Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Reinforcement Learning based Design of Linear Fixed Structure Controllers

Nathan P. Lawrence * **Gregory E. Stewart** ***
Philip D. Loewen * **Michael G. Forbes** ****
Johan U. Backstrom **** **R. Bhushan Gopaluni** **

* *Department of Mathematics, University of British Columbia, Vancouver, BC V6T 1Z2, Canada (e-mail: lawrence@math.ubc.ca, loew@math.ubc.ca).*
** *Department of Chemical and Biological Engineering, University of British Columbia, Vancouver, BC V6T 1Z3, Canada (e-mail: bhushan.gopaluni@ubc.ca)*
*** *Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: stewartg@ece.ubc.ca)*
**** *Honeywell Process Solutions, North Vancouver, BC V7J 3S4, Canada (e-mail: michael.forbes@honeywell.com, johan.backstrom@honeywell.com)*

Abstract: Reinforcement learning has been successfully applied to the problem of tuning PID controllers in several applications. The existing methods often utilize function approximation, such as neural networks, to update the controller parameters at each time-step of the underlying process. In this work, we present a simple finite-difference approach, based on random search, to tuning linear fixed-structure controllers. For clarity and simplicity, we focus on PID controllers. Our algorithm operates on the entire closed-loop step response of the system and iteratively improves the PID gains towards a desired closed-loop response. This allows for embedding stability requirements into the reward function without any modeling procedures.

Keywords: reinforcement learning, process control, PID control, derivative-free optimization

1. INTRODUCTION

Reinforcement learning (RL) is a branch of machine learning in which the objective is to learn an optimal strategy for interacting with an environment through experiences (Sutton and Barto, 2018). Traditional tabular methods for RL do not apply in continuous state or action spaces and are cumbersome in high-dimensional settings. For instance, the game of Go contains an intractable number of possible board configurations, which motivates the synthesis of deep learning with RL (Silver et al., 2016).

The success of RL methods reported in the literature is due to increasingly complicated algorithms. Combined with the inherent stochasticity due to random seeds or the underlying environment itself, as well as sensitivity to hyperparameters, the problem of reproducibility has become prevalent (Henderson et al., 2018; Islam et al., 2017). Several recent works have proposed simple algorithms that achieve performance competitive or superior to standard Q-learning and policy gradient methods (Salimans et al., 2017; Rajeswaran et al., 2017; Mania et al., 2018).

The applications of machine learning and RL to process control are relatively recent and limited in industrial implementation (Venkatasubramanian, 2019; Spielberg et al., 2019). Among the first approaches were Lee and Lee (2001, 2008), in which the authors develop an approximate dynamic programming approach with function approximation as a computationally efficient framework for model predictive control and gain scheduling, respectively. More recently, Spielberg et al. (2019) and Wang et al. (2018) proposed deep RL algorithms for control of discrete-time nonlinear processes. Both approaches are in the class of actor-critic methods, in which the actor (controller) is represented by a deep neural network. It is then shown empirically that networks represent a flexible class of controllers capable of learning to control complex systems. In contrast, PID controllers are simple and industrially established mechanisms for set-point tracking. However, setting the PID tuning parameters is known to be challenging and represents a nonlinear design problem. PID tuning thus represents both a challenging RL problem, and one which has the practical goal of being implemented in a production control system.

In this work, we develop an adaptive tuning algorithm based on a simple random search procedure for linear fixed-structure controllers. For simplicity in the development we focus on PID controllers. Our algorithm tracks a desired closed-loop step response by evaluating the distance between the desired response and the response gen-

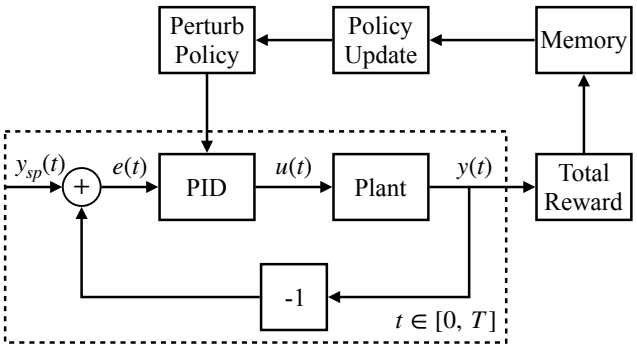

Fig. 1. A standard closed-loop structure is shown inside the dashed box. Arrows crossing the dashed line indicate the passing of some time-horizon $[0,\ T]$. Outside the dashed box, we store cumulative rewards based on slightly perturbed policies, which are used to update the policy with the finite-difference scheme described in section 4.2.

erated by slightly perturbing the policy which produces the controller parameters. We update the policy using a finite-difference approximation of the objective. Finally, although our method does not make use of a plant model, we focus on single-input single-output systems.

