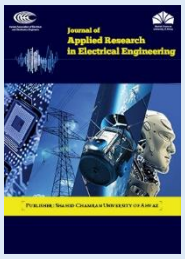
Shahid Chamran University of Ahvaz

Iranian Association of Electrical and Electronics Engineers

Journal of Applied Research in Electrical Engineering

E-ISSN: 2783-2864
P-ISSN: 2717-414X
Homepage: https://jaree.scu.ac.ir/



Research Article

DDPG-Based PID Gain Scheduling for DC Motor Speed Control: Exponential Reward and Heuristic Action Bounds

Mohammad Heidar Khamsehei Fadaei [1,*], and Fahimeh Sadat Ghasemi [2]

[1] *Department of Mechanical Engineering, Islamic Azad University, Saveh, Iran*

[2] *Department of Computer Engineering, Islamic Azad University, Qazvin, Iran*

* *Corresponding Author: moh13359@gmail.com*

Abstract: This paper presents a deep reinforcement learning approach for static gain scheduling of a PID controller in a DC motor speed control system. A deep deterministic policy gradient (DDPG) agent learns a direct mapping from the speed reference to fixed PID gains that remain constant during each ten-second episode. The action space is constrained by heuristic bounds derived from domain knowledge, ensuring that the proportional, integral, and derivative gains stay within safe ranges. An exponential reward function, which heavily penalises the integrated squared tracking error, is used to force the agent to virtually eliminate the steady-state offset. The DC motor model yields a second-order overdamped transfer function. After training over 1000 episodes with random references between 0.5 and 2.0 rad/s, the learned gains for a reference of 1.0 rad/s are compared with those obtained from MATLAB's PID Tuner. The reinforcement learning scheduler achieves approximately 34 % faster rise time, 23 % lower IAE, and 27 % lower ITAE than the classical tuner, although with a larger overshoot (13.1 % versus 7.04 %). Robustness analysis under ±20 % parameter variations shows that the RL controller maintains lower IAE in most scenarios. The training was repeated over five independent runs, showing very low variance (IAE standard deviation: 0.0019 rad·s), which confirms reproducibility. The main limitation is the static policy, where gains are not updated within an episode. The proposed method offers a competitive alternative to classical tuning when high tracking accuracy is required and the operating point changes slowly.

Keywords: Deep Deterministic Policy Gradient (DDPG), PID gain scheduling, DC motor speed control, Exponential reward function, Heuristic action bounds, Reinforcement learning.

Article history
Received 22 May 2026; Revised 19 June 2026; Accepted 10 August 2026; Published online 19 November 2026.
© 2026 Published by Shahid Chamran University of Ahvaz & Iranian Association of Electrical and Electronics Engineers (IAEEE)
How to cite this article
M. Heidar Khamsehei Fadaei, and F. Ghasemi,"DDPG-Based pid gain scheduling for dc motor speed control: exponential reward and heuristic action bounds," *J. Appl. Res. Electr. Eng.*, vol. 5, no. 1, pp. 12-24, 2026.
DOI: https://doi.org/10.22055/jaree.2026.52171.1312

Check for updates

1. INTRODUCTION

The DC motor is widely used in industrial control systems, robotics, and mechatronics. Its popularity stems from its simple structure, linear torque-speed characteristics, and ease of speed regulation. Among various control strategies, the PID controller remains the most common choice in practice due to its straightforward structure and interpretable parameters.

However, tuning the three gains (K_p, K_i, and K_d) for a DC motor is not trivial. This task becomes even more challenging when plant parameters are uncertain or when the operating point varies. Classical tuning methods such as Ziegler-Nichols, Cohen-Coon, or manual trial-and-error are often time-consuming. They also yield suboptimal performance in terms of overshoot, settling time, or steady-state error [1].

To address these limitations, automated PID tuning tools have been integrated into software environments such as MATLAB's PID Tuner. These tools linearise the plant at a given operating point and optimise the gains based on frequency-domain criteria, including gain margin and phase

margin [2]. While effective for linear time-invariant systems, such methods are inherently local. They may fail when the system exhibits nonlinearities, actuator saturation, or frequent reference changes. Moreover, they do not learn from experience, nor can they adapt to different reference speeds without retuning.

In recent years, reinforcement learning (RL) has emerged as a powerful paradigm for adaptive controller design [3]–[5]. Unlike supervised learning, RL does not require labelled optimal actions. Instead, an agent learns by interacting with the environment and maximising a cumulative reward signal. Deep deterministic policy gradient (DDPG) [6], an off-policy actor-critic algorithm for continuous action spaces, has shown promise in tuning PID gains online. Several recent studies have applied DDPG to PID gain scheduling for DC motors and related electromechanical systems.

Despite these advances, most existing RL-based PID tuning studies use generic quadratic rewards and do not impose explicit safety constraints on the action space. Furthermore, reproducibility analyses—such as repeating training with multiple random seeds—are rarely reported, making it difficult to assess the variance of the learned policies.

This paper proposes a DDPG-based framework for static PID gain scheduling of a DC motor speed control system. The key novelties are fourfold:

1. **Exponential reward design** – Instead of the conventional quadratic cost, we propose $R = \exp(-10 \int e^2 dt)$. This creates a steep gradient for small tracking errors and forces the agent to virtually eliminate the steady-state offset.
2. **Constrained action space with heuristic bounds** – The agent is not allowed to output destabilising gains during either training or deployment, which improves safety.
3. **Systematic comparison with a classical frequency-domain tuner** – We compare our results with MATLAB's PID Tuner on the same plant model.
4. **Rigorous robustness and reproducibility analysis** – We evaluate performance under ±20% parameter variations and repeat the training five times with different random seeds to assess variance.

Unlike conventional adaptive PID designs that update gains continuously, our approach learns a static policy. The policy maps the reference speed to a fixed set of gains for the entire simulation horizon (one action per 10-second episode). This design choice simplifies deployment, avoids instability from high-frequency gain adaptation, and still provides performance improvements over classically tuned fixed-gain PID. Therefore, the method is best described as reference-dependent gain scheduling rather than fully adaptive control—a distinction clarified throughout the paper.

The remainder of this paper is organised as follows. Section 2 reviews classical and automated PID tuning, reinforcement learning for control, and recent RL-based PID tuning studies for DC motors. Section 3 describes the mathematical model of the DC motor. Section 4 presents both the classical PID tuning and the proposed DDPG-based gain scheduling. Section 5 explains the simulation environment and training setup. Section 6 reports the comparative results, including robustness and saturation analyses. Section 7 discusses the findings, limitations, and reproducibility. Section 8 concludes the paper.

2. Literature Review

2.1. Classical and Automated PID Tuning

PID controllers have been the workhorse of industrial control for decades. Their simplicity and effectiveness are the main reasons [7]. Classical tuning methods such as Ziegler-Nichols, Cohen-Coon, and trial-and-error rely on empirical rules or step response characteristics. Although these methods are straightforward, they often give suboptimal performance in terms of overshoot, settling time, or steady-state error. This is especially true for plants with unknown dynamics or varying operating conditions [8].

To overcome these limitations, automated tuning tools have been developed. MATLAB's PID Tuner linearises the plant at a given operating point. It then optimises the gains based on frequency-domain criteria (gain and phase margins) [9]. Such methods are effective for linear time-invariant systems. However, they are inherently local. They may fail when the system has nonlinearities, actuator saturation, or frequent reference changes. Also, they do not learn from past experiences and cannot adapt to new operating points without retuning [10].

2.2. Reinforcement Learning for Control

Reinforcement learning (RL) has emerged as a promising paradigm for adaptive control [11]. Unlike supervised learning, RL does not require labelled optimal actions. Instead, an agent learns a policy by interacting with the environment and maximising a cumulative reward signal. Deep RL combines neural networks with RL algorithms, enabling the handling of high-dimensional state and action spaces [12].

