optimal control of batch processes using Monte-Carlo deep deterministic policy gradient with phase segmentation. *Computers & Chemical Engineering*, *144*, Article 107133. http://dx.doi.org/10.1016/j.compchemeng.2020.107133.