2019 IEEE Bombay Section Signature Conference (IBSSC)

Performance Analysis of Deep Q Networks and Advantage Actor Critic Algorithms in Designing Reinforcement Learning-based Self-tuning PID Controllers

Rajarshi Mukhopadhyay, Soutrik Bandyopadhyay, Dr. Ashoke Sutradhar, Dr. Paramita Chattopadhyay
Dept. of Electrical Engineering
IIEST, Shibpur
Howrah, India
rajarshi.mkrj@gmail.com, soutrik.band@gmail.com, as@ee.iiests.ac.in, paramita_chattopadhyay@yahoo.com

***Abstract*—Use of Reinforcement Learning (RL) in designing adaptive self-tuning PID controllers is a relatively new horizon of research with Q-learning and its variants being the predominant algorithms found in the literature. However, the possibility of using an interesting alternative algorithm i.e. Advantage Actor Critic (A2C) in the above context is relatively unexplored. In the present study, Deep Q Networks (DQN) and A2C approaches have been employed to design self-tuning PID controllers. Comparative performance analysis of both the controllers was undertaken in a simulation environment on a servo position control system, with various static and dynamic control objectives, keeping a conventional PID controller as a baseline. A2C based Adaptive PID Controller(A2CAPID) is more promising in trajectory tracking problems whereas DQN based Adaptive PID Controller(DQNAPID) is rather suitable for systems with relatively large plant parameter variations.**

***Index Terms*—Reinforcement Learning, Deep Q Learning, Advantage Actor Critic, Self Tuning PID**

I. Introduction

Proportional-Integral-Derivative (PID) controllers have been regarded as one of the most the favorable choice for the industrial process control community over a period of the last eighty years. The problem of adjusting their gain parameters (Tuning) to achieve optimal closed-loop performance, surfaced immediately after their early commercial production. Since then, they have remained the focus of interest for researchers. Most of the initial solutions were heuristic, model-free and based on certain handcrafted features of the plant, derived from open or closed-loop system response study in the time or frequency domain [1], [2]. Undoubtedly, these computationally simple and fast methods provided some early relief to control engineers, but soon it was realized that their efficacy is limited by a number of factors like:- the presence of third or higher-order dynamics and Multiple Input Multiple Output (MIMO) configurations, dead-time and different model uncertainties and nonlinearities. Gradually, development in digital computing platforms boosted the use of model-based methods with cost-function based optimization techniques for these problems [3]. However, all of these methods are off-line and suffer from the problem of online variations in plant parameters. The proposition of self-tuning adaptive PID controllers is an important improvement as it enables the controllers to dynamically adjust the gains on-the-fly without shutting down the plant. The fusion of different Soft Computing techniques along with conventional PID Controllers has proved to be extremely fruitful in designing adaptive self-tuning controllers. A number of research publications can be found in this avenue, proposing Fuzzy Logic [4] and Neural Network [5] based on-line self-adjusting algorithms for PID gain tuning. In addition to that, use of Genetic Algorithm (GA) [6], Particle Swarm Optimization (PSO) [7], Artificial Ant Colony [8], Bee Colony optimization [9] based algorithms are also found in literature. Scope of these algorithms are limited by their computational complexities when used in real-time for the process with unknown or uncertain models.

Reinforcement learning (RL) is a class of machine learning techniques in which, the agent (in control perspective it is the controller itself) learns to take an optimal decision (gain values for PID) in a stochastic environment by itself through a sequence of trial and error in order to maximize some sort of reward function [10]. Unlike supervised learning, no explicit critic is required to evaluate the actions taken by the agent. Compared to other optimization algorithms mentioned earlier, RL is effective when the learning environment (process model in the control perspective) is unknown or uncertain. Deep Q Networks (DQN) and Advantage Actor Critic methods (A2C) lie in the purview of RL and have been used individually in designing model-free adaptive controllers in the past [11]. However, a need was felt to compare these two algorithms in the context of Self-tuning Adaptive PID controllers keeping their static and dynamic performance requirements in mind. In the present study, two different configurations of self-tuning PID controllers have been implemented namely, DQN based

978-1-5386-7401-7/19/$31.00 ©2019 IEEE

Adaptive PID controller (DQNAPID) and A2C based Adaptive PID controllers (A2CAPID) on a servo position control system for comparative study.