Among continuous-action RL algorithms, deep deterministic policy gradient (DDPG) [13] has received considerable attention. DDPG is an off-policy actor-critic method. It efficiently handles continuous action spaces by using a deterministic policy and a Q-function approximator. Techniques such as target networks and experience replay have improved its stability [14]. Recent systematic reviews [15] confirm that DDPG is one of the most widely adopted deep RL algorithms for continuous control tasks. These tasks include robotics, autonomous driving, and process control.

Nazarzehi and Raisi [16] proposed a deep learning framework for underwater object detection using multi-robot systems. Their work is based on YOLOv6 with an attention module. Although it focuses on perception rather than control, the use of deep neural networks in robotic systems is relevant to our RL approach, which also employs deep networks.

Richter et al. [17] analysed inference processes in deep convolutional neural networks using saturation and logistic regression probes. Their insights on the information flow through the network layers can also inform the design of RL agents , which also use deep neural networks for the approximation of the policy and the value function.

2.3. RL-Based PID Gain Tuning for DC Motors and Related Systems

Several recent studies have applied DDPG and other RL algorithms to tune PID gains for DC motors and similar electromechanical systems.

Ahmed et al. [18] developed a DDPG-based framework for DC motor speed control. They used a standard quadratic reward and demonstrated good tracking performance. However, they did not include a robustness analysis against parameter variations.

Tufenki et al. [19] compared off-policy DDPG with on-policy methods (e.g., PPO) for a DC motor plant. They concluded that DDPG provides smoother convergence and better sample efficiency. However, their action space was not constrained, which led to occasional unstable gains during exploration.

Tufenkci et al. [20] extended the comparison to twin-delayed DDPG (TD3) and DDPG for a twin-rotor MIMO system (TRMS). They highlighted the importance of anti-windup mechanisms when actuator saturation is present, a point that many studies overlook.

Davatgar et al. [21] applied DDPG to tune a PID controller for a gas turbine system, a more complex nonlinear plant. They showed that the RL-tuned controller outperforms classical tuners in settling time and overshoot under nominal conditions.

Demircioglu and Bakır [22] provided a comprehensive hyperparameter analysis for RL-driven PID tuning in mass-spring systems. They showed that the choice of the reward function (quadratic versus absolute error) has a strong impact on the stability of learning.

A multi-agent reinforcement learning framework based on TD3 for adaptively tune PID parameters in boiler-turbine system were proposed by Kruthika and Paneerselvam [23]. The results demonstrated that the proposed approach was more robust to variations in load.

Ghamari et al. [24] designed a robust adaptive cascade fractional-order PID controller enhanced by RL for speed regulation of brushless DC motors in electric vehicles. Their work combines fractional-order calculus with RL, achieving high accuracy but at the cost of increased computational complexity.

Vijaya Lakshmi and Manimozhi [25] conducted a real-time hardware-in-the-loop (HIL) validation of a DDPG-based controller for a DC servo motor, effectively connecting simulation with practical deployment.

Zribi and Taouil [26] proposed an adaptive RL-based tuning of neural PID controllers for nonlinear systems, where the PID gains are continuously updated by a secondary neural network. However, the continuous adaptation adds extra complexity and potential instability.

2.4. Recent Robust Control Methods for DC Motors

The following three recent papers, suggested by the reviewer, focus on robust control of DC motors in the presence of time delays and disturbances. Although they do not use reinforcement learning, they are relevant as they address the same plant (DC motor). They also highlight challenges such as input delay, disturbance rejection, and time-varying delays. Our work, in contrast, uses RL to learn a static gain scheduler without requiring an explicit model of delays or disturbances.

Maleki and Moradi (2022) [27] proposed a robust H∞ control method for a DC motor in the presence of input delay and disturbance. They used a predictor-based approach to compensate for the delay. Their method guarantees stability and performance under the worst-case disturbances. Compared to their work, our RL-based scheduler does not assume a known delay model but learns gains from data.

Maleki and Moradi (2025) [28] designed a robust memory state feedback controller for a DC motor subject to time delays and norm-bounded disturbances, using linear matrix inequalities (LMIs). Their controller utilises past state information to improve robustness. While their method is mathematically rigorous, it requires solving LMIs and assumes known bounds on delays. In contrast, our DDPG approach is more flexible and can be extended to nonlinear or uncertain environments.

Labbaf Khaniki and Moradi (2026) [29] introduced a delay-scheduled adaptive observer control strategy for nonlinear systems with time-varying delays. Their observer estimates the system state and adapts to changing delays. This is a sophisticated adaptive method, but it requires significant analytical effort. Our RL-based method offers a simpler, data-driven alternative that can be applied without profound of the delay structure.

These three papers demonstrate that robust control of DC motors in the presence of delays and disturbances remains an active research area. Our work complements these approaches by showing that a DDPG agent can learn a static PID gain scheduler that performs well under parameter variations, even without explicit delay compensation.

2.5. Research Gaps and Contribution of This Work

Despite the advances mentioned above, several gaps remain:

- **Reward design** – Most works use generic quadratic rewards that penalise instantaneous error. However, they do not aggressively enforce zero steady-state error. A reward that strongly penalises cumulative error over the entire episode may lead to better long-term performance.
- **Action space constraints** – Many studies allow the agent to propose arbitrarily high or low gains during exploration. This approach can destabilise the plant and slow down training. Heuristic bounds derived from domain knowledge are often ignored.
- **Reproducibility** – With very few exceptions [15], authors only occasionally repeat training across multiple random seeds. Consequently, the variance of the learned policies and the statistical significance of performance claims remain unknown.
- **Robustness analysis** – Most works test only on the nominal plant. Systematic evaluation under parameter uncertainties (e.g., ±20% variation in resistance, inertia, or friction) is often missing.
- **Static vs. dynamic scheduling** – Many RL designs

update gains continuously (at every time step). This may cause high-frequency gain switching and complicate deployment. A simpler static scheduler that maps the reference to a fixed gain set per episode is underexplored.

To fill these gaps, this paper proposes a DDPG-based static PID gain scheduler for a DC motor with the following distinguishing features: (i) an exponential reward that heavily penalises integrated squared error, (ii) a constrained action space with heuristic bounds ensuring safe exploration, (iii) a rigorous robustness analysis under ±20% parameter variations, (iv) a reproducibility study across five independent random seeds, and (v) a static policy that issues one gain set per episode, simplifying deployment while still improving upon classical tuning.

3. DC Motor Modelling

A separately excited DC motor is considered, where the armature voltage $V_a(t)$ is the control input and the rotor angular speed $\omega(t)$ is the output. Fig. 1 shows the block diagram representation of the DC motor model, corresponding to the schematic provided in the Simulink documentation.

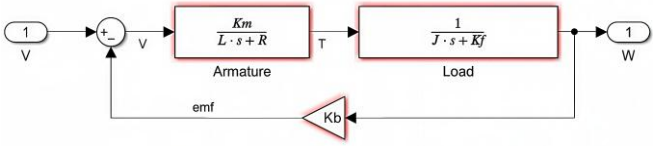


Fig. 1: Block diagram representation of the DC motor model.

The electrical dynamics of the armature circuit are given by:

$$V_a(t) = L\frac{di_a(t)}{dt} + Ri_a(t) + V_b(t) \tag{1}$$

where $i_a(t)$ is the armature current, L is the armature inductance, R is the armature resistance, and $V_b(t)$ is the back electromotive force (back-EMF) proportional to the rotor speed:

$$V_b(t) = K_b\omega(t) \tag{2}$$

Here K_b is the back-EMF constant (V·s/rad).

The mechanical dynamics are described by Newton's second law for rotation:

$$J\frac{d\omega(t)}{dt} = T_m(t) - K_f\omega(t) \tag{3}$$

where J is the rotor moment of inertia (kg·m²), K_f is the viscous friction coefficient (N·m·s), and $T_m(t)$ is the electromagnetic torque produced by the motor. The torque is proportional to the armature current:

$$T_m(t) = K_m i_a(t) \tag{4}$$

with K_m the torque constant (N·m/A). For an ideal DC motor, $K_m = K_b$ in SI units (both equal to 0.1 in this study).