This paper is organized as follows: Section 2 gives a brief description of PID control. Section 3 outlines common methods in reinforcement learning. Section 4 describes our approach and algorithm for PID tuning based on simplified RL strategies discussed in section 3. Further, we contrast our approach to other RL-based tuning approaches. Finally, we show several simulation results in section 5.

2. PID CONTROL

In this section, we highlight some common strategies for PID control as they motivate our approach presented in the following section.

We use the parallel form of the PID controller:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{d}{dt} e(t). \tag{1}$$

Although the structure of a PID controller is simple, requiring only three tuning parameters k_p, k_i, k_d, adjusting these parameters to meet certain performance specifications is difficult in practice. Below we describe some performance metrics and strategies for tuning k_p, k_i, k_d.

2.1 Performance measures

Our proposed algorithm in section 4 does not rely on a plant model. Therefore, to evaluate the performance of the closed-loop step response of a system we use the accumulated error over some time horizon $[0, T]$. Common measures include the integral error, such as integral absolute error (IAE) or integral squared error (ISE):

$$\text{IAE} = \int_0^T |e(t)|\, dt \quad \text{and} \quad \text{ISE} = \int_0^T e(t)^2 dt. \tag{2}$$

Note that in order for the IAE or ISE to be a useful measure of performance, T should be large enough to measure the error accumulated through the closed-loop step response up to steady-state. In practice, we approximate the IAE and ISE through sampling at discrete time steps.

The criteria in (2) motivate our reward function in the following sections. However, IAE and ISE may also incorporate a weight on the magnitude of the control signal or its first-order difference.

2.2 Internal Model Control

Internal model control (IMC) utilizes a reference model in the feedback loop by incorporating the deviation of the plant output from that of the model. The resulting control structure often results in a PID controller for a large number of single-input single-output processes (Rivera et al., 1986). The simple PID tuning rules of Skogestad (2001) provide robust performance based on a first-order or second-order approximate model of the plant. Such a model can be obtained from a single step test of the plant. One then measures the plant gain, and time delay, as well as first and second order time constants. The resulting model can then be used to derive suitable PID gains for the plant using, for example, the SIMC rules (Skogestad, 2001). In section 5, we use SIMC to initialize the PID gains in algorithm 1.

3. REINFORCEMENT LEARNING

In this section, we provide a brief overview of common RL methods and their applications in process control. We then formulate the problem of tuning a PID controller as a RL problem and describe our approach to solving it.

3.1 The Reinforcement Learning Problem

For each *state* [1] x_i the agent encounters, it takes some *action* a_i, leading to a new state x_{i+1}. Upon taking action a_i, the agent receives a reward $r(x_i, a_i)$. The reward measures how desirable a particular state-action interaction is. To interact optimally with the environment the agent learns to maximize the cumulative reward due to a sequence of interactions. Formally, the environment is modeled by a Markov decision process with initial distribution $p(x_1)$ and transition probability $p(x_{i+1}|x_i, a_i)$. The agent then transitions from states to actions based on a conditional probability distribution π referred to as a *policy*. If $h = (x_1, a_1, r_1, , \ldots, x_j, a_j, r_j)$ is some trajectory generated by the policy π with sub-sequential states distributed according to p, we write $h \sim p^{\pi}(\cdot)$. If we assume the policy has some parametric structure given by π_θ, then the problem we aim to solve is:

$$\begin{aligned} &\text{maximize} && \mathbb{E}_{h\sim p^{\pi_\theta}(\cdot)}\big[R(h)|x_0\big] \\ &\text{over all} && \theta \in \mathbb{R}^n, \end{aligned} \tag{3}$$

where R denotes the accumulated reward received over an arbitrary trajectory.

