Received: 13 September 2025 | Revised: 14 April 2026 | Accepted: 10 June 2026 | Published online: 17 July 2026

RESEARCH ARTICLE

Journal of Computational and Cognitive Engineering
2026, Vol. 00(00) 1–23
DOI: 10.47852/bonviewJCCE62027656

BON VIEW PUBLISHING

Manipulator Control Using DRL: Sim-to-Real Validation on a 3-DoF Arm

Shivkumar Sankaralingam[1], Nippun Kumaar Arulmani Angamuthu[1,2] and Suja Palaniswamy[1,*]

[1] *Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham-Bengaluru, India*

[2] *Department of Computer Science and Engineering (AI &ML), Dayananda Sagar University-Bengaluru, India*

Abstract: Deep reinforcement learning is emerging as a powerful alternative to traditional inverse kinematics for controlling robotic manipulators. By learning optimal actions through interaction with the environment, it enables adaptable and precise control in complex continuous workspaces, making it suitable for dynamic manipulator operations. This paper proposes an actor–critic deep reinforcement learning framework for manipulator control in a continuous workspace. Conventional inverse kinematics solutions can become computationally complex for high-degree-of-freedom manipulators or complex kinematic structures. Accurately generating joint angles by deriving configuration-specific equations therefore remains a persistent challenge. This study employs deep deterministic policy gradient and twin delayed deep deterministic policy gradient algorithms to directly predict joint angles for target-reaching tasks, eliminating reliance on conventional inverse kinematics formulations. A key contribution is a simple yet effective linear reward function that supports stable convergence in continuous action spaces. The proposed framework is implemented using ROS2 and Gazebo simulation and validated on a custom-built physical manipulator. The work addresses complex equation-based control through reinforcement learning and demonstrates proof of concept on a 3-degree-of-freedom manipulator. Experimental results show high task success rates of 95.77% for the deep deterministic policy gradient algorithm and 94.71% for the twin delayed deep deterministic policy gradient algorithm, with accurate end-effector positioning within 0.01 m. These results confirm the effectiveness of the proposed framework for accurate and reliable manipulator control with reduced model complexity and improved real-world applicability.

Keywords: deep reinforcement learning, DDPG, TD3, manipulator control, simulation-to-real transfer

1. Introduction

Reinforcement learning (RL) has emerged as a transformative paradigm in the field of robotics, providing robots with the capacity to acquire and adjust their behaviors through interactions with their surroundings. By combining a process of trial-and-error exploration with the utilization of rewards or penalties, RL algorithms empower robots to enhance their decision-making capabilities [1]. This methodology has demonstrated significant promise across various domains within robotics, encompassing navigation, manipulation, grasping, and task execution. Traditional control methods such as Proportional–Integral–Derivative (PID) control [2], Computed Torque (CT) control [3], Sliding Mode (SM) control [4], fuzzy control [5], and optimization-based approaches [6, 7] have historically been used for robotic manipulator control. These techniques perform well in structured environments where accurate system models are available. However, as highlighted by Ajwad et al. [8], their performance often degrades in complex or dynamic environments due to strong dependence on precise mathematical modeling and manual parameter tuning. This limitation has motivated the exploration of learning-based approaches for manipulator control.

While machine learning (ML) techniques are effective in pattern recognition and model estimation, and deep learning (DL) architectures can process high-dimensional sensor data [9, 10], both approaches typically rely on large, pre-collected datasets. As highlighted in References [11, 12], they often lack the adaptability required in dynamic or real-time environments, struggle with generalization to unseen scenarios, and are sensitive to noise and model variations, making them less suitable for direct low-level control in robotics. Motivated by these limitations, this work explores RL as an alternative framework for manipulator control, enabling policies to be learned directly through interaction with the environment without requiring explicit system models. By using RL, the manipulator can learn from its experience by interacting with the environment, making it more adaptable and generalizable to different scenarios. It also reduces the collection of large amounts of data and could promote real-time decision-making, which would enable the manipulator to provide quick responses with reliability.

Inverse kinematics (IK) methods require accurate analytical or numerical models of the robot's kinematic structure to compute joint configurations corresponding to a desired end-effector position. While effective for well-defined manipulators, these approaches can become increasingly complex for robots with

***Corresponding author:** Suja Palaniswamy, Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham-Bengaluru, India. Email: p_suja@blr.amrita.edu

higher degrees of freedom, redundant configurations, or intricate link geometries. In addition, IK solutions are often configuration-specific and sensitive to singularities within the workspace. As discussed in References [13–15], these characteristics can limit adaptability in environments involving modeling uncertainties, sensor noise, or dynamic disturbances, motivating the exploration of learning-based control strategies.

Deep reinforcement learning (DRL) provides an alternative paradigm for manipulator control by enabling agents to learn control policies directly through interaction with the environment rather than relying on explicit analytical models [16]. This capability is particularly advantageous for robotic manipulators, which often exhibit nonlinear dynamics, complex joint interactions, and varying operating conditions. Unlike traditional model-based approaches, DRL can learn continuous control policies that map observed states directly to joint actions, allowing the system to adapt to uncertainties and variations in the environment. Furthermore, actor–critic methods such as deep deterministic policy gradient (DDPG) and twin delayed deep deterministic policy gradient (TD3) are specifically designed for continuous action spaces, making them well suited for robotic control tasks where joint angles must be adjusted smoothly and precisely.