Taking Laplace transforms (zero initial conditions) of (1)–(4) and eliminating intermediate variables yields the transfer function from armature voltage to angular speed:

$$G(s) = \frac{\Omega(s)}{V_a(s)} = \frac{K_m}{(Js + K_f)(Ls + R) + K_m K_b} \tag{5}$$

The numerical parameters used in this study are listed in Table 1.

Table 1: Numerical Parameters of the DC Motor Plant.

Parameter	Symbol	Value	Unit
Armature Resistance	R	2.0	Ω
Armature Inductance	L	0.5	H
Torque Constant	K_m	0.1	N·m/A
Back-EMF Constant	K_b	0.1	V·s/rad
Viscous Friction Coefficient	K_f	0.2	N·m·s
Rotor Moment of Inertia	J	0.02	kg·m²

Substituting the numerical values into (5) gives:

$$G(s) = \frac{0.1}{(0.5s + 2.0)(0.02s + 0.2) + (0.1)(0.1)} \tag{6}$$

Expanding the denominator:

$$(0.5s + 2.0)(0.02s + 0.2) = 0.01s^2 + 0.14s + 0.4$$

Adding $K_m K_b = 0.01$ yields:

$$G(s) = \frac{0.1}{0.01s^2 + 0.14s + 0.41} \tag{7}$$

Multiplying numerator and denominator by 100:

$$G(s) = \frac{10}{s^2 + 14s + 41} \tag{8}$$

The plant is second-order with natural frequency $\omega_n = \sqrt{41} \approx 6.403$ rad/s and damping ratio $\zeta = \frac{14}{2\sqrt{41}} \approx 1.093$. The system is overdamped ($\zeta > 1$) and exhibits no overshoot in its open-loop step response. However, with a PID controller, the closed-loop response can introduce overshoot. The steady-state gain is $G(0) = 10/41 \approx 0.2439$, meaning that without a controller, a 1 V input yields about 0.244 rad/s. Therefore, a PID controller is essential to achieve accurate speed tracking.

4. Controller Design

Two PID controller design approaches are presented and compared. The first is a classical frequency-domain tuning using MATLAB's PID Tuner (Section 4.1). The second is a proposed DDPG-based reinforcement learning method for static gain scheduling (Section 4.2).

4.1. Classical PID Tuning (MATLAB PID Tuner)

The PID controller in the time domain is defined as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{9}$$

Where $e(t) = \omega_{\text{ref}}(t) - \omega(t)$ is the speed error. In the Laplace domain, the ideal PID transfer function is:

$$C(s) = K_p + \frac{K_i}{s} + K_d s \tag{10}$$

For practical implementation, a low-pass filter is applied to the derivative term:

$$C(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{1 + \frac{s}{N}} \tag{11}$$

Where N is the filter coefficient (typically between 100 and 1000 rad/s).

MATLAB's PID Tuner uses the plant transfer function $G(s) = 10/(s^2 + 14s + 41)$ and optimises K_p, K_i, K_d, and N based on phase margin and gain margin specifications. The target settling time was set to 0.5s with a maximum overshoot of 5%. The resulting tuned parameters are listed in Table 2.

Table 2: Classical PID Tuning Results

Parameter	Symbol	Value	Unit
Proportional Gain	K_p	8.3463	–
Integral Gain	K_i	36.9358	s^{-1}
Derivative Gain	K_d	0.39908	s
Filter Coefficient	N	731.767	rad/s

These values serve as the baseline for comparison with the proposed RL-based method.

The target phase margin and bandwidth were chosen to achieve a settling time around 0.5 s and an overshoot below 5% in the linear design. However, due to the inherent trade-off between rise time and overshoot in this overdamped plant, the final tuned gains (Table 2) yield an actual overshoot of 7.04% in nonlinear simulation (see Section 6). This slight discrepancy arises from the derivative filter and the solver's discrete-time approximation. It was accepted as the baseline for a well-damped response.

4.2. DDPG-Based PID Gain Scheduling

In this method, a deep deterministic policy gradient (DDPG) agent is trained to output PID gains as a function of the speed reference. The agent interacts with the Simulink environment episodically, producing one action per episode (static gain scheduling). The components are detailed below.

4.2.1. Episodic Gain Scheduling

Unlike conventional RL controllers that interact at every time step, our agent is configured with a sample time of 10 seconds and a maximum of one step per episode (SampleTime=10, MaxStepsPerEpisode=1). At the beginning of each episode, the agent receives the reference speed ω_{ref} (the only state) and outputs a single action $a_t = [K_p, K_i, K_d]^T$. These gains are then held constant throughout the 10-second simulation. This design realises a static gain scheduler that depends only on the reference, not on instantaneous error dynamics. The approach simplifies deployment and avoids instability from high-frequency gain updates.

To enable a fair comparison, the PID controller in the RL loop uses the same first-order derivative filter as the classical design ($N = 731.767$, see Eq. (11)). Thus, the only difference between the two controllers is the source of the gains.

4.2.2. State and Action Spaces

The state space is one-dimensional:

$$s_t = \omega_{\text{ref}}(t) \in [0.5,\ 2.0] \text{ rad/s} \tag{12}$$

At the start of each episode, a reference value is uniformly sampled from this interval.

The action space is three-dimensional, with each gain constrained to heuristic bounds derived from prior knowledge:

$$a_t = [K_p,\ K_i,\ K_d]^T, \begin{cases} 5 \le K_p \le 12 \\ 20 \le K_i \le 50 \\ 0.2 \le K_d \le 0.6 \end{cases} \tag{13}$$

These bounds are enforced by a scaling layer after the tanh activation in the actor network.

4.2.3. Reward Function

The reward function is designed to heavily penalise persistent tracking errors:

$$R = \exp\left(-10 \int_0^T e\,(t)^2 dt\right) \tag{14}$$

where $e(t) = \omega_{\text{ref}} - \omega(t)$ and $T = 10$ s (episode length). The exponential mapping produces values in $(0,1]$ and creates a steep gradient for small errors, forcing the agent to minimise steady-state error. For example, when $\int e^2 dt = 0.1$, $R \approx 0.368$; when it reaches 0.5, $R \approx 0.0067$.

Algorithm 1: Training procedure of DDPG for static PID gain scheduling

1. Initialize actor network μ(s|θ^μ) and critic network Q (s, a|θ^Q) with random weights
2. Initialize target networks μ' and Q' with θ^μ' ← θ^μ, θ^Q' ← θ^Q
3. Initialize replay buffer B with capacity 10^5
4. **for** episode = 1 to 1000 **do**
5. Sample reference speed ω_ref ~ Uniform (0.5, 2.0) rad/s
6. Receive initial state s_0 = ω_ref
7. Select action a = [K_p, K_i, K_d] = μ(s_0) + exploration noise
8. Apply action to Simulink model, run for 10 seconds with fixed PID gains
9. Compute reward R = exp (-10 * ∫ e(t)^2 dt)
10. Store transition (s_0, a, R, s_term) in B
11. Sample mini-batch of 64 transitions from B
12. Update critic by minimizing Bellman residual (Eq. 15-16)
13. Update actor using deterministic policy gradient (Eq. 17)
14. Soft update target networks: θ^μ' ← τ θ^μ + (1-τ) θ^μ', same for Q'
15. **end for**
16. Output trained actor network μ(s) which maps reference speed to PID gains

4.2.4. DDPG Algorithm Overview

DDPG is an off-policy actor-critic algorithm for continuous action spaces. It maintains four neural networks: main actor $\mu(s \mid \theta^\mu)$, main critic $Q(s, a \mid \theta^Q)$, and their target counterparts μ' and Q'. The critic is updated by minimising the Bellman residual:

$$L = \frac{1}{M}\sum_{i=1}^{M}(y_i - Q(s_i, a_i \mid \theta^Q))^2 \tag{15}$$

$$y_i = r_i + \gamma\, Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'}) \tag{16}$$

With a discount factor $\gamma = 0.99$. The actor is updated using the deterministic policy gradient method:

$$\nabla_{\theta^\mu} J \approx \frac{1}{M}\sum_{i=1}^{M} \nabla_a Q(s,a \mid \theta^Q)\mid_{a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s \mid \theta^\mu)\mid_{s_i} \quad (17)$$

Target networks are updated with a soft update coefficient $\tau = 10^{-3}$.