In Section II a brief description of the DQN and A2C algorithms are provided. Section III deals with the problem formulation and the details of the simulation study. In Section IV, results obtained from simulation studies are provided with analysis. Section V concludes the essence of the entire study along with possible future avenues.

II. Reinforcement Learning

A. Markov Decision Processes

The backbone of Reinforcement Learning Problem is the concept of a Markov Decision Process(MDP), which is an idealized mathematical structure upon which algorithms of Reinforcement Learning are based. Markov Property of a Stochastic Process states that the conditional probability distribution of future states of the process depend only on the present state and not on the states preceding it. Formally, if $x_0, x_1, x_2, x_3...x_{n-1}, x_n$ be the states at discrete time steps under a stochastic process, the process is said to be Markov if -

$$Pr(X_n = x_n | X_{n-1} = x_{n-1}, ..., X_0 = x_0) = Pr(X_n = x_n | X_{n-1} = x_{n-1}) \quad (1)$$

A Reinforcement Learning(RL) Agent interacts with an MDP(environment) at each discrete time step by observing a set of variables(states) and enforces upon the environment a control action. The application of this action alters the state of the system. This action is then evaluated against a performance objective and subsequent reward(a numeric signal) is received by the agent. The reward structure is so chosen that maximizing the reward would, in some sense, achieve the desired predefined goal [10].

Fig. 1 shows the interaction between an RL Agent and the

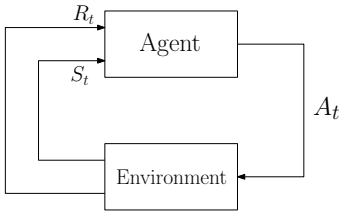

Fig. 1. Interaction between Agent and Environment

environment. At each time step t, the agent selects an action $A_t \in A(s)$, the action space. The state of the environment perceivable by the agent at time-step t is $S_t \in S$, where S denotes the observable state space of the agent. The agent receives a stochastic reward signal R_t in accordance with the ability to meet performance objectives. The agent must strive to maximize the total reward(G_t) it receives with a discounting factor(γ) (2).

$$G_t = \sum_{n=0}^{\infty} \gamma^n R_{t+n+1} \quad (2)$$

B. Policy and Value Functions

The mapping between the state and the subsequent action is called the policy (π) of the agent. Formally, the probability of the agent to choose an action a in the state s is called the policy of the agent over all action and state pairs (3).

$$\pi(a|s) = Pr\{A_t = a | S_t = s\} \; \forall \; a \in A(s), s \in S \quad (3)$$

The state-value function($v_\pi(s)$) of an MDP is defined as the expectation of future rewards given the current state (4).

$$v_\pi(s) = E_\pi\{G_t | S_t = s\} \; \forall \; s \in S \quad (4)$$

The action-value function ($q_\pi(s, a)$) of the MDP is defined as the expectation of future rewards given the current state s and choosing the action a (5).

$$q_\pi(s, a) = E_\pi\{G_t | S_t = s, A_t = a\} \; \forall \; s \in S, a \in A(s) \quad (5)$$

The relationship between action-value and state-value functions is given by

$$q_\pi(s, a) = R + \gamma v_\pi(s) \; \forall \; s \in S, a \in A(s) \quad (6)$$

C. Deep Q Network(DQN)

Q-Learning is an off-policy Temporal Difference Learning algorithm, where the action-value function is learned as episodes progress in time. The action-value function $q_w(s, a)$ is represented by a neural network, which acts as a Nonlinear function approximator and generalizes over the entire state and action space. Formally, $q_w(s, a) : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}$, where m is the dimension of the observable state space and n is the dimension of the action space. Let the parameter vector of the neural network q_w be $\mathbf{w}_q$, where $\mathbf{w}_q \in \mathbb{R}^d$. The parameters are optimized using Stochastic Gradient Descent so as to minimize the Temporal Difference Error (7).