Despite the growing success of DRL in robotic control, its application to manipulator systems remains constrained by several practical and methodological challenges. First, many existing DRL-based approaches for manipulator control are designed for discretized or partial continuous workspaces [17, 18], limiting their applicability in real-world scenarios that demand fine-grained, continuous control across the entire operational space. Second, several prior works employ complex reward structures, including exponential [19], trigonometric [20, 21], or multi-metric formulations [22], which, while effective in certain scenarios, can increase training complexity, slow down convergence, and make reward shaping more sensitive and less generalizable. Third, although DRL has demonstrated promise in simulation environments, relatively few studies provide thorough hardware-level validation, especially for continuous joint-space control of manipulators [23–25]. This limits the understanding of sim-to-real generalizability—an essential factor for deploying these systems in practical applications. To address these gaps, this work proposes a lightweight, actor–critic DRL-based framework for controlling a 3-degree-of-freedom (3-DoF) robotic manipulator in a continuous three-dimensional workspace. The proposed approach makes the following key contributions:

1) An alternative to traditional IK with DRL-based joint angle prediction, enabling real-time adaptability without relying on complex kinematic modeling
2) Introduction of a simple, piecewise-linear reward function that promotes stable learning behavior while reducing reward computational complexity.
3) Deployment of compact multilayer perceptron (MLP)-based actor and critic networks within both the DDPG and TD3 frameworks, optimized through systematic hyperparameter tuning.
4) Comprehensive evaluation of the proposed approach in both simulated (Gazebo, ROS2) and real-world environments using a custom-built physical manipulator.
5) Achievement of high end-effector accuracy within 0.01 m and success rates exceeding 95%, demonstrating robust performance and effective sim-to-real transfer.
6) The findings underscore the practicality of DRL in manipulator control tasks and provide a model-free control alternative that avoids explicit IK computation especially in scenarios involving dynamic and unstructured environments.

The contributions of this work can be categorized into two aspects: application-level contributions and methodological contributions. From an application perspective, the study demonstrates the deployment of actor–critic DRL algorithms for continuous control of a 3-DoF robotic manipulator with successful sim-to-real transfer using a ROS2 and Gazebo framework. From a methodological perspective, the work investigates a simplified piecewise-linear reward formulation designed to improve training stability while reducing reward design complexity compared with more computationally complex nonlinear reward structures commonly used in RL-based manipulator control.

The structure of this paper is organized as follows. Section 2 reviews the existing literature on the topic, covering areas such as the progression of manipulator control techniques, the application of ML and DL methods, and the rise of RL in manipulator control. It also examines the formulation of reward functions across different use cases and scenarios. Section 3 introduces the RL framework for the problem statement, detailing the model algorithms employed. Section 4 explains the system architecture, encompassing the manipulator design and model structures. Section 5 explores the kinematic analysis of the 3-DoF manipulator. Section 6 presents the evaluation metrics used, hyperparameter tuning processes, and training and testing outcomes, including both simulation and hardware results. Section 7 discusses the key contribution of the work. Finally, Section 8 provides the paper's conclusions and suggests future research directions.

2. Related Works

To understand the research area in manipulator control, a historical overview of control theory applied to robotic manipulators, focusing on the foundational principles of robot control in its early stages, as analyzed by Spong [1], was studied. Ajwad et al. [8] reviewed advanced control techniques for robotic manipulators, focusing on intelligent PID, robust control, and adaptive approaches. They highlighted the future trends and open-source simulators. In Reference [13], the modeling and control techniques for various robotic manipulators, addressing vibration issues and evaluating different models and methods, were reviewed. It also explored various optimization algorithms for controlling manipulator position, velocity, and vibration. In Reference [2], a PID controller was used to control the manipulator that reduces overshoot and conserves electrical energy by dynamically adjusting confidence weights using real-time error and delay. The CT algorithm has been used in Reference [3] to control the manipulator integrated with online least squares support vector regression to improve performance and build an adaptive architecture without prior system model information. Azeez et al. [4] proposed an SM control for a 3-link manipulator with a multi-elite artificial bee colony algorithm for optimal trajectory tracking and noise cancelation. Wang et al. [5] introduced a position controller using adaptive fuzzy techniques to enhance robotic manipulator performance in uncertain environments. The fuzzy logic was also implemented for a four-axis manipulator in Reference [26]. In Reference [6], a self-tuning Particle Swarm Optimization (PSO)-based fuzzy PID controller was proposed to enhance the robustness and accuracy of manipulators affected by disturbances and noise. In Reference [7], a hybrid algorithm combining the Spiral Dynamic Algorithm and Bacterial Foraging Algorithm was proposed for a flexible manipulator to improve

exploration, convergence speed, and fitness accuracy. In Reference [27], alternative methods were also compared such as adaptive fuzzy and hybrid fuzzy control algorithms for manipulator control through simulations, using performance indices like steady-state error and maximum error. However, the trend for controlling robotic arms then shifted toward using ML and DL techniques.