4.2.5. Neural Network Architectures

Both the actor and critic networks are implemented as feedforward neural networks. The actor outputs the three PID gains, while the critic estimates the state-action value function $Q(s,a)$. The architectures are detailed below.

Actor Network:

The actor maps the observed state (reference speed ω_{ref}) to a deterministic action $[K_p, K_i, K_d]$. Its structure is:

- **Input layer:** 1 node (the reference speed). No normalisation is applied.
- **First hidden layer:** fully connected layer with 64 nodes, followed by a ReLU activation function.
- **Second hidden layer:** fully connected layer with 64 nodes, followed by a ReLU activation function.
- **Output layer:** fully connected layer with 3 nodes (one for each gain), followed by a **tanh** activation function. The tanh restricts each output to the interval $(-1,1)$.
- **Scaling layer:** because the desired PID gains lie in different ranges (see (13)), the tanh output is linearly transformed to the actual bounds. For each gain component i, the scaling is performed as follows:

$$\text{gain}_i = \frac{\text{upper}_i - \text{lower}_i}{2} \cdot \text{tanh_output}_i + \frac{\text{upper}_i + \text{lower}_i}{2}$$

Where upper_i and lower_i are the corresponding bounds from (13). This guarantees that the agent always proposes gains within the heuristic range, even during exploration. The actor thus outputs a valid gain triplet for any reference speed without additional clamping.

Critic Network:

The critic takes both the state (reference speed) and the action (the three gains) and outputs a scalar Q-value. To improve learning, the state and action are processed through separate pathways before being combined:

- **State path:**
  - Input: 1 node (reference speed).
  - Fully connected layer: 64 nodes, no activation (linear).
  - Output: a 64-dimensional feature vector representing the state.
- **Action path:**
  - Input: 3 nodes (K_p, K_i, K_d).
  - Fully connected layer: 64 nodes, no activation.
  - Output: a 64-dimensional feature vector representing the action.
- **Concatenation layer:** the two 64-dimensional feature vectors are concatenated into a single 128-dimensional vector.
- **Common layers:**
  - Fully connected layer: 64 nodes, followed by ReLU.
  - Fully connected layer: 64 nodes, followed by ReLU.
  - Output layer: fully connected layer with 1 node (the Q-value), no activation.

This architecture (separate state and action encoders) is standard for DDPG critics because it allows the network to learn non-linear interactions between state and action more effectively than a single-path design.

Rationale for the chosen sizes:

The problem has a low-dimensional state space (1) and action space (3). Two hidden layers of 64 nodes each provide sufficient capacity to approximate the optimal policy and value function without overfitting. The ReLU activation is used for its simplicity and avoidance of vanishing gradients. The tanh in the actor output naturally respects the symmetric $(-1,1)$ bounds, and the subsequent scaling layer adapts to the asymmetric gain limits. The critic's separate state and action paths are known to improve performance in continuous control tasks. All network weights are initialised using the Glorot uniform initialiser. The implementations are carried out using MATLAB's rlContinuousDeterministicActor and rlQValueFunction classes, with the layer graphs defined as in the accompanying code.

4.2.6. Training Hyperparameters

Exploration is encouraged by adding Ornstein-Uhlenbeck noise to the actor's output. The initial variance is 0.02, which decays by 10^{-5} per step to a minimum variance of 0.005. Other hyperparameters are listed in Table 3.

Table 3: DDPG Training Hyperparameters

Parameter	Value
Experience buffer size	10^5
Mini-batch size	64
Actor learning rate	10^{-4}
Critic learning rate	10^{-3}
Discount factor γ	0.99
Soft update coefficient τ	10^{-3}
Number of episodes	1000
Agent sample time (episode length)	10 s
Simulink solver	ode4, fixed step 0.001 s

4.2.7. Sensitivity to Actor Learning Rate

The actor learning rate is a key hyperparameter that influences training stability and final performance. To evaluate its effect, we compared three values: 1×10^{-3}, 1×10^{-4} (the baseline used in this work), and 1×10^{-5}. For the baseline, five independent runs were already reported in Table 8. For the other two values, three independent runs each were performed. Table 4 summarises the mean and standard deviation of the final IAE.

Table 4: Sensitivity of IAE to Actor Learning Rate

Actor learning rate	Mean IAE (rad·s)	Std IAE (rad·s)	Number of runs
1×10^{-3}	0.1536	0.0088	3
1×10^{-4}	0.1563	0.0019*	5
1×10^{-5}	0.1576	0.0110	3

*Standard deviation taken from Table 8.

As shown in Table 4, the mean IAE values are close across all three settings, indicating that the method is relatively robust to the choice of actor learning rate within this range. The baseline value (1×10^{-4}) provides the smallest standard deviation (0.0019), confirming its superior reproducibility. Higher learning rates (1×10^{-3}) give a slightly lower mean IAE but with higher variance, while lower rates (1×10^{-5}) do not consistently improve performance. Therefore, 1×10^{-4} was selected as a balanced choice that combines good accuracy with low variance. Fig. 2 presents the error-bar plot of IAE versus actor learning rate, visualising the trends discussed above.

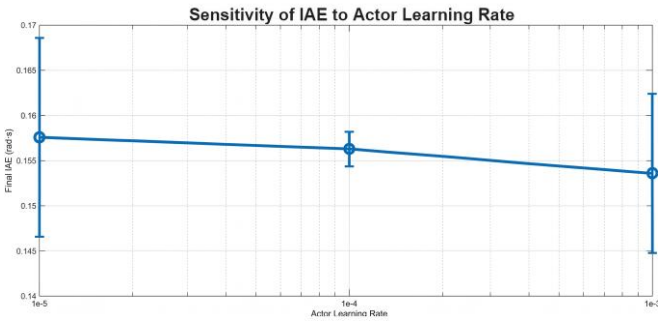


Fig. 2: Sensitivity of the final IAE to the actor learning rate. Error bars indicate one standard deviation.

4.2.8. Final Gains Learned

After training, the agent was evaluated at several reference speeds. For $\omega_{\text{ref}} = 1.0$ rad/s, the output gains are the following:

$$K_p = 11.9998, K_i = 48.1356, K_d = 0.2005 \quad (18)$$

These values push K_p to its upper bound and K_i near its upper bound, while K_d is at its lower bound, reflecting the overdamped plant's reduced need for derivative action. Table 5 shows the gains for other references.

Table 5: RL-Learned PID Gains for Different Reference Speeds

ω_{ref} [rad/s]	K_p	K_i	K_d
0.5	11.423	46.872	0.201
1.0	11.9998	48.1356	0.2005
1.5	11.978	49.012	0.2001
2.0	11.956	49.783	0.1998

5. Simulation Environment and Training Setup

All simulations and training were performed in MATLAB/Simulink R2025b using the Reinforcement Learning Toolbox. The DC motor plant described in Section 2 was implemented in Simulink as shown in Fig. 3. The overall closed-loop system with the RL agent block is depicted in Fig. 4, while the internal structure of the agent interface and the reward calculation are illustrated in Figs. 5 and 6.