$$J = R + \gamma \max_{a'} q_w(s', a') - q_w(s, a) \quad (7)$$

where R is the immediate reward obtained by the agent. The update rule for the parameter vector thus becomes

$$\mathbf{w}_q \leftarrow \mathbf{w}_q + \alpha[R + \gamma \max_{a'} q_w(s', a') - q_w(s, a)] \nabla_w q_w(s, a) \quad (8)$$

where α is the learning rate. The Algorithm is outlined as follows in [Algorithm 1]

D. Advantage Actor Critic (A2C) Method

Advantage Actor Critic Method is a Policy Gradient approach where the policy of an RL Agent is parameterized and is learned directly from experience.

In this approach, 2 networks - Policy Network, $\pi_\theta(s)$ (Actor) and Value Network, $v_\omega(s)$ (Critic) are used with $\theta \in \mathbb{R}^p$ and $\omega \in \mathbb{R}^r$ being the parameter vectors respectively.

The Advantage is defined as the difference between action-value function and state-value function. Formally,

$$A(s, a) = q(s, a) - v(s) \quad (9)$$

978-1-5386-7401-7/19/$31.00 ©2019 IEEE

Algorithm 1: DQN Algorithm

Initialize a network q(s,a);
Initialize parameter vector $\mathbf{w}_q$ randomly;
while *episodeNumber* < *totalEpisodes* **do**
  Initialize **s** from a sample value of set S;
  done ← False;
  while ***not*** *done* **do**
    Generate an action **a** from **s** using a policy derived from q;
    Take action **a** and observe R,s';
    $\mathbf{w}_q \leftarrow \mathbf{w}_q + \alpha[R + \gamma \max_{a'} q_w(s', a') - q_w(s, a)]\nabla_w q_w(s, a)$;
    $s \leftarrow s'$;
    if *s is Terminal* **then**
      done ← True;
    end
  end
  episodeNumber ← episodeNumber+1;
end

Algorithm 2: A2C algorithm

Initialize networks $v_\omega(s)$ and $\pi_\theta(s)$;
Initialize parameter vectors ω and θ randomly;
while *episodeNumber* < *totalEpisodes* **do**
  Initialize **s** from a sample value of set S;
  done ← False;
  while ***not*** *done* **do**
    Generate an action **a** from $\pi_\theta(\mathbf{s})$;
    Take action **a** and observe R,s';
    $\delta = R + \gamma v_\omega(s') - v_\omega(s)$;
    $\theta \leftarrow \theta + \alpha^\theta \delta \nabla_\theta \log(Pr(a|\pi_\theta(s)))$;
    $\omega \leftarrow \omega + \alpha^\omega \delta \nabla_\omega v_\omega(s)$;
    $s \leftarrow s'$;
    if *s is Terminal* **then**
      done ← True;
    end
  end
  episodeNumber ← episodeNumber+1;
end

Substituting (6) in (9)

$$A(s, a) = R + \gamma v(s') - v(s) \tag{10}$$

The parameter update rule of the Policy network is given by

$$\theta \leftarrow \theta + \alpha^\theta \delta \nabla_\theta \log(Pr(a|\pi_\theta(s))) \tag{11}$$

where δ is given by

$$\delta = R + \gamma v_\omega(s') - v_\omega(s) \tag{12}$$

The parameter update rule of the Value network is given by

$$\omega \leftarrow \omega + \alpha^\omega \delta \nabla_\omega v_\omega(s) \tag{13}$$

where the α^θ and α^ω are the step sizes for Policy and Value network respectively. The A2C algorithm is outlined in [Algorithm 2]