The use of neural networks has become widespread. Liu et al. [9] explored artificial neural networks (ANNs) for modeling and controlling robotic manipulators, highlighting their effectiveness in handling uncertain dynamics and complex structures. It discusses ANN-based methods and control techniques. Rawat et al. [11] reviewed intelligent control techniques for robotic manipulators, including AI-driven methods like neural networks and metaheuristic algorithms. It highlights their effectiveness in addressing complex dynamics and enhancing real-world applications. Jin et al. [10] reviewed the increasing use of neural networks for controlling robot manipulators in scientific research and engineering applications, highlighting theoretical foundations, recent progress, and potential directions for practical implementation. Neural networks offer high-speed parallel processing and promise labor-saving and accuracy-enhancing solutions in manipulator control systems. In Reference [28], various ML algorithms in the context of robotic system design of manipulators were discussed. Liu et al. [12] presented a neural network-based adaptive control method for robotic manipulators, highlighting its importance in tasks like logistics sorting and fruit picking. The proposed method uses an Radial Basis Function (RBF) neural network and Lyapunov function for system stability, incorporating back-stepping control to achieve adaptive and stable closed-loop system performance. Li and Li [29] introduced a sparse auto-encoder controller for manipulator kinematic control, utilizing model-based weights to improve robustness against noise and parameter uncertainty. The controller incorporates input saturation and demonstrates effective path tracking through extensive simulations. Ren and Ben-Tzvi [14] proposed using modified Generative Adversarial Networks to efficiently learn inverse kinematics and dynamics of robotic manipulators with limited data. In Reference [30], SCARA robots equipped with Convolutional Neural Networks (CNNs) were proposed that effectively automate waste sorting, enhancing recycling accuracy and efficiency. Though ML and DL were widely used for manipulator control, they faced drawbacks such as high data requirements, significant computational resource needs, complexity and interpretability issues, risks of overfitting and poor generalization, and challenges in achieving real-time performance [10].

RL has become the current trend and technique for robotic solutions across various domains [31]. It is extensively used in mobile robots [32], manipulators [33], autonomous vehicles [34], robotic surgery [35], and industrial automation. This approach enables robots to learn optimal behaviors through interaction with their environment, making it versatile and powerful for diverse applications. Han et al. [17] reviewed recent advances in DRL algorithms for robotic manipulation, covering methods, challenges, and solutions.

Li et al. [19] introduced a motion planning framework for redundant robot manipulators using DRL to optimize both path length and energy consumption. Luipers et al. [22] studied the computational challenges of robotic learning with reinforcement learning by combining model-based deep RL for planning and IK for execution. Joshi et al. [20] introduced a DRL approach for robotic grasping with visio-motor feedback, reducing the need for hand-designed features and improving grasping accuracy. In Reference [36], a DRL framework for prioritizing tasks in redundant robots was introduced, ensuring critical tasks are smoothly executed using the Deep Q-Network (DQN) algorithm. Zheng et al. [37] addressed inefficient convergence in manipulator trajectory planning in dynamic environments. It introduced an action selection technique and a reward function, significantly enhancing efficiency in DRL. Honerkamp et al. [23] presented a DRL approach to enable mobile robots to perform complex manipulation tasks. It leverages kinematic feasibility as a dense reward signal and performs across various robot platforms in both simulated and real-world experiments. In Reference [15], a DRL approach for solving the complex IK problem of a 7-DoF robotic manipulator using DQN was proposed. In Reference [18], a DRL learning-based motion planning algorithm for manipulators to enhance adaptability and autonomy was introduced. Luipers et al. [22] presented the computational challenges of robotic learning through RL by combining model-based DRL for planning and IK for execution using a simulated Franka Emika Panda robot. In Reference [38], a closed-loop pose/force controller for a soft manipulator using DRL, promising sim-to-real transfer performance, was developed. The controller leverages domain randomization and reward engineering to handle complex interactions, revealing emergent mechanical intelligence in the robot. Carlos et al. [24] developed a simulated 7-DoF robotic manipulator in CoppeliaSim, controlled using an advantage actor–critic DRL agent. He et al. [39] developed and validated an actor–critic reinforcement learning control strategy for a flexible two-link manipulator to achieve vibration suppression and trajectory tracking. Unlike existing works that focus primarily on discrete or partially continuous workspaces, or continuous workspaces confined to small regions, our method successfully handles a continuous workspace spanning the entire manipulator region. This not only provides a more comprehensive solution but also ensures better accuracy and applicability to real-world robotic systems.