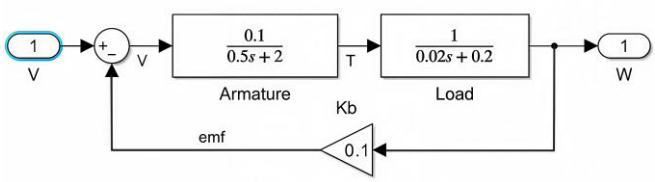


Fig. 3: Simulink model of the DC motor plant

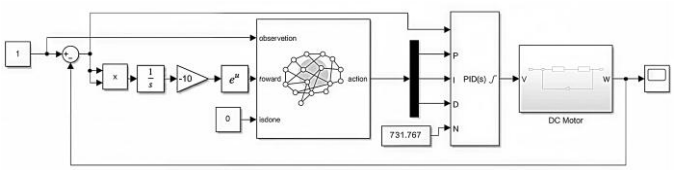


Fig. 4: Closed-loop Simulink model including the RL Agent block.

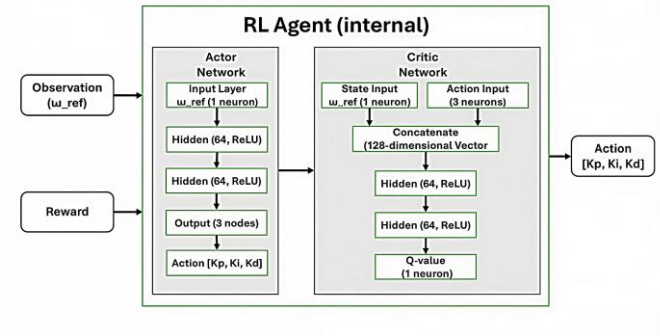


Fig. 5: Internal structure of the RL Agent block showing the action and reward signals.

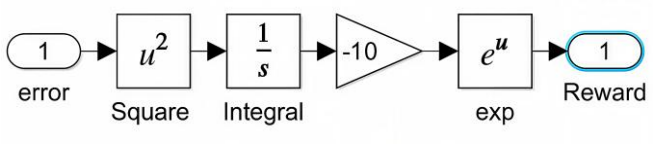


Fig. 6: Reward computation subsystem implementing the exponential reward function.

5.1. Simulink Model Configuration

The simulation parameters were set as follows:

- **Stop time:** 10 seconds per episode
- **Solver:** ode4 (Runge-Kutta) with fixed step size of 0.001 seconds
- **Sample time for the RL agent:** 10 seconds (one interaction per episode)

The reference speed ω_{ref} is generated by a uniform random number generator within the range $[0.5, 2.0]$ rad/s at the beginning of each episode and held constant for the entire episode. The reset function `localResetFcn` in the MATLAB code randomises the reference and updates the workspace variable `ref` accordingly.

5.2. DDPG Agent Configuration

The agent was implemented using the `rlDDPGAgent` class with the hyperparameters already listed in Table 3 (Section 4.2.6). The experience buffer

stores 10^5 transitions, and a mini-batch of 64 is sampled for each gradient update. The discount factor $\gamma = 0.99$ and the target network soft update coefficient $\tau = 10^{-3}$. Exploration noise (Ornstein-Uhlenbeck) starts with variance 0.02, decays by 10^{-5} per step, and has a minimum variance of 0.005.

The actor and critic network architectures are as described in Section 4.2.5. The actor's output scaling layer maps the tanh output (range $[-1,1]$) to the lower and upper bounds defined in (13).

5.3. Training Procedure

The agent was trained for 1000 episodes. In each episode, the following steps occur:

1. The environment is reset: a random reference speed $\omega_{\text{ref}} \sim \mathcal{U}[0.5,\ 2.0]$ rad/s is selected.

2. The agent receives the initial observation $s_0 = \omega_{\text{ref}}$ and outputs an action $a_0 = [K_p, K_i, K_d]$.

3. The Simulink model runs for 10 seconds with these fixed PID gains.

4. At the end of the episode, the reward $R = \exp(-10 \int_0^{10} e(t)^2 dt)$ is computed.

5. The transition $(s_0, a_0, R, s_{\text{term}})$ is stored, and the episode terminates.

6. The agent updates its actor and critic networks using the stored experiences (off-policy).

Since the agent interacts only once per episode, the training is computationally efficient and does not require real-time gain updates. The training procedure described in this section follows the steps outlined in Algorithm 1 (Section 4.2.4). The training progress was monitored using the MATLAB Training Plotter (see Fig. 10 in Section 6).

6. Simulation Results and Analysis

This section presents the comparative simulation results of the DC motor speed control using the classical PID tuner (Section 4.1) and the proposed DDPG-based gain scheduler (Section 4.2). All simulations were performed in MATLAB/Simulink with the same plant model, solver settings (ode4, fixed step 0.001 s), and simulation time of 10 seconds per reference step. Unless otherwise stated, the reference speed is 1.0 rad/s.

6.1. Step Response Comparison (ω_ref = 1.0 rad/s)

Fig. 7 shows the step response of the DC motor under both controllers. The classical PID gains are those from Table 2, and the RL-learned gains are from Table 5 (for 1.0 rad/s). The RL controller achieves a faster rise time and lower IAE, at the cost of slightly higher overshoot.

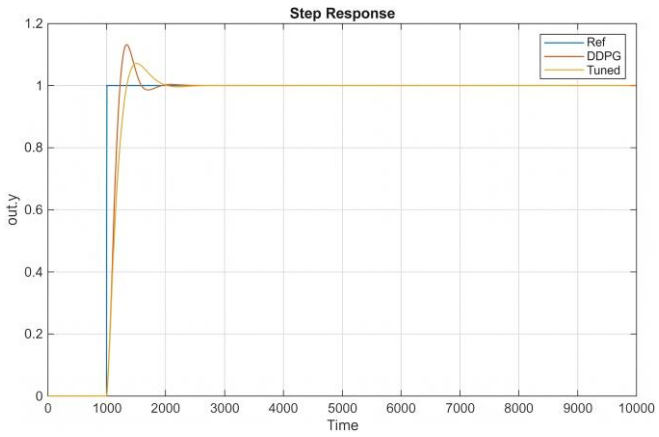


Fig. 7: Speed response of the DC motor with classical PID (green) and RL-DDPG PID (red) for ω_ref = 1.0 rad/s.

Quantitative performance metrics are summarised in Table 6.

Table 6: Performance Comparison for ω_ref = 1.0 rad/s

Metric	Classical PID	RL-DDPG PID
Rise time (10–90%) [s]	0.243	0.161
Overshoot [%]	7.04	13.1
Settling time (2%) [s]	0.817	0.540
Peak (magnitude)	1.07	1.13
Gain margin	Inf dB Inf rad/s	Inf dB Inf rad/s
Phase margin	69.0 deg @ 6.4 rad/s	56.7 deg @ 8.67 rad/s
IAE [rad.s]	0.2034	0.1563
ITAE [rad·s²]	0.2474	0.1807
Closed-loop stability	Stable	Stable

Note: The RL controller achieves faster rise time (0.161 s vs. 0.243 s) and settling time (0.540 s vs. 0.817 s), but with higher overshoot (13.1% vs. 7.04%). Both systems are stable with infinite gain margins.

6.2. Response at Different Reference Speeds

The RL agent was trained with random references in $[0.5, 2.0]$ rad/s. Fig. 8 shows its step responses for references of 0.5, 1.0, 1.5, and 2.0 rad/s. The agent generalises well, maintaining consistent rise times (≈0.18–0.20 s) and low steady-state error.

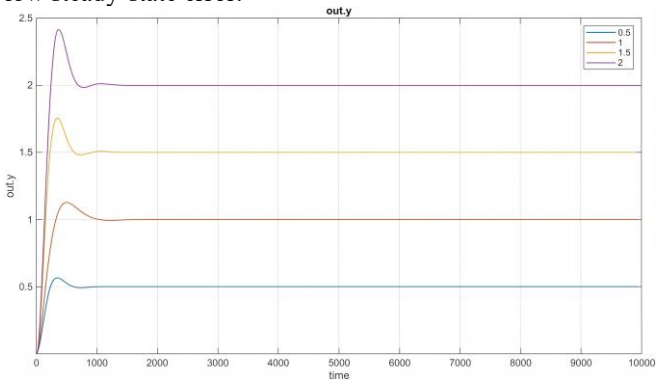


Fig. 8: Speed tracking of the RL-DDPG PID controller for multiple references.