III. Problem Formulation and Implementation Details

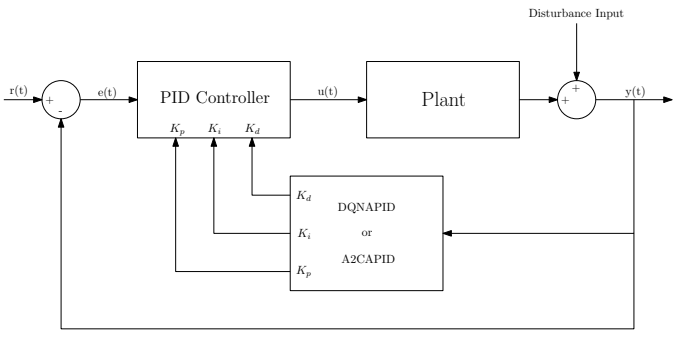

Fig. 2. Block Diagram of the proposed architecture

The plant chosen for the present study is that of a position control system of a servo motor [12] with the transfer function of plant given by

$$G_p(s) = \frac{19.9}{s(0.09s + 1)} \tag{14}$$

The two agents - DQNAPID and A2CAPID are compared and contrasted on various static and dynamic control objectives with a conventional PID controller acting as a baseline. The conventional PID controller was designed and tuned for optimal performance for the plant (14) using MATLAB PID Tuner App and the corresponding gains are -

Proportional Gain(K_p) = 1, Integral Gain(K_i) = 5, Derivative Gain(K_d) = 0.1

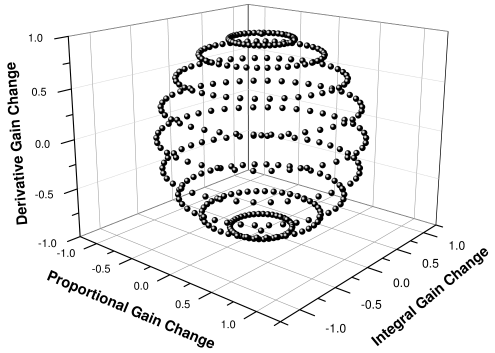

Fig. 3. The action space for the agents

The entire run of a simulation is divided into a number of episodes each of time T_e, consisting of numerous sample instants spaced evenly with sampling time (T_s). At each episode, the state of the two reinforcement learning agents (DQNAPID and A2CAPID) is defined as the coordinate in $\mathbb{R}^3$ representing (K_p, K_i, K_d) the gains of the PID Controller. At each episode, the agent chooses an action from the action

978-1-5386-7401-7/19/$31.00 ©2019 IEEE

space(a set of vectors on an unit sphere in $\mathbb{R}^3$ as shown in Fig. 3). The action represents a vector $(\Delta K_p, \Delta K_i, \Delta K_d)$. The state of the next episode (i.e. the controller gains) is obtained by the vector addition of the previous state and the action chosen by the Agent. This Gain vector is fed to the PID Controller(shown in Fig. 2) and the plant is simulated for the next episode. The performance of k^{th} episode is evaluated by the cost function of Integral Absolute Error(IAE), defined by

$$IAE_k = \int_{(k-1)T_e}^{kT_e} |r(t) - y(t)|dt \tag{15}$$

where $r(t)$ is the reference signal and $y(t)$ is the plant output. The goal of the RL agent is to minimize the IAE at each subsequent episode. The Reward signal for the k^{th} episode is constructed as follows

$$R_k = 3.5sgn(IAE_k - IAE_{k-1}) + 1.5 \tag{16}$$

A. Hyper-parameters and Network Architectures

The Hyper-parameters for the algorithms were chosen by a heuristic approach so as to ensure proper learning and are tabulated in Table I. Tables II - IV show the network architectures of the Neural networks used.