In reinforcement learning, the reward function is crucial as it guides the agent's behavior by providing feedback on the desirability of actions, helping it learn the most effective strategies. While complex reward functions such as those involving exponential [19], trigonometric [20, 25], or multi-metric objectives [22] can improve learning specificity in some tasks, they often introduce challenges in convergence stability and interpretability. Designing and tuning reward functions can be challenging in reinforcement learning applications. Our approach addresses this by proposing a piecewise-linear reward structure that simplifies training and enhances convergence, especially in continuous action spaces.

In contrast to the majority of existing works that focus on discrete or small-scale continuous workspaces, we propose a DRL-based control strategy for a 3-DoF robotic manipulator operating within a fully continuous 3D hemisphere-shaped workspace, as illustrated in Figure 1. Our framework serves as a viable alternative to IK with actor–critic DRL algorithms (DDPG and TD3), allowing for direct joint angle prediction without reliance on explicit kinematic models. To promote stability and reduce complexity, we introduce a simple, piecewise-linear reward function and utilize a compact MLP-based policy architecture. These design choices emphasize ease of implementation and training efficiency, addressing limitations in reward shaping and sim-to-real transfer noted in prior literature [19, 22, 38]. The summary of the survey is provided in Table 1. This table summarizes existing DRL-based approaches for manipulator control and highlights their main advantages and limitations. In contrast to prior methods that rely on IK, complex reward engineering, large visual datasets, or high computational

Table 1
Summary of DRL methods for manipulator control

References	DRL method	Application	Advantages	Limitations
[15]	Deep Q-Network (DQN)	Solving inverse kinematics for 7-DoF manipulators	Eliminates explicit kinematic modeling	Limited scalability for continuous control
[18]	DRL-based motion planning	Autonomous manipulator motion planning	Improved adaptability and autonomy	Requires extensive training data
[19]	Deep reinforcement learning for motion planning	Path planning for redundant manipulators optimizing path length and energy	Handles multi-objective optimization, adaptable to complex environments	Complex reward design and longer training time
[20]	Deep RL with visiomotor feedback	Robotic grasping tasks	Eliminates handcrafted features, improves grasp accuracy	Requires large visual datasets
[22]	Model-based deep RL + inverse kinematics	Planning and execution for Franka Emika Panda robot	Improves planning efficiency by combining model knowledge and learning	High computational requirements
[23]	DRL manipulation framework	Complex manipulation tasks with mobile robots	Works across different platforms and environments	Training complexity and computational load
[36]	Deep Q-Network (DQN)	Task prioritization in redundant robots	Efficient decision-making for sequential tasks	Discrete action space limits control precision
[37]	Improved DRL with optimized action selection	Trajectory planning in dynamic environments	Faster convergence and improved efficiency	Sensitive to reward design
[38]	Deep RL pose/force control	Closed-loop control of soft manipulators	Good sim-to-real transfer using domain randomization	Complex reward engineering
[39]	Actor–critic reinforcement learning	Vibration suppression and trajectory tracking in flexible manipulators	Effective control in continuous workspace	Training stability challenges

requirements, the proposed method provides a model-free DRL-based alternative for direct joint angle prediction. The key strength of this work lies in the use of a simple piecewise-linear reward function with compact MLP-based actor–critic networks under DDPG and TD3 frameworks. The proposed approach achieves stable learning, high end-effector accuracy within 0.01 m, and success rates above 95% in both Gazebo/ROS2 simulation and real-world experiments, demonstrating practical applicability and effective sim-to-real transfer.

3. Reinforcement Learning Formulation

Our proposed framework features a continuous workspace defined as a hemisphere with a diameter of 0.4 m, offering a total volume of approximately 0.0168 m^3, as shown in Figure 1. This design enables the manipulator to access all regions within the hemisphere seamlessly. The framework ensures high precision in representing continuous values, with a granularity of up to two decimal places, facilitating fine-grained analysis and control. This continuous design not only maximizes the manipulator's operational reach but also allows for systematic exploration of the entire workspace.

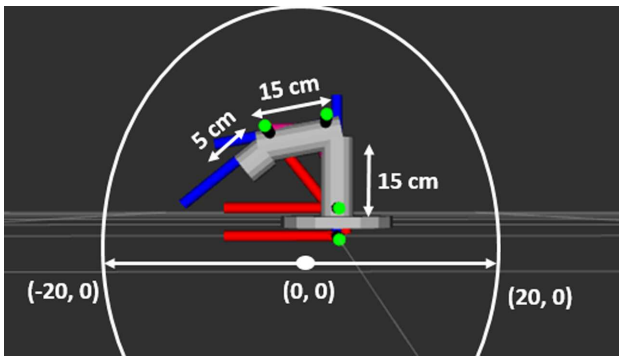
3.1. Environment

The environment is a 3-DoF robotic arm or manipulator. Figure 1 illustrates the 3D model of the manipulator in the RViz visualization tool.

3.2. Agent

The agent is the developed robotic controller, which serves as the software or brain behind controlling the manipulator. It is responsible for learning the environment.