Common approaches to solving (3) involve Q-learning and the policy gradient theorem (see Spielberg et al. (2019) and the references therein). In particular, variations of these methods involving function approximation have made RL

[1] The 'state' x_i is some measurement which characterizes the environment.

problems tractable in continuous state and action spaces (Lillicrap et al. (2015); Silver et al. (2014); Sutton et al. (2000)). Otherwise, discretization of the state and action spaces is necessary, leading to the "curse of dimensionality". These methods and variations thereof have led to remarkable results in the game of Go and simulated continuous control such as locomotion tasks in MuJoCo (Silver et al., 2016; Lillicrap et al., 2015; Todorov et al., 2012). Methods involving function approximation are better suited for these applications than discretization due to the huge number of possible state-action configurations.

3.2 Simple Reinforcement Learning Strategies

In contrast to the significant results described above, the issue of *reproducibility* has recently been examined in the RL community. Islam et al. (2017) and Henderson et al. (2018) compare performance of open-source implementations of popular policy gradient algorithms on various MuJoCo tasks. Factors such as hyperparameters, neural network architecture, or random seeds can have a drastic effect on algorithm performance.

In light of these issues, several recent works have proposed simpler algorithms with competitive or superior performance in benchmark MuJoCo tasks compared to the state-of-the-art performance reported with policy gradient methods.

Salimans et al. (2017) consider a gradient-free optimization method called *evolution strategies* (ES) as the basis for solving the optimization problem (3). The underlying idea of ES is to perturb the parameters of the policy, evaluate the policy, then combine the policy parameters with the best return. ES is simpler than the approaches highlighted in section 3.1 because there is no value function approximation procedure and its policy updates do not rely on computing the gradient of (3).

While Salimans et al. (2017) show that ES is competitive with standard RL benchmarks, their approach uses a neural network parameterization for the policy and includes several back-end algorithmic enhancements. To this end, Rajeswaran et al. (2017) achieve benchmark performance on standard MuJoCo locomotion tasks using a natural policy gradient algorithm with linear policies, thereby showing that neural networks are not necessary for these tasks. A synthesis of these approaches is proposed by Mania et al. (2018), in which a gradient-free algorithm for training linear policies is shown to achieve roughly equal overall performance on these MuJoCo locomotion tasks. The proposed algorithm of Mania et al. (2018) is then the basis of our approach.

4. PID FINE-TUNING VIA REINFORCEMENT LEARNING

In this section, we outline our strategy for PID tuning via reinforcement learning and contrast it with previous such approaches.

4.1 States, Actions, and Rewards

In our approach, we define the actions of the RL agent to be a vector of PID gains and the state to be a discretization of the closed-loop step response under these PID gains over a finite time horizon. Similarly, the target transfer function is represented by a vector of *target data*—a compatible discretization of the target transfer function step response. In principle, the target data may come from a simulated plant even if the RL algorithm is operating on a physical system. The reward for a state-action pair is then the mean absolute (or squared) error between the state and the target data. Concretely, we write $y(t)$ for the value of the closed-loop step response at the particular time t. Then we choose a sample count n and a vector of sampling times $T = [0, t_1, \ldots, t_{n-1}]$, and write $x = [y(0), \ldots, y(t_{n-1})]$. The corresponding target data $\tilde{y}(t_i)$ is contained in a vector $\tilde{x}$. Finally, the action is denoted by $K = [k_p, k_i, k_d]$. We express the reward for action K in state x as

$$r(x, K) = -\frac{1}{n} \|x - \tilde{x}\|_q^q \tag{4}$$

where the exponent $q \in \{1, 2\}$ is fixed.

4.2 Closed-Loop Transfer Function Tracking Algorithm

For Algorithm 1 we use the notation above. We refer to the process being controlled as P, and introduce functions $\texttt{feedback}(P, K)$ to describe the negative feedback loop for plant P under the PID controller with given gains K, and $\texttt{step}(H, T)$ to generate the vector of outputs at each time-step in T for some given system H. The goal is to determine a $k \times n$ matrix M for which optimal PID gains K can be expressed in terms of the state vector x via $K = K_0 + Mx$.