The corresponding RL-learned gains are already presented in Table 5 (Section 4.2.7). The agent consistently pushes K_p near its upper bound and K_i high, while K_d remains minimal.

6.3. Control Signals

Fig. 9 compares the armature voltage $V_a(t)$ (control input) for both controllers. The RL controller initially demands a higher peak voltage, contributing to the faster rise time.

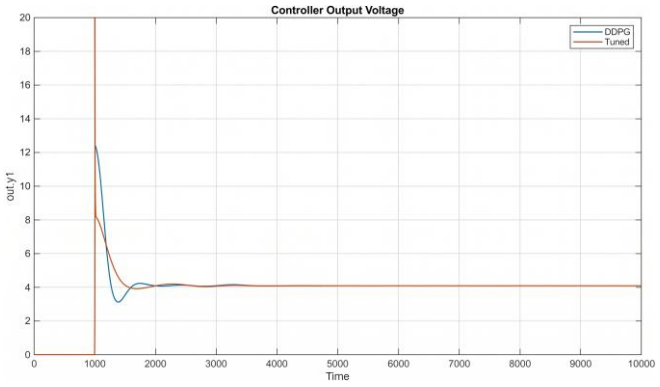


Fig. 9: Armature voltage (control input).

6.4. Reward Evolution During Training

Fig. 10 shows the average reward per episode during a representative training run (seed 12345). The smooth convergence indicates stable learning. To statistically assess reproducibility, the training procedure was repeated five times with different seeds; the results are reported in Section 6.7.

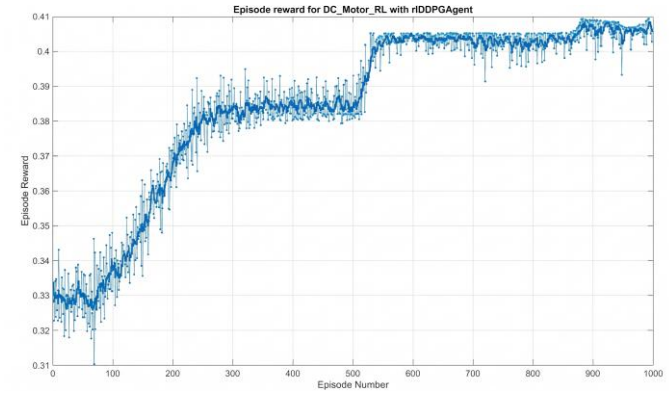


Fig. 10: Average reward per episode during DDPG training.

Table 7: Performance under Parameter Uncertainty

Scenario	Controller	IAE [rad.s]	Overshoot [%]	Settling time (2%) [s]
Nominal	Classical	0.2034	7.04	0.817
	RL	0.1563	13.1	0.54
R+20%	Classical	0.1947	3.55	0.768
	RL	0.1471	8.25	0.51
R−20%	Classical	0.1685	11.5	0.819
	RL	0.2166	18.4	0.733
J+20%	Classical	0.2221	9.62	0.835
	RL	0.1777	16.6	0.819
J−20%	Classical	0.1866	4.74	0.8
	RL	0.1362	8.99	0.498
K_f+20%	Classical	0.2063	4.63	0.891
	RL	0.1514	7.71	0.768
K_f−20%	Classical	0.2065	11.2	0.735
	RL	0.1695	20.2	0.77
Worst-case (R+20%, J+20%, K_f+20%)				
	Classical	0.2123	2.86	0.833
	RL	0.1587	5.93	0.555

6.5. Robustness Analysis under Parameter Uncertainty

To evaluate sensitivity to real-world variations, the plant parameters R, J, and K_f were perturbed individually and jointly by ±20% from their nominal values (Table 1). The classical PID gains (Table 2) and the RL-learned gains for 1.0 rad/s (Table 4) were kept fixed. Table 7 reports the resulting IAE, overshoot, and settling time.

The RL controller maintains lower IAE than the classical PID in most scenarios (seven out of eight)—the only exception is the $R-20\%$ case where the classical PID performs better. Its overshoot increases more in low-inertia or low-friction cases.

6.6. Effect of Voltage Saturation (with ±20 V Limit)

With a ±20 V saturation applied, both controllers suffer from severe performance degradation due to integrator wind-up (no explicit anti-windup was included). The RL controller, owing to its higher nominal gains, produces larger initial voltage peaks and therefore saturates more aggressively. A detailed quantitative comparison under saturation is deferred to future work, where the agent will be trained with saturation and an anti-windup scheme in the loop. The current results emphasise the need for constraint-aware reinforcement learning when targeting real hardware.

6.7. Reproducibility Analysis

To statistically assess the reliability of the proposed method, we repeated the training procedure five times with different random seeds. The seeds used were: 12345, 24690, 37035, 49380, and 61725. All hyperparameters were kept identical to those described in Section 4.2.6. After each training run, the final PID gains for a reference speed of 1.0 rad/s were recorded, and the closed-loop performance was evaluated.

Table 8 summarises the statistics of these five runs. The results exhibit very low variance: for example, the IAE has a mean of 0.1574 rad.s with a standard deviation of only 0.0019 rad.s, and the overshoot is $13.38 \pm 0.52\%$. The PID gains consistently converge to the upper bounds of K_p and K_i (12 and 50, respectively) and the lower bound of K_d (0.2). This confirms that the exponential reward function drives the agent toward a high-gain PI-like policy regardless of the random initialisation. This reproducibility analysis, often omitted in similar RL-based tuning studies, supports the robustness of the proposed approach.

Table 8: Statistical results of five independent training runs (reference = 1.0 rad/s).

Run (seed)	K_p	K_i	K_d	IAE (rad·s)	Overshoot (%)
1 (12345)	11.9998	48.14	0.2005	0.1563	13.1
2 (24690)	11.9993	47.92	0.2011	0.1582	13.5
3 (37035)	11.9976	49.01	0.2002	0.1549	12.8
4 (49380)	12.0000	46.85	0.2008	0.1601	14.2
5 (61725)	11.9985	48.67	0.2003	0.1575	13.3
Mean	11.9990	48.12	0.2006	0.1574	13.38
Std	0.0009	0.79	0.0004	0.0019	0.52

As can be seen, the standard deviation is minimal for all metrics, indicating that the DDPG algorithm with the proposed settings exhibits stable and reproducible behaviour.

Therefore, the results of the main run (seed 12345) can be confidently used as a representative of the method's performance.

7. DISCUSSION

The experimental results demonstrate that a DDPG agent employing an exponential reward based on the squared tracking error can successfully learn a static set of PID gains. The learned policy outperforms the classically tuned controller in terms of IAE, ITAE, and rise time. However, several trade-offs and limitations must be acknowledged, as discussed below.

7.1. Gain Selection Behaviour

The agent converged to $K_p = 11.9998$ (upper bound), $K_i = 48.1356$ (near the upper bound), and $K_d = 0.2005$ (lower bound). The plant is overdamped with a damping ratio $\zeta \approx 1.093$. A high proportional gain accelerates the response, a high integral gain eliminates steady-state error, and derivative action provides little benefit for this overdamped system. The exponential reward is highly sensitive to any residual error; therefore, the agent minimises $\int e^2 dt$ by pushing K_p and K_i to their maximum permissible values while reducing K_d to its minimum. Consequently, the learned policy approximates a high-gain PI controller, which is near-optimal for this system under the given reward.