TABLE I
HYPER-PARAMETERS FOR THE RL AGENTS

Parameter	Value
Learning Rate (α)	0.1
Learning Rate Decay Rate	0.01
Discount Factor (γ)	0.9
Probability of Exploration (ϵ)	0.5
Epsilon Decay Rate	0.8
Action Space Dimension	200
State Space Dimension	3

TABLE II
NETWORK ARCHITECTURE FOR DQNAPID

Layer Name	Nodes	Activation Function
Input Layer	3	NA
Hidden Layer 1	16	tanh
Hidden Layer 2	16	tanh
Hidden Layer 3	48	tanh
Output Layer	200	linear

TABLE III
NETWORK ARCHITECTURE FOR A2CAPID-POLICY NETWORK

Layer Name	Nodes	Activation Function
Input Layer	3	NA
Hidden Layer 1	128	relu
Hidden Layer 2	256	relu
Output Layer	200	linear

TABLE IV
NETWORK ARCHITECTURE FOR A2CAPID-VALUE NETWORK

Layer Name	Nodes	Activation Function
Input Layer	3	NA
Hidden Layer 1	128	relu
Hidden Layer 2	256	relu
Output Layer	1	linear

B. Platform Details

Both DQNAPID and A2CAPID were implemented in a simulation environment written completely from scratch in Python v3.6. The plant dynamics model was represented as an ensemble of ordinary differential equations and were solved recursively using ODE Solver shipped with Python library of Python-Control. The Neural Networks were implemented using Pytorch [13] and Keras [14] Libraries.

IV. RESULTS AND ANALYSIS

A. Set Point Regularization

The RL Agents were pitted against each other in a set point regularization problem with set point of $r(t) = 100$. The output response for each of the RL agent is shown in Fig. 4 and the corresponding controller effort is shown in Fig. 5. From the plots, it is observed that both DQNAPID and A2CAPID track the reference satisfactorily. The peak overshoot, settling time and rise time of both these controllers are lesser than that of the PID controller as shown in Table V, with marginal difference between them.

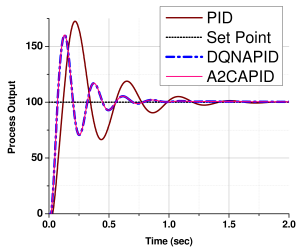

Fig. 4. Plant Output For Reference Tracking Problem

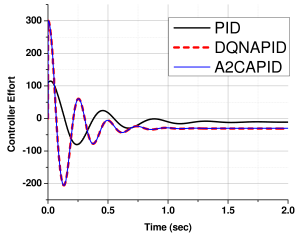

Fig. 5. Controller Effort For Reference Tracking Problem

TABLE V
STATIC PERFORMANCE INDICES FOR THE AGENTS

Performance Index	*PID*	*DQNAPID*	*A2CAPID*
Peak Overshoot (in %)	72.524	60.142	60.147
Settling Time (in sec)	1.62	0.96	0.98
Rise Time (in sec)	0.11	0.067	0.068

B. Sinusoidal Trajectory Tracking Problem

A sinusoidal trajectory of $r(t) = 100\cos(\omega t)$ with angular frequency(ω) varying from 0 to 20 rad/s linearly with time, was enforced on the agents. Output response for each of the RL agent is shown in Fig. 6. The agents were then tested on sinusoidal trajectories with ω of 0.63 rad/s, 6.3 rad/s and 63 rad/s. The output response of each of these frequencies is shown in Fig. 7 , Fig. 8 and Fig. 9 respectively.

From the plots, it is observed that both DQNAPID and A2CAPID track the trajectory satisfactorily with DQNAPID suffering relatively less attenuation as is the case with A2CAPID and PID controllers.