Algorithm 1 Closed-Loop Transfer Function Tracking

Output: Optimal PID gains K
Hyperparameters: stepsize $\alpha > 0$, standard deviation $\sigma > 0$ for policy exploration, number of sampling directions N, vector of sampling times T
Initialize: PID gains vector K_0 of length k, policy $M = 0_{k \times n}$
Initialize: Target data $\tilde{x}$ for times in T
Set $K = K_0$
for each episode **do**
  $x \leftarrow \texttt{step}(\texttt{feedback}(P, K), T)$
  for each j in $1, \ldots, N$ **do**
    Choose $\delta_j \in \mathbb{R}^{k \times n}$ at random.
    Perturb policy:
$$\begin{cases} K^+ \leftarrow (M + \sigma\delta_j)x + K_0 \\ K^- \leftarrow (M - \sigma\delta_j)x + K_0 \end{cases}$$
    Calculate closed-loop step responses:
$$\begin{cases} x_j^+ \leftarrow \texttt{step}(\texttt{feedback}(P, K^+), T) \\ x_j^- \leftarrow \texttt{step}(\texttt{feedback}(P, K^-), T) \end{cases}$$
    Evaluate rewards due to perturbation:
$$\begin{cases} r_j^+ \leftarrow \text{Reward at } x_j^+ \\ r_j^- \leftarrow \text{Reward at } x_j^- \end{cases}$$
  end for
  Evaluate standard deviation σ_r of the $2N$ rewards
  $M \leftarrow M + \frac{\alpha}{\sigma_r} \frac{1}{N} \sum_{j=1}^{N} \left[r_j^+ - r_j^-\right] \delta_j$
  $K \leftarrow Mx + K_0$
end for

Intuitively, Algorithm 1 is exploring the parameter space of PID gains centered at K_0; note that K_0 remains fixed throughout training. If we initialize K_0 with a zero vector then the first iteration simply operates with a zero-mean Gaussian matrix for the policy at line 10. Alternatively, if a set of PID gains is known to be stabilizing, or obtained through methods such as relay tuning or IMC, then we use those parameters to define K_0, and then evolve and improve them (guided by the reward objective) through evaluating the performance of small perturbations to the policy.

In line 9, the random matrices are determined by drawing each entry from an independent standard normal distribution. After the rewards are collected for several perturbed policies, in line 15 we update the policy using a scaled average of the finite-difference approximation for the gradient given by $\left[r^+ - r^-\right]\delta$. We scale by a fixed step-size $\alpha > 0$ as well as the reciprocal of the standard deviation of the rewards obtained from each sampled direction δ. Together, these scaling factors give smaller update steps when the current policy is more sensitive to perturbations, while larger steps are permitted when the perturbed policies yield similar rewards. Finally, line 16 updates K on the basis of the new policy and current output measurements. The small number of hyperparameters illustrates the simplicity and interpretability of the algorithm.

4.3 Differences from other RL-based tuning approaches

Here we highlight a few RL-based PID tuning approaches across various applications. With applications to wind turbine control, Sedighizadeh and Rezazadeh (2008) propose an actor-critic approach in which the PID gains are the actions taken by the actor at each time-step. Carlucho et al. (2017) develop an on-line discretization scheme of the state and action spaces, allowing for the implementation of the Q-learning algorithm for control of mobile robots. Finally, Brujeni et al. (2010) implement the classical SARSA algorithm for control of chemical processes. Their approach uses IMC to define a collection of PID gains which comprises the action space. At each time-step in the control of a physical continuous stirred tank heater the algorithm selects the best gains.

Our approach does not require training a neural network for value function approximation nor to represent an actor. Instead, our policy is given by a matrix whose size is determined by the number of sampling times in an episode and the number of tunable parameters for a linear controller (e.g., PI or PID). Further, our policy update procedure occurs on a different time scale than the sampling time. In particular, we update the policy based on entire closed-loop step responses, rather than at each time-step of a step response. This distinction avoids an important phenomenon associated with switching control strategies. Namely, if two controllers are known to be stabilizing, switching between them can still destabilize the closed-loop (see Example 1 in Malmborg et al. (1996)). Closed-loop transfer function tracking is then an intuitive approach for embedding performance specifications into a reward function without destabilizing the closed-loop with stabilizing controllers. With this view, it is justified to treat PID parameters as actions in the RL framework.

5. SIMULATION RESULTS

We present several simulation examples to illustrate our algorithm. The first example is a proof of concept in which we initialize the PID controller with unstable weights and construct a solution for Algorithm 1 to find. The second example initializes the PID parameters with SIMC (Skogestad, 2001), then Algorithm 1 updates the PID parameters to compensate for slow changes in the plant gain. See Appendix A for the hyperparameters used in Algorithm 1.