Figure 1
3-DoF robotic manipulator as environment



3.3. State

The state refers to the position of the end-effector within a 3D workspace. It contains both the home and the destination coordinates. It is represented as shown in Equation (1).

$$s = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix} \in \mathbb{R} \tag{1}$$

where (x_1, y_1, z_1) are the home coordinates, (x_2, y_2, z_2) are the destination coordinates, and each element $x_1, y_1, z_1, x_2, y_2, z_2 \in \mathbb{R}$ in the coordinate space.

3.4. Actions

The actions correspond to the joint angles of the manipulator. The joint space is defined by $(\theta_1, \theta_2, \theta_3)$, with each angle ranging from $\frac{-\pi}{2}$ to $\frac{\pi}{2}$. This is represented in Equation (2).

$$a = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \in [\frac{-\pi}{2}, \frac{\pi}{2}] \tag{2}$$

where $\theta_1, \theta_2, \theta_3$ are the joint angles of the manipulator and each angle $\theta_i \in [\frac{-\pi}{2}, \frac{\pi}{2}]$.

3.5. Rewards

The reward function R is formulated to guide decision-making based on the Euclidean distance D between target coordinates and end-effector coordinates of the robotic manipulator. Specifically, R follows a piecewise structure: if D falls within the narrow interval [-0.01 m, 0.01 m], the reward is computed as $\frac{k}{D+\tau}$, where $k = 500$ and $\tau = 0.1$. This formulation ensures a high reward when the end-effector is near the target, inversely scaling with the distance D. Conversely, outside this range, R adopts a linear penalty proportional to D but negatively scaled by k. Thus, the function optimizes performance by incentivizing proximity to the target while penalizing deviations, leveraging a nuanced reward strategy. This piecewise and linear function adjusts the reward based on proximity to the target within a specific tolerance, with k scaling the reward magnitude inversely proportional to D when within range and directly proportional when outside it. The simple reward function is provided in Equation (3).

$$R = \begin{cases} \frac{k}{D+\tau} \; if \; D \in [-0.01, \; 0.01] \\ -k \times D \; otherwise \end{cases} \tag{3}$$

The symbol $\times$ denotes standard scalar multiplication between the constant k and the distance D. The positive reward parameter k was set to 500, calculated using Equation (4). A scalar value of 100 was chosen for simplicity, although any positive constant could be used, as this factor primarily serves a scaling purpose. Additionally, for the positive reward, a constant $\tau = 0.1$ was incorporated to ensure nonzero divisibility.

$$k = \frac{100}{Workspace \; Radius} \tag{4}$$

Unlike several prior works that use complex or nonlinear reward functions, such as exponential [17], trigonometric [22, 25], or multi-objective formulations [19], our approach adopts a simple piecewise-linear reward structure.

To demonstrate that the proposed reward function performs better than nonlinear functions, the following reward functions were designed to compare with our novel approach, as shown in Equations (5) and (6).

$$R = \begin{cases} -k \times (1 - \tanh|D|) \; if \; D \in [-0.01, \; 0.01] \\ -k \times \tanh D \; otherwise \end{cases} \tag{5}$$

$$R = \begin{cases} \frac{k}{D^2+\tau} \; if \; D \in [-0.01, \; 0.01] \\ -k \times D^2 \; otherwise \end{cases} \tag{6}$$

In Equation (5), the reward is based on the hyperbolic tangent function, providing a smooth, bounded response as the error changes. In Equation (6), the reward uses an inverse quadratic form near the target and a quadratic penalty when the error moves outside the tolerance range. These functions are used to evaluate the effectiveness of the proposed reward design against nonlinear alternatives.

The agent was trained with the reward functions for 10,000 episodes, and the resulting success rates are shown in Figure 2. All reward functions were evaluated under identical training conditions, including the same network architecture, hyperparameters, and environment configuration, ensuring a fair comparison of learning performance. The proposed linear reward function achieves a higher success rate and demonstrates faster learning compared to the quadratic and tanh-based reward functions. While the quadratic and tanh functions improve gradually, their final success rates remain lower than that of the proposed approach, indicating the effectiveness of the linear reward design.

The cumulative rewards obtained using the different reward functions during training are shown in Figure 3. The proposed linear reward function shows a steady improvement in rewards as training progresses. In comparison, the quadratic and tanh-based reward functions exhibit smaller reward improvements and greater variability. This indicates that the proposed reward design provides more stable learning and better reward accumulation over time.

While the choice of a piecewise-linear reward function has not been extensively benchmarked in literature for manipulator control tasks, it is empirically validated in this study to yield stable convergence and a high success rate exceeding 95%. This simplification not only reduces reward computational complexity but