978-1-5386-7401-7/19/$31.00 ©2019 IEEE

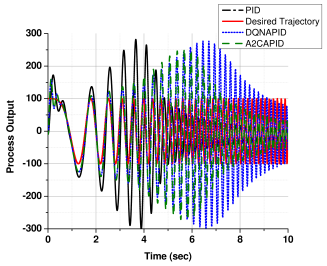

Fig. 6. Plant Output For Trajectory Tracking Problem

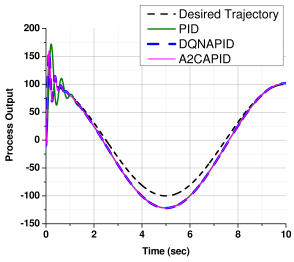

Fig. 7. Process output for $\omega = 0.63rad/s$

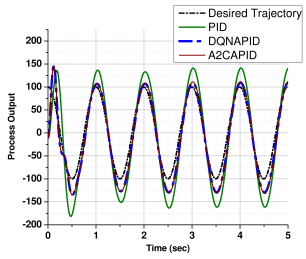

Fig. 8. Process output for $\omega = 6.3rad/s$

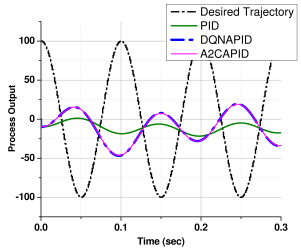

Fig. 9. Process output for $\omega = 63rad/s$

C. Reference Tracking Problem with Plant Parameter Variation and Output Disturbance

The agents were then tested on an experiment where the plant parameters are disturbed by 25% at time t = 5s. To judge the performance of the controllers in uncertain environments, an output step disturbance of 20% was further added along with plant parameter variation at time t = 5s to 6s. The agents were judged on their ability to track the reference of $r(t) = 100$. Fig 10 shows the plant output and Fig. 11 shows the controller effort under the experiment. The plots show that DQNAPID nullifies the effect of disturbance better than A2CAPID, which, in turn performs better than PID controller. The state trajectories of the DQNAPID and A2CAPID agents for the same experiment is shown in Fig. 12

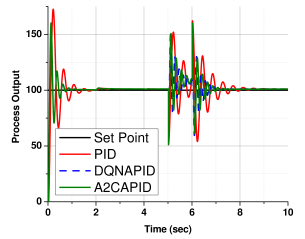

Fig. 10. Plant Output For Parameter Variation Problem

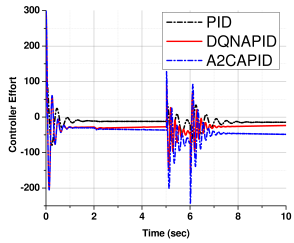

Fig. 11. Controller Effort For Parameter Variation Problem

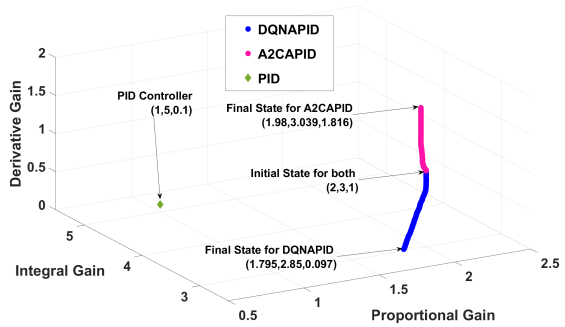

Fig. 12. State Trajectory for Parameter Variation Experiment for DQNAPID and A2CAPID

A summary of the IAE for all experiments is given in Table VI and the Final Controller Gains learned by the agents are given in Table VII.

TABLE VI
SUMMARY OF INTEGRAL ABSOLUTE ERROR FOR ALL THE ALGORITHMS IN VARIOUS TESTING SCENARIOS

Experiment Name	Integral Absolute Error for T = 10s		
	PID	*DQNAPID*	*A2CAPID*
Reference Tracking	38.8238	24.9878	24.9808
Trajectory Tracking	952.38	1021.58	919.09
Parameter Variation	76.84	44.99	46.85

TABLE VII
FINAL CONTROLLER GAINS LEARNED BY THE RL AGENTS.(ALL EXPERIMENTS WERE INITIALIZED WITH $K_p = 2, K_i = 3, K_d = 1$)

Experiment Name	Final Controller Gains					
	DQNAPID			A2CAPID		
	K_p	K_i	K_d	K_p	K_i	K_d
Reference Tracking	2.347	2.654	1.143	1.952	3.386	1.169
Trajectory Tracking	2.044	2.868	1.778	1.519	3.597	0.773
Parameter Variation	1.795	2.850	0.097	1.980	3.039	1.816