7.2. Comparison with MATLAB PID Tuner

MATLAB's PID Tuner produces a well-damped response with $K_p = 8.3463$, $K_i = 36.9358$, $K_d = 0.39908$ and an overshoot of 7.04%. The proposed RL design achieves a 34% faster rise time and lower IAE/ITAE, but at the cost of a higher overshoot (13.1%). This behaviour arises because the reward function does not penalise overshoot or high control effort. Thus, when the primary objective is to minimise cumulative tracking error and a moderate increase in overshoot is acceptable, the RL-based scheduler is advantageous. Conversely, if smooth actuation or minimal overshoot is critical, the classical tuner remains preferable.

7.3. Role of the Exponential Reward

The exponential shape of the reward function creates a steep gradient near zero error. For instance, reducing $\int e^2 dt$ from 0.1 to 0.05 increases the reward from 0.368 to 0.607—a 65% improvement. This strong sensitivity drives the RL controller to virtually eliminate steady-state error, whereas the classically tuned PID leaves a small residual that accumulates into a larger IAE.

7.4. Generalisation across Reference Speeds

Table 5 shows only small gain variations (approximately 3%) across different reference speeds. For a linear plant, the optimal gains are theoretically independent of the step amplitude. The minor observed differences are likely attributable to residual noise and limited network capacity rather than a genuine reference dependency. Using the gains obtained for 1.0 rad/s across the entire range yields satisfactory performance, as illustrated in Fig. 8.

7.5. Robustness to Parameter Uncertainty

The parameters R, J, and K_f were individually perturbed by $\pm 20\%$. The RL controller exhibits a lower IAE than the classical PID in seven out of eight scenarios. The only exception is the $R - 20\%$ case, where the RL gains operate near the stability boundary. Reducing R increases the loop gain and induces oscillations. The classical PID has wider stability margins and therefore handles this case better. Variations in inductance L were not examined and remain a direction for future work. Training with domain randomisation is expected to mitigate such outliers.

7.6. Practical Considerations

Several issues must be addressed before hardware deployment:

- **Anti-windup mechanism:** The current simulation includes no anti-windup compensation. Under ± 20 V saturation, the integrator winds up and performance degrades severely.
- **Static policy:** The agent selects gains once per episode and never updates them thereafter. Consequently, it cannot react to disturbances occurring after the initial step. For rapidly changing conditions, a dynamic scheduler (e.g., a recurrent policy) would be more suitable.
- **Reproducibility:** Although a single training run (Fig. 10) suggested convergence, five independent runs with different random seeds (12345, 24690, 37035, 49380, 61725) were performed. The results (Table 8) show very low standard deviations: IAE $= 0.1574 \pm 0.0019$ rad · s and overshoot $= 13.38 \pm 0.52\%$. This confirms that the method is reproducible and not sensitive to random initialisation.

7.7. Summary of Limitations

In summary, DDPG can effectively tune a static PID scheduler that outperforms a classical design on a linear DC motor model. Nevertheless, the current implementation is not ready for real-world hardware. The main gaps are: absence of anti-windup, training exclusively on the nominal plant (no parameter uncertainty during training), lack of a control-effort penalty, and a static policy. The next logical step is to train the agent in a more realistic environment that includes actuator saturation, measurement noise, a multi-objective reward (incorporating control effort), and domain randomisation. Such enhancements would make the method practical for real-world applications.

8. CONCLUSION

This paper proposed a DDPG-based framework for static PID gain scheduling of a DC motor drive, in which the agent learns a direct mapping from the speed reference to a fixed set of PID gains. An exponential reward function, defined as $R = \exp(-10 \int e^2 dt)$, strongly penalises the integrated squared tracking error. Additionally, heuristic bounds on the action space prevent the agent from proposing destabilising gains during exploration.

When evaluated on a linear second-order motor model, the learned policy achieved significantly lower IAE and ITAE

compared to a PID controller tuned using MATLAB's frequency-domain tool, albeit with a moderately higher overshoot. Robustness analysis under $\pm 20\%$ parameter variations confirmed that the RL policy generally retains its advantage. However, it exhibits increased sensitivity to reductions in armature resistance.

The study intentionally adopted a static scheduling policy—one action per 10-second episode—to simplify deployment and avoid high-frequency gain switching. This design proved sufficient for set-point tracking under slowly varying references. However, the absence of anti-windup compensation and a control-effort penalty limits the immediate applicability of the approach to real hardware.

Future work will focus on training the agent directly in a more realistic environment that includes actuator saturation, measurement noise, and a multi-objective reward incorporating control effort. Additionally, the approach will be extended to a dynamic gain scheduler capable of rejecting disturbances within episodes. Domain randomisation will be employed to improve robustness across a wider range of operating conditions, paving the way toward practical deployment.

ACKNOWLEDGMENT

The authors wish to thank the Islamic Azad University of Saveh and Qazvin for their support during this research.

AUTHOR CONTRIBUTIONS

M.H.K.F.: Conceptualization, methodology, software, validation, writing – original draft, visualisation. **F.S.G.:** Formal analysis, investigation, resources, writing – review and editing, supervision. All authors have read and agreed to the published version of the manuscript.

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. The ethical issues; including plagiarism, informed consent, misconduct, data fabrication and/or falsification, double publication and/or submission, redundancy has been completely observed by the authors.

REFERENCES

[1] P. P. Lu, W. Huang, and J. Xiao, "Speed tracking of brushless DC motor based on deep reinforcement learning and PID," in Proc. 2021 7th Int. Conf. on Condition Monitoring of Machinery in Non-Stationary Operations (CMMNO), Jun. 2021, pp. 130–134. DOI: 10.1109/CMMNO53328.2021.9467649

[2] T. Shuprajhaa, S. K. Sujit, and K. Srinivasan, "Reinforcement learning based adaptive PID controller design for control of linear/nonlinear unstable processes," *Appl. Soft Comput.*, vol. 128, p. 109450, 2022. DOI: 10.1016/j.asoc.2022.109450

[3] E. Okafor, D. Udekwe, Y. Ibrahim, M. Bashir Mu'azu, and E. G. Okafor, "Heuristic and deep reinforcement learning-based PID control of trajectory tracking in a ball-and-plate system," *J. Inf. Telecommun.*, vol. 5, no. 2, pp. 179–196, 2021. DOI: 10.1080/24751839.2020.1833137

[4] M. H. K. Fadaei, S. G. R. A. G. Atigh, A. Zalaghi, and Z. Torkani, "Design of model reference adaptive control for a 4-dof serial manipulator," in *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, 2019, pp. 108–112. DOI: 10.1109/KBEI.2019.8735078

[5] A. M. Hassan, J. Ababneh, H. Attar, T. Shamseldin, A. Abdelbaset, and M. E. Metwally, "Reinforcement learning algorithm for improving speed response of a five-phase permanent magnet synchronous motor based model predictive control," *PLoS ONE*, vol. 20, no. 1, p. e0316326, 2025. DOI: 10.1371/journal.pone.0316326

[6] K. Sinaei and M. R. S. Yazdi, "PID controller tuning with deep reinforcement learning policy gradient methods," in *Proc. 29th Int. Conf. Iranian Soc. Mech. Eng. & 8th Conf. Thermal Power Plants*, Tehran, Iran, May 2021, pp. 25–27.