Figure 2
Reward function performance comparison

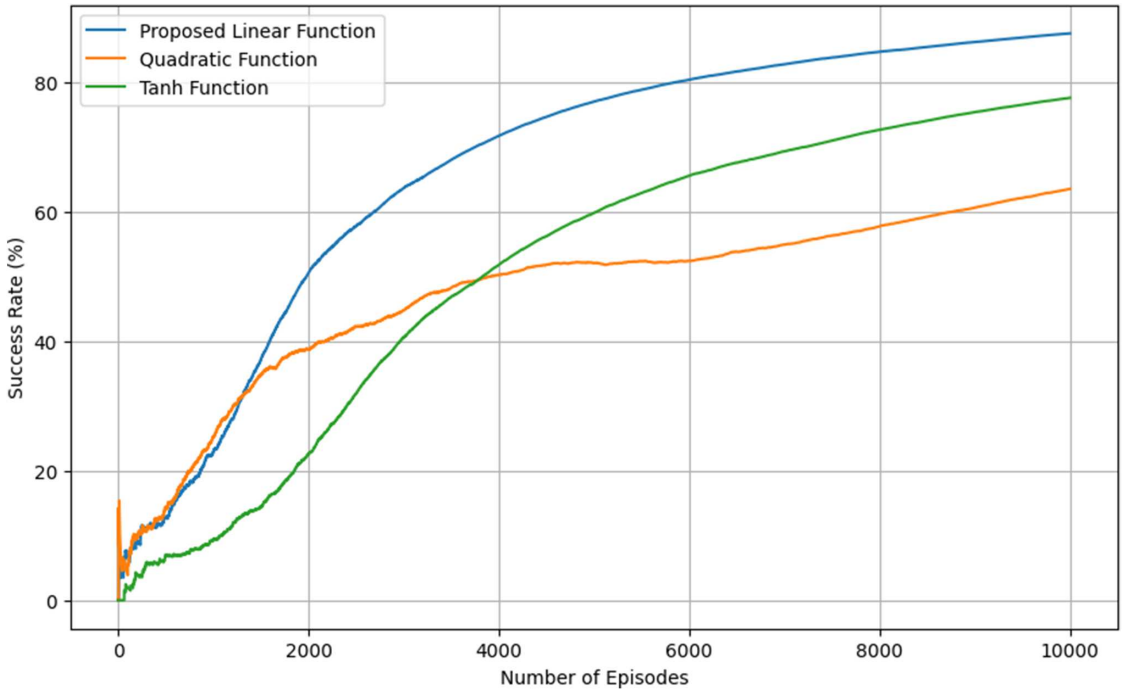
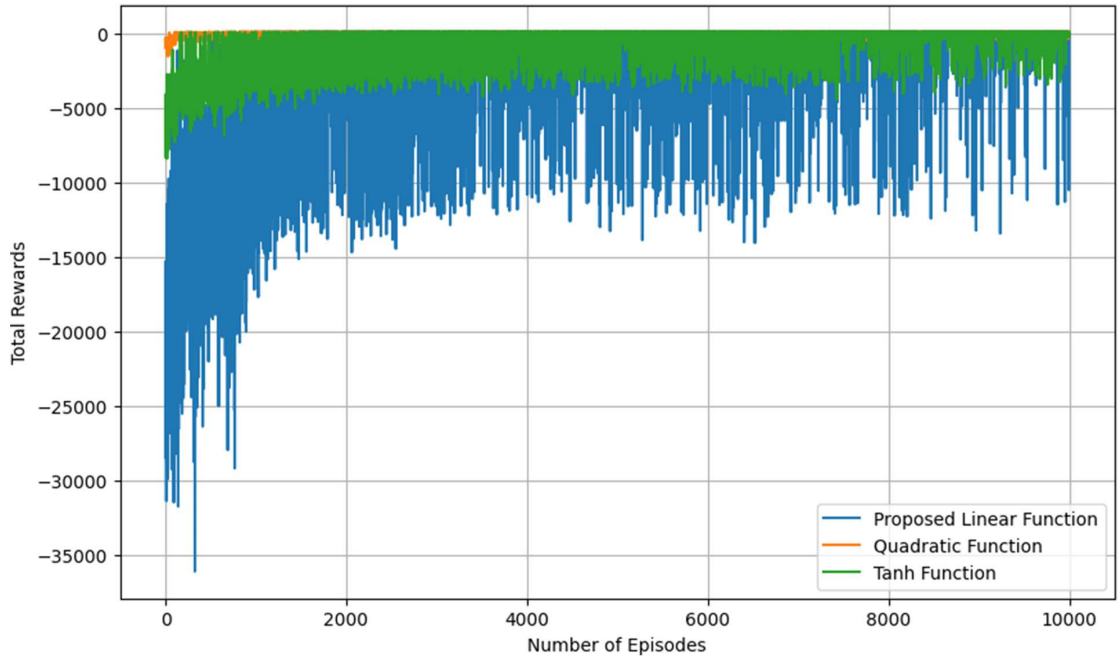


Figure 3
Accumulated reward performance comparison



also facilitates more interpretable learning dynamics, contributing to the practical applicability of the proposed DRL framework.

Compared with the nonlinear reward formulations evaluated in this study, the proposed linear reward function demonstrated faster convergence and more stable reward accumulation during training. As observed in Figures 2 and 3, the proposed formulation achieved stable convergence in approximately 31% fewer training episodes than the quadratic and tanh-based reward functions. In addition, the linear reward structure avoids repeated nonlinear operations associated with trigonometric or exponential reward calculations, thereby reducing computational overhead during reward evaluation.

