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 prescribing 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 amount 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 superior or competitive performance 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 or 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 accepted controllers designed for set-point tracking. However, the setting of 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 passing 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 a finite-difference scheme described in section 4.2.Updated 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

A detailed review of PID control is beyond the scope of this paper. 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:

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

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):

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

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. Equations (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 as well as their applications in process control. We then formulate the problem of tuning a PID controller as a RL problem, then describe our approach to solving this problem.

3.1 The Reinforcement Learning Problem

For each state $x_i$, 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, but to interact optimally with the environment the agent learns to maximize the expected reward following each state-action interaction. 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 $\pi$ 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 $\pi$ 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 $\pi_\theta$, then the problem we aim to solve is:

$$\max \mathbb{E}_{h \sim p_\pi^\theta(\cdot)}[R(h)|x_0] \quad \text{over all} \quad \theta \in \mathbb{R}^n, \quad (3)$$

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

1 The ‘state’ $x_i$ is some measurement which characterizes the environment.
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 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), 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, therefore 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, 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 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 \( g(t) \) for the value of the closed-loop step response at the particular time \( t \); since the state is defined over a fixed time horizon \( T = [0, t_1, \ldots, t_{n-1}] \), we 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 at state \( x \) and action \( K \) as

\[
 r(x, K) = -\frac{1}{n} \| x - \tilde{x} \|_q^q
\]  

where \( q = 1 \) or 2 and is fixed.

4.2 Closed-Loop Transfer Function Tracking Algorithm

For Algorithm 1 we introduce the following notation: We refer to the process being controlled as \( P \); denote the vector of PID gains by \( K = [k_p, k_i, k_d] \); let \( T = [0, t_1, \ldots, t_{n-1}] \) be a vector of sampling times; feedback \( P, K \) refers to the negative feedback loop for plant \( P \) under the PID controller with fixed gains \( K \); finally, let \( \text{step}(H, T) \) refer to the vector of outputs at each time-step in \( T \) for some system \( H \).

The main idea behind Algorithm 1 is to linearly map the state vector \( x \) to a new set of PID gains \( K \). If we set \( K_0 \) to 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 can initialize the algorithm with the stabilizing tuning parameters, and then evolve and improve these gains (subject to the reward objective) through evaluating the performance of small perturbations to the policy. Note that in line 10 the vector \( K_0 \) remains fixed throughout training; intuitively, Algorithm 1 is exploring the parameter space of PID gains centered around \( K_0 \).

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 \( \frac{r^+ - r^-}{\delta} \). 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 \( \delta \). 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 minimal number of hyperparameters illustrates the simplicity and interpretability of our algorithm.
important phenomenon associated with switching control time-step of a step response. This distinction avoids an entire closed-loop step responses, rather than at each update procedure occurs on a different time scale than the linear controller (e.g., PI or PID). Further, our policy is determined by the number of sampling times in an actor. Instead, our policy is given by a matrix whose size for value function approximation nor to represent an algorithm selects the best gains.

\begin{algorithm}
\caption{Closed-Loop Transfer Function Tracking Algorithm}
\begin{algorithmic}
\State \textbf{Output:} Optimal PID gains $K$
\State Hyperparameters: stepsize $\alpha > 0$, standard deviation $\sigma > 0$ for policy exploration, number of sampling directions $N$, sampling times $T$
\State Initialize: PID gains vector $K_0$ of length $k$, policy $M = 0_{k \times n}$
\State Initialize: Target data $\tilde{x}$ for times $T$
\For{each episode}
\State $x \leftarrow \text{step(feedback}(P,K),T)$
\For{each $j$ in $1, \ldots, N$}
\State Sample $\delta_j \in \mathbb{R}^{k \times n}$ with independent standard normal entries
\State Perturb policy and evaluate performance of new PID gains:
\State $K^+ \leftarrow (M + \sigma \delta_j)x + K_0$
\State $K^- \leftarrow (M - \sigma \delta_j)x + K_0$
\EndFor
\State Collect output samples of closed-loop step response under new PID gains:
\State $x^+_j \leftarrow \text{step(feedback}(P,K^+),T)$
\State $x^-_j \leftarrow \text{step(feedback}(P,K^-),T)$
\State Evaluate rewards:
\State $r^+_j \leftarrow \text{Reward at } x^+_j$
\State $r^-_j \leftarrow \text{Reward at } x^-_j$
\EndFor
\State Evaluate standard deviation $\sigma_r$ of the $2N$ rewards
\State $M \leftarrow M + \frac{\alpha}{\sigma_r^2} \sum_{j=1}^N (r^+_j - r^-_j) \delta_j$
\State $K \leftarrow Mx + K_0$
\EndFor
\end{algorithmic}
\end{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 Rezaazadeh (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 as well as 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:

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

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 $\alpha$, the exploration parameter $\sigma$, 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 amendable 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

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

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 existing tuning methods.

At the beginning of each episode we slightly change the gain of (6); we also set the time-delay to be 1 plus a small amount of mean-zero Gaussian noise. The gain drifts such that at episode 400 there is a 30% change in gain in the negative direction (final gain 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. Note that the error is steadily being maintained even at the beginning of training.
We have developed a simple and intuitive algorithm based on random search for tuning of 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 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


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 sensitivity of the algorithm.

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. 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.