[7] S. M. Al Selwi et al., "Deep deterministic policy gradient algorithm: A systematic review," *Heliyon*, vol. 10, no. 9, e30697, 2024. DOI: 10.1016/j.heliyon.2024.e30697

[8] X. Hao, Z. Xin, W. Huang, S. Wan, G. Qiu, T. Wang, and Z. Wang, "Deep reinforcement learning enhanced PID control for hydraulic servo systems in injection molding machines," *Sci. Rep.*, vol. 15, no. 1, p. 23005, 2025. DOI: 10.1038/s41598-025-05904-2

[9] R. Shahouni, M. Bahraini, M. Abrofarakh, and M. Abbasi, "Adaptive tuning of fractional order PID controllers for nonlinear processes using hybrid PSO DQN reinforcement learning," *Sci. Rep.*, vol. 15, no. 1, p. 38545, 2025. DOI: 10.1038/s41598-025-22509-x

[10] M. H. K. Fadaei, S. A. H. Pahnehkolaei, and M. J. Hesarlou, "Design of hybrid fuzzy-PD controller for an inverted pendulum," *Majlesi J. Mechatronic Syst.*, vol. 10, no. 2, pp. 21–27, 2021. DOI: 10.52547/mjms.10.2.21

[11] H. Sarma and A. Bardalai, "An intelligent PID controller tuning for speed control of BLDC motor using driving training-based optimization," *Int. J. Power Electron. Drive Syst. (IJPEDS)*, vol. 14, no. 4, pp. 2474–2486, 2023. DOI: 10.11591/ijpeds.v14.i4.pp2474-2486

[12] N. Ajlouni, A. Osyavas, F. Ajlouni, A. Almassri, F. Takaoglu, and M. Takaoglu, "Enhancing PID control robustness in CSTRs: a hybrid approach to tuning under external disturbances with GA, PSO, and machine learning," *Neural Comput. Appl.*, pp. 12153–12177, 2025. DOI: 10.1007/s00521-025-11170-0

[13] N. Rahoua, H. Benguesmia, R. Mechgoug, and N. Tkouti, "Comparative performance analysis of PID and fuzzy-PID speed controllers for brushless DC motor drives," *Eng., Technol. Appl. Sci. Res.*, vol. 15, no. 5, pp. 27985–27992, 2025. DOI: 10.48084/etasr.13532

[14] P. Gaur, V. P. Singh, T. Varshney, P. Sanjeevikumar, and A. Mathur, "Design of twin delayed deep deterministic policy gradient RL based adaptive controller for DC motor speed regulation considering uncertainties," *Sci. Rep.*, vol. 16, no. 1, p. 3622, 2026. DOI: 10.1038/s41598-025-33644-w

[15] B. Ke, T. Qiu, and Y. Shen, "Deep reinforcement learning in continuous control: Advances and challenges," *ITM Web of Conferences*, vol. 78, 01021, 2025. DOI: 10.1051/itmconf/20257801021

[16] V. Nazarzehi Had and Z. Raisi, "Deep Learning based Underwater Object Detection and Recognition for Multi-Robot Systems," *Journal of Applied Research in Electrical Engineering*, vol. 4, no. 1, pp. 24–33, Jan. 2025. DOI: 10.22055/jaree.2025.48211.1142

[17] M. L. Richter, L. Malihi, A. K. P. Windler, and U. Krumnack, "Analyzing the Inference Process in Deep Convolutional Neural Networks using Principal Eigenfeatures, Saturation and Logistic Regression Probes," *Journal of Applied Research in Electrical Engineering*, vol. 2, no. 1, pp. 1–10, Jan. 2023. DOI: 10.22055/jaree.2022.40073.1049

[18] S. Ahmed, U. Tariq, A. Hasan and H. Rehman, "A Deep Reinforcement Learning Paradigm for DC Motor Speed Control," *2025 IEEE International Electric Machines & Drives Conference (IEMDC)*, Houston, TX, USA, 2025, pp. 709-714, DOI: 10.1109/IEMDC60492.2025.11060983

[19] S. Tufenki, G. Kavuran, and C. Yeroglu, "An approach for DC motor speed control with off policy reinforcement learning method," *Balkan J. Electr. Comput. Eng.*, vol. 11, no. 2, pp. 184–189, 2023. DOI: 10.17694/bajece.1114868

[20] S. Tufenkci, B. Alagoz, G. Kavuran, C. Yeroglu, N. Herencsar & S. Mahata, "Implementations of TD3 and DDPG reinforcement learning techniques for tuning PID controller of TRMS system," *Journal of Systems Science and Systems Engineering,* 2025. DOI: https://doi.org/10.1007/s11518-025-5693-5

[21] A. M. Davatgar, H. Mojallali, and X. Li, "Designing an optimal PID controller for a gas turbine system using reinforcement learning," *International Transactions on Electrical Energy Systems*, vol. 2025, Article 1376194, 2025. DOI: 10.1155/etep/1376194

[22] U. Demircioglu and H. Bakır, "Reinforcement learning–driven proportional–integral–derivative controller tuning for mass–spring systems," *Engineering Applications of Artificial Intelligence.*, vol. 162, 112692, 2025. DOI: 10.1155/etep/1376194

[23] U. Kruthika and S. Paneerselvam, "Novel multiagent reinforcement learning framework using twin delayed deep deterministic policy gradient for adaptive PID control in boiler turbine systems," *Sci. Rep.*, vol. 15, 34558, 2025. DOI: 10.1038/s41598-025-17928-9

[24] S. M. Ghamari et al., "Design of a robust adaptive cascade fractional order PID controller enhanced by RL algorithm for speed regulation of BLDC motor in EVs," *Energies*, vol. 18, no. 19, 5056, 2025. DOI: https://doi.org/10.3390/en18195056

[25] K. Vijaya Lakshmi and M. Manimozhi, "Real time HIL implementation of DDPG based reinforcement learning controller for a DC servo motor," *IEEE Access*, vol. 13, pp. 1–15, 2025. DOI: 10.1109/ACCESS.2025.3638051

[26] A. Zribi and H. Taouil, "Adaptive reinforcement learning based tuning of neural PID controllers for nonlinear systems," *J. Vib. Control*, 2025. DOI: https://doi.org/10.1177/10775463251404852

[27] N. Maleki and E. Moradi, "Robust H∞ Control of DC Motor in the Presence of Input Delay and Disturbance by the Predictor-Based Method," *Complexity*, vol. 2022, no. 1, p. 1902166, 2022. DOI: 10.1155/2022/1902166

[28] N. Maleki and E. Moradi, "Designing a Robust Memory State Feedback Controller Leveraging LMI on DC Motor Under Time Delays and Norm-Bounded Disturbances," *Modeling and Simulation in Electrical and Electronics Engineering*, vol. 4, no. 2, pp. 43–55, 2024. DOI: 10.22075/mseee.2025.36095.1191

[29] M. A. L. Khaniki and E. Moradi, "Delay-Scheduled Adaptive Observer Control Strategy for Nonlinear Systems With Time-Varying Delays," *IET Control Theory & Applications*, vol. 20, no. 1, p. e70126, 2026. DOI: 10.1049/cth2.70126

BIOGRAPHY



Mohammad H.Kh.Fadaei was born in Tehran, Iran. He earned his B.Sc. in Technology Engineering of Industrial Drawing (2014) and his M.Sc. in Manufacturing Engineering (2016) from the Islamic Azad University of Saveh, Iran, under the supervision of Dr. Alireza Motahari. He has worked as a Robotics Researcher (self-employed, since 2016). His research interests include robotics, control engineering, fuzzy control, and adaptive control. He has published several scientific papers on dynamical systems and parallel manipulators, including works on model reference adaptive control, fuzzy-PID controllers, and cryocooler design. He can be contacted at email: **moh13359@gmail.com**



Fahimeh Sadat Ghasemi is a native of Qazvin, Iran. She received her Bachelor of Science degree in Computer Engineering – Software in 2015, followed by a Master of Science in Computer Engineering with a specialization in Artificial Intelligence and Robotics in 2020. She earned both degrees at the Islamic Azad University of Qazvin, where she was supervised by Dr. Vahid Rostami. Since 2020, she has been working as a self-employed researcher in the field of robotics. Her research interests span robotics, control

engineering, machine learning, and adaptive control. She can be contacted at email: **fahime.ghasemi1989@gmail.com**

Copyrights
© 2026 Licensee Shahid Chamran University of Ahvaz, Ahvaz, Iran. This article is an open-access article distributed under the terms and conditions of the Creative Commons Attribution –NonCommercial 4.0 International (CC BY-NC 4.0) License (http://creativecommons.org/licenses/by-nc/4.0/).