5.1 Example 1

In this example, we demonstrate our tuning method by constructing a desired closed-loop transfer function with a given plant model and set of target PID parameters.

Consider the following continuous-time transfer function:

$$P(s) = \frac{1}{(s+1)^3}. \tag{5}$$

We randomly initialize k_p, k_i, k_d around zero and set the desired parameters to be $k_p = 2.5, k_i = 1.5, k_d = 1.0$. The initial parameters may destabilize the plant as shown in figure 2. The target data then comes from uniform samples of the step response from the closed-loop transfer function $CP/(1 + CP)$ where C is the PID controller with the aforementioned target parameters.

We highlight several important notes about this experiment. First, the speed at which the algorithm finds the correct parameters is determined by the step-size α, the exploration parameter σ, and finally the relative distance between initial and target gains. We initialized the gains far away from the target to illustrate the trajectories of the gains during the learning procedure. Figure 3 shows the evolution of the PID parameters over the course of training. Note that the highlighted region indicates the parameters seen during the exploration described in line 10 of Algorithm 1. We run the simulation for many episodes to show the parameters hovering steadily around the solution. This behaviour aligns with the error (reward scaled by -1) curve shown in figure 4. We show several output responses in figure 2 corresponding to various levels of the error curve.

Our second remark is that the algorithm does not use any knowledge about the plant dynamics nor does it utilize a modeling procedure. Further, the PID control structure is only implicitly used, meaning the actions K directly influence the closed-loop, but could correspond to any linear controller. Finally, the target step response is user-specified, which makes Algorithm 1 amenable to performance specifications.

5.2 Example 2

In this example, we tune a PID controller using Algorithm 1 subject to drift in the plant gain and uncertainty in the time-delay.

Consider a nominal plant given by

$$P(s) = \frac{-0.02}{s+1}e^{-s}. \tag{6}$$

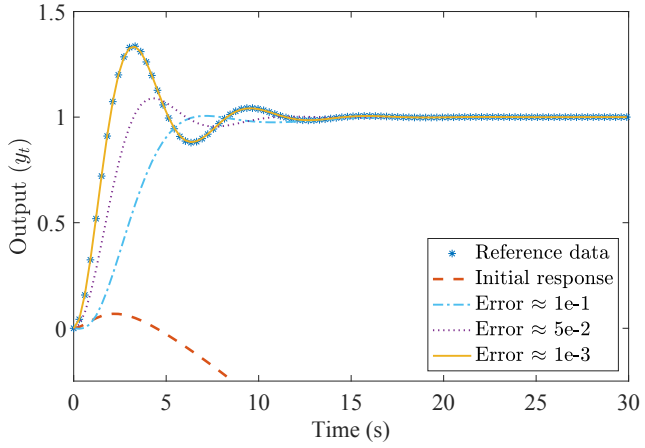

Fig. 2. The closed-loop step response at the beginning is shown with a dashed line, at end of training shown with a solid line, along with the reference data.

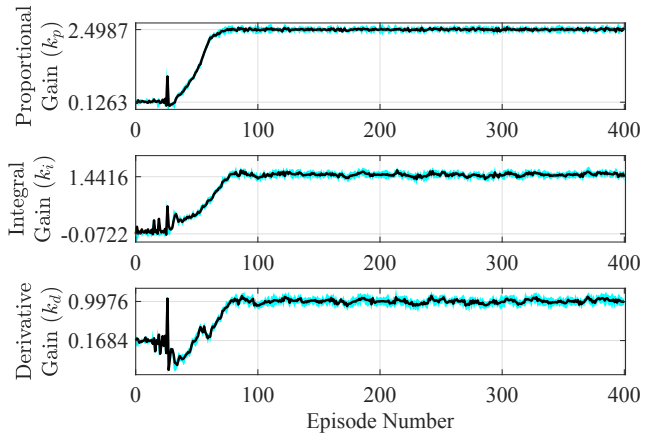

Fig. 3. The value of the updated PID gains at each iteration is shown in black. The highlighted region shows the range of values seen at each episode in line 10 of Algorithm 1