The structure of the reward function plays a crucial role in determining the stability and convergence behavior of reinforcement learning algorithms. In continuous control problems such as robotic manipulator motion, overly complex reward formulations may introduce steep gradients or highly nonlinear reward variations that can destabilize policy updates. In contrast, the proposed

piecewise-linear reward provides a smoother and more predictable learning signal for the actor–critic networks. This enables the critic network to estimate value functions more reliably, which in turn stabilizes actor updates and reduces oscillations during training. Consequently, the proposed reward formulation improves both convergence efficiency and learning stability.

3.6. Deep Deterministic Policy Gradient

DDPG is an actor–critic reinforcement learning algorithm designed for continuous action spaces. It combines deterministic policy gradients with deep neural networks to learn control policies directly from high-dimensional state representations. DDPG has been widely applied to robotic control tasks due to its ability to handle continuous joint actions efficiently [21].

3.7. Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed DDPG (TD3) extends DDPG by introducing mechanisms to mitigate overestimation bias and improve training stability. These include clipped double Q-learning, delayed policy updates, and target policy smoothing [40]. TD3 has demonstrated improved robustness compared to DDPG in several continuous control benchmarks [41].

4. System Architecture

The system architecture outlines the overall framework that integrates the manipulator with the control algorithms. This involves both the physical and software components necessary for effective operation and learning. In this section, the manipulator design for simulation and hardware with a model architecture developed for DDPG and TD3 is discussed.

4.1. Manipulator design

The manipulator design, illustrated in Figure 1 using the RViz 3D visualization tool, features a 3-DoF configuration. This design includes three distinct segments: the torso, the upper arm, and the lower arm. Specifically, the torso and upper arm each have a length of 15 cm, while the lower arm measures 5 cm. The workspace of the manipulator is designed as a hemisphere with a radius of 20 cm, allowing the arm to operate within this spatial volume.

4.2. Hardware design

To assess the real-time performance of the trained model, a physical 3-DoF manipulator was constructed and integrated with hardware components. The manipulator's joint angles are sent by the microcontroller to the manipulator, interfacing with a Python script that executes the trained model's predictions. This control system is shown in Figure 4.

4.3. Denavit–Hartenberg parameters

The Denavit–Hartenberg parameters constitute a standardized set of four parameters crucial for defining the geometric and kinematic properties of robotic manipulator joints. The labeled diagram of a 3-DoF manipulator is shown in Figure 5; it illustrates a robotic arm capable of moving in a plane by adjusting the angles (θ_1, θ_2, θ_3) at each joint. By manipulating these angles,

Figure 4
Control system block diagram for hardware

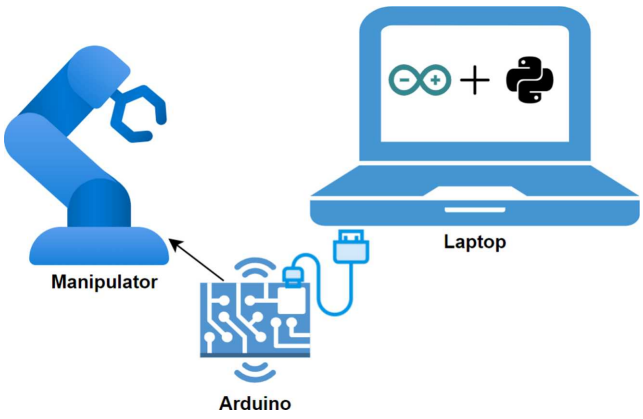
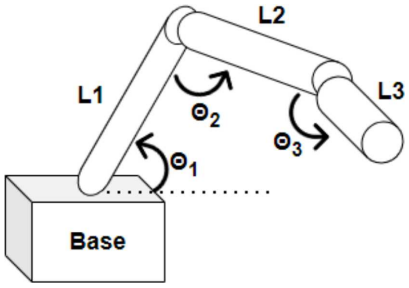


Figure 5
3-DoF manipulator diagram



the arm can reach different positions and orientations within its workspace, which in this case is a hemisphere. Each link in the arm is connected to the previous one by a rotational joint, allowing movement in a specific plane. Based on the diagram, the calculated values of the Denavit–Hartenberg Parameters for a 3-DoF manipulator are shown in Table 2.