From the analysis of the simulation results it is evident that, both DQNAPID and A2CAPID have shown better transient and static performance compared to conventional PID Controllers under different circumstances - Set Point Regularization, Sinusoidal Trajectory Tracking and Plant Parameter Variation. Unlike the conventional PID , DQNAPID and A2CAPID don't require any plant related information a priori for successful tuning. However, DQNAPID performed slightly better than A2CAPID in both static and dynamic control objectives. In sinusoidal trajectory tracking, DQNAPID suffers significantly less attenuation than A2CAPID with increasing frequency and has a moderate phase lag resulting in a minor increase in IAE.

978-1-5386-7401-7/19/$31.00 ©2019 IEEE

V. Conclusion

Undoubtedly Reinforcement Learning has a promising future in developing adaptive PID controllers. The present investigation reveals that the system with the possibility of a wide range of plant parameter variation, DQNAPID is more suitable. On the other hand, in trajectory tracking, A2CAPID performs better. However, the convergence of both algorithms is greatly dependent on the initial values of the controller gains. In the area of Control Systems Research, Reinforcement Learning is in its infancy. Sometimes the control actions produced by these RL based algorithms may not be physically realizable. To address these concerns, new RL algorithms like Trust Region Policy Optimization(TRPO) and Proximal Policy Optimization(PPO) may be explored, which may lead to a new avenue of research.

VI. Acknowledgments

Rajarshi Mukhopadhyay gratefully acknowledges the financial assistance provided by the MHRD, Govt. of India in terms of doctoral fellowship.

References

[1] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *trans. ASME*, vol. 64, no. 11, 1942.

[2] C. C. Hang, K. J. Åström, and W. K. Ho, "Refinements of the ziegler–nichols tuning formula," in *IEE Proceedings D (Control Theory and Applications)*, vol. 138, no. 2. IET, 1991, pp. 111–118.

[3] A. O'Dwyer, *Handbook of PI and PID controller tuning rules*. Imperial College Press, 2009.

[4] S.-Z. He, S. Tan, F.-L. Xu, and P.-Z. Wang, "Fuzzy self-tuning of pid controllers," *Fuzzy sets and systems*, vol. 56, no. 1, pp. 37–46, 1993.

[5] R. Hernández-Alvarado, L. García-Valdovinos, T. Salgado-Jiménez, A. Gómez-Espinosa, and F. Fonseca-Navarro, "Neural network-based self-tuning pid control for underwater vehicles," *Sensors*, vol. 16, no. 9, p. 1429, 2016.

[6] J. Zhang, J. Zhuang, H. Du *et al.*, "Self-organizing genetic algorithm based tuning of pid controllers," *Information Sciences*, vol. 179, no. 7, pp. 1007–1018, 2009.

[7] X.-z. Li, F. Yu, and Y.-b. Wang, "Pso algorithm based online self-tuning of pid controller," in *2007 International Conference on Computational Intelligence and Security (CIS 2007)*. IEEE, 2007, pp. 128–132.

[8] I. Chiha, N. Liouane, and P. Borne, "Tuning pid controller using multi-objective ant colony optimization," *Applied Computational Intelligence and Soft Computing*, vol. 2012, p. 11, 2012.

[9] G. Yan and C. Li, "An effective refinement artificial bee colony optimization algorithm based on chaotic search and application for pid control tuning," *Journal of Computational Information Systems*, vol. 7, no. 9, pp. 3309–3316, 2011.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[11] I. Carlucho, M. De Paula, S. A. Villar, and G. G. Acosta, "Incremental q-learning strategy for adaptive pid control of mobile robots," *Expert Systems with Applications*, vol. 80, pp. 183–199, 2017.

[12] A. Ghosh, S. Sen, and C. Dey, "Design and real-time implementation of a fuzzy pi controller on a servo speed control application," in *2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)*, Feb 2015, pp. 384–387.

[13] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[14] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

978-1-5386-7401-7/19/$31.00 ©2019 IEEE