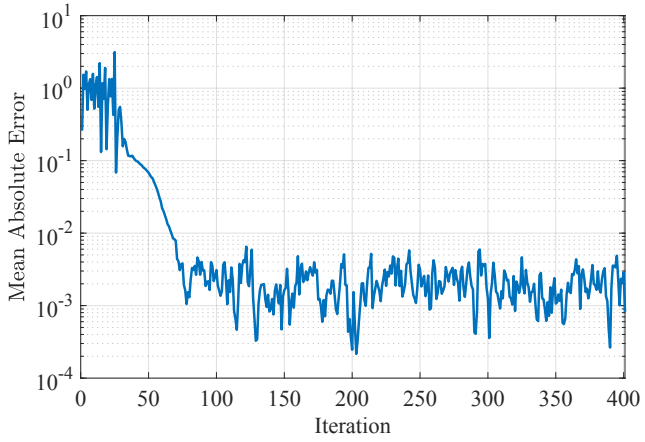

Fig. 4. The mean absolute error decreases on a logarithmic scale with the number of episodes.

We generate our target closed-loop data with P under the PID gains obtained with the SIMC tuning method referenced in section 2.2. These gains are K_0 in Algorithm 1. We use SIMC for initialization due to its simplicity as well as to illustrate the compatibility of Algorithm 1 with

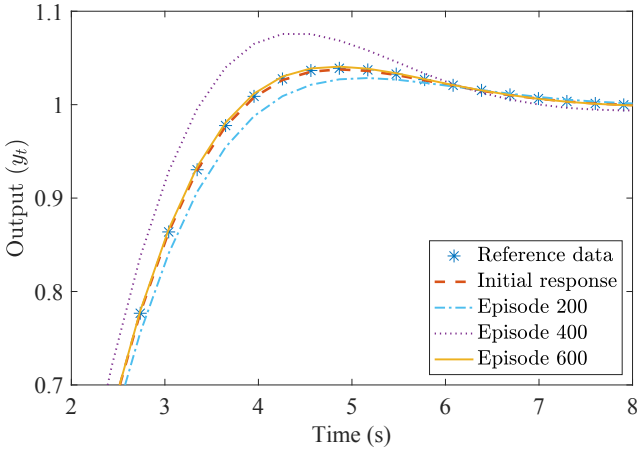

Fig. 5. The closed-loop step response corresponding to different plant gains and adjusted PID parameters so as to maintain initial performance.

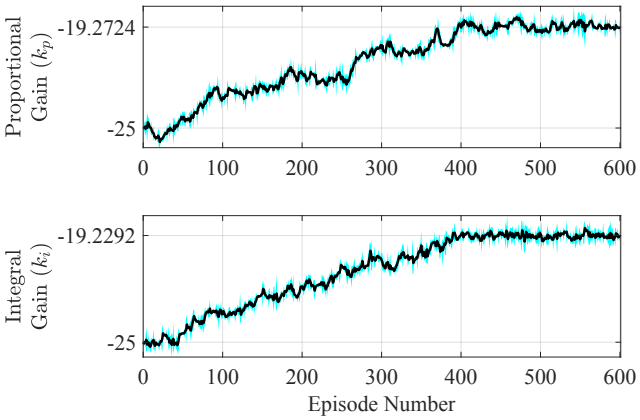

Fig. 6. The updated integral and proportional gains at each episode. The plant gain remains fixed after episode 400.

existing tuning methods. Note that we are not comparing our algorithm against SIMC.

At the beginning of each episode we slightly change the gain in the numerator of P; we also perturb the time-delay from 1 by adding a small amount of mean-zero Gaussian noise. The gain drifts linearly, so that at episode 400 its magnitude has increased by 30%. (The final numerator is -0.026.) At episode 400 we keep the plant gain fixed simply to observe the parameter updates in figure 6 for both changing and static plant gains. Figure 7 shows the error being maintained as the plant gain drifts and figure 5 shows snapshots of the closed-loop output response throughout training. It is worth mentioning that the error is steadily being maintained even at the beginning of training. This is due to the SIMC-based initialization and local parameter improvements made by Algorithm 1.

6. CONCLUSION

We have developed a simple and intuitive algorithm based on random search for tuning linear fixed-structure controllers. In the RL framework, we treat the entire closed-loop as the environment subject to new controller parameters as the actions. The reward function encodes

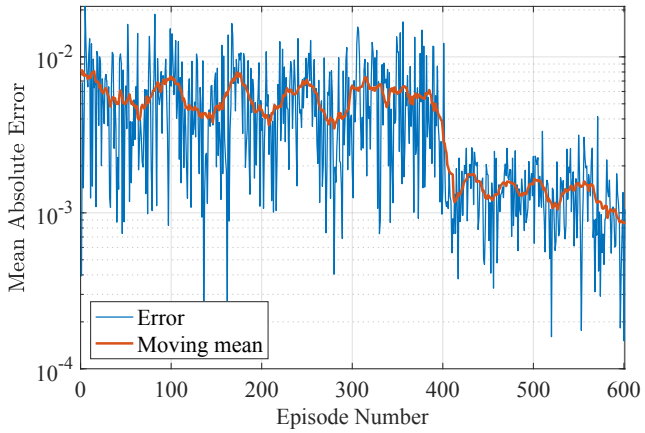

Fig. 7. The mean absolute error at each episode is shown in blue with a red curve overlapping it to show the moving average across 20 episodes.

performance requirements by considering a desired closed-loop transfer function. The simplicity of our algorithm allows for minimal hyper-parameter tuning, as well as straightforward initialization of the policy around a set of initial controller parameters.

ACKNOWLEDGEMENTS

We would like to thank Profs. Benjamin Recht and Francesco Borrelli of University of California, Berkeley for insightful and stimulating conversations. We would also like to acknowledge the financial support from Natural Sciences and Engineering Research Council of Canada (NSERC) and Honeywell Connected Plant.

REFERENCES

Brujeni, L.A., Lee, J.M., and 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., and Acosta, G.G. (2017). Incremental q-learning strategy for adaptive PID control of mobile robots. *Expert Systems with Applications*, 80, 183–199.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. (2017). Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*.

Lee, J.M. and Lee, J.H. (2001). Neuro-dynamic programming method for mpc1. *IFAC Proceedings Volumes*, 34(25), 143–148.

Lee, J.M. and Lee, J.H. (2008). Value function-based approach to the scheduling of multiple controllers. *Journal of process control*, 18(6), 533–542.

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

Malmborg, J., Bernhardsson, B., and Åström, K.J. (1996). A stabilizing switching scheme for multi controller systems. *IFAC Proceedings Volumes*, 29(1), 2627–2632.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, 1800–1809.

Rajeswaran, A., Lowrey, K., Todorov, E.V., and Kakade, S.M. (2017). Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, 6550–6561.

Rivera, D.E., Morari, M., and Skogestad, S. (1986). Internal model control: PID controller design. *Industrial & engineering chemistry process design and development*, 25(1), 252–265.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Sedighizadeh, M. and Rezazadeh, A. (2008). Adaptive PID controller based on reinforcement learning for wind turbine control. In *Proceedings of world academy of science, engineering and technology*, volume 27, 257–262. Citeseer.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., and Lanctot, M. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529, 484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*. Beijing, China.

Skogestad, S. (2001). Probably the best simple PID tuning rules in the world. In *AIChE Annual Meeting, Reno, Nevada*, volume 77.

Spielberg, S., Tulsyan, A., Lawrence, N.P., Loewen, P.D., and Bhushan Gopaluni, R. (2019). Toward self-driving processes: A deep reinforcement learning approach to control. *AIChE Journal*, 65(10), e16689.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R.S., McAllester, D.A., Singh, S.P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the Advances in Neural Information Processing Systems*, 1057–1063.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Venkatasubramanian, V. (2019). The promise of artificial intelligence in chemical engineering: Is it here, finally? *AIChE Journal*, 65(2), 466–478.

Wang, Y., Velswamy, K., and Huang, B. (2018). A novel approach to feedback control with deep reinforcement learning. *IFAC-PapersOnLine*, 51(18), 31–36.

Appendix A. IMPLEMENTATION DETAILS

We scripted Algorithm 1 in `MATLAB` and simulated the processes using the Control System Toolbox. We use different hyper-parameters for each example. However, we note that any set of hyper-parameters listed below lead to similar results for the each examples, but do not illustrate the parameter updates as clearly.

Example 1: $\alpha = 0.005$, $\sigma = 0.005$, $N = 10$

Example 2: $\alpha = 0.01$, $\sigma = 0.05$, $N = 1$ (2 policy perturbations per episode)

For all examples, samples were taken in increments of 0.30 seconds.

It is also possible to incorporate momentum in the policy update (Line 15 of Algorithm 1). This can lead to smaller and more steady errors and smoother parameter updates, but if the initial policy is unstable (e.g., Example 1) it can also exacerbate instability. We therefore omit it for simplicity.