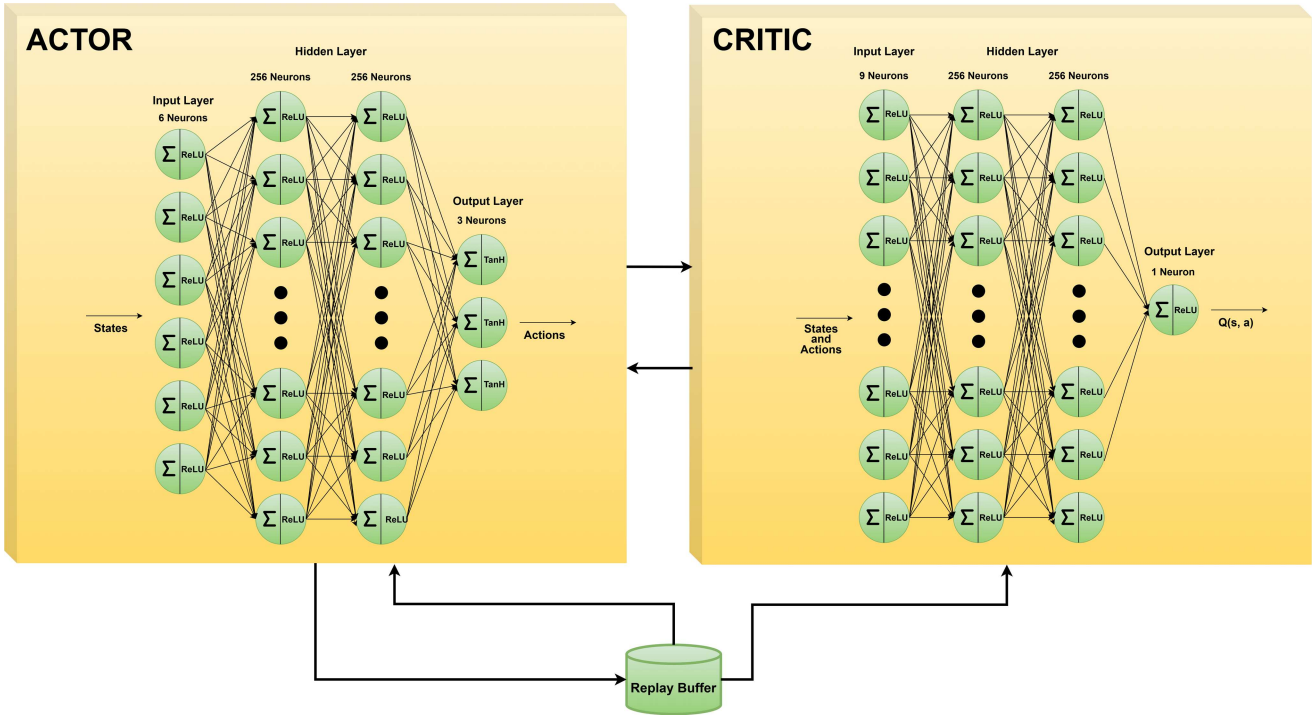
Table 2
Calculated values of Denavit–Hartenberg parameters

Link	Link length	Twist angle	Link offset	Joint angle
L1	0.15	90	0.0	θ_1
L2	0.15	0	0.0	θ_2
L3	0.05	0	0.0	θ_3

4.4. DDPG model architecture

The DDPG model architecture depicted in Figure 6 employs an MLP for both the actor and critic networks. The actor network has 6 input layers and 3 output layers, tailored to accommodate the dimensions of the state and action spaces, respectively. It has 2 hidden layers with 256 neurons in each layer. The Leaky ReLU activation function is utilized in the input and hidden layers to introduce nonlinearity and enhance learning capabilities. The output layer uses the TanH activation function, which provides joint angle values in the range of [−1, 1]. These values are then linearly transformed to the range of [−90, 90]. The critic network

Figure 6
Model architecture of DDPG agent



has 9 input layers and 1 output layer. It has 2 hidden layers with 256 neurons in each layer. The Leaky ReLU activation function is utilized throughout the model. The output layer provides the action-value function $Q(s, a)$, where s is state and a is action. This architecture is structured to facilitate effective learning and adaptation in complex reinforcement learning tasks.

4.5. TD3 model architecture

The TD3 model architecture depicted in Figure 7 follows a variant of the DDPG architecture with an additional critic network, enhancing its capacity for robust performance in reinforcement learning tasks. During training, this configuration involves comparing the action-value function $Q(s, a)$ predicted by both critic networks and selecting the minimum value to guide actor network updates. The architecture utilizes an MLP for both the actor and the two critics. The number of layers and neurons and the type of activation function used are similar to the DDPG architecture. This structured approach aims to optimize learning efficiency and achieve reliable performance in complex control scenarios.

5. Kinematic Analysis of 3-DoF Manipulator

The 3-DoF manipulator has been analyzed quite comprehensively in terms of its reachable points in the space of states. It was concluded that certain points were unreachable due to constraints associated with the manipulator's configuration. By identifying unreachable points owing to configuration restrictions, the training dataset was cleansed to improve the performance of the learning agent. This was done strategically to prevent unnecessary inefficiencies and errors on the part of the agent as it tries to access places that are difficult to reach. The reachable workspace of the manipulator is depicted graphically in Figures 8 and 9, showing three-dimensional and two-dimensional views of the reachable valid points, respectively. The careful selection of learning points increases the effectiveness of learning but also ensures that the agent operates within feasible configuration constraints, hence allowing reliable and accurate control of the manipulator.

6. Results and Discussions

In this section, the results and performance of our proposed framework are presented and discussed. We provide a detailed analysis of the results obtained from the hyperparameter tuning, training, and testing phases, as well as in-depth insights from the simulations and hardware performance evaluations.

6.1. Evaluation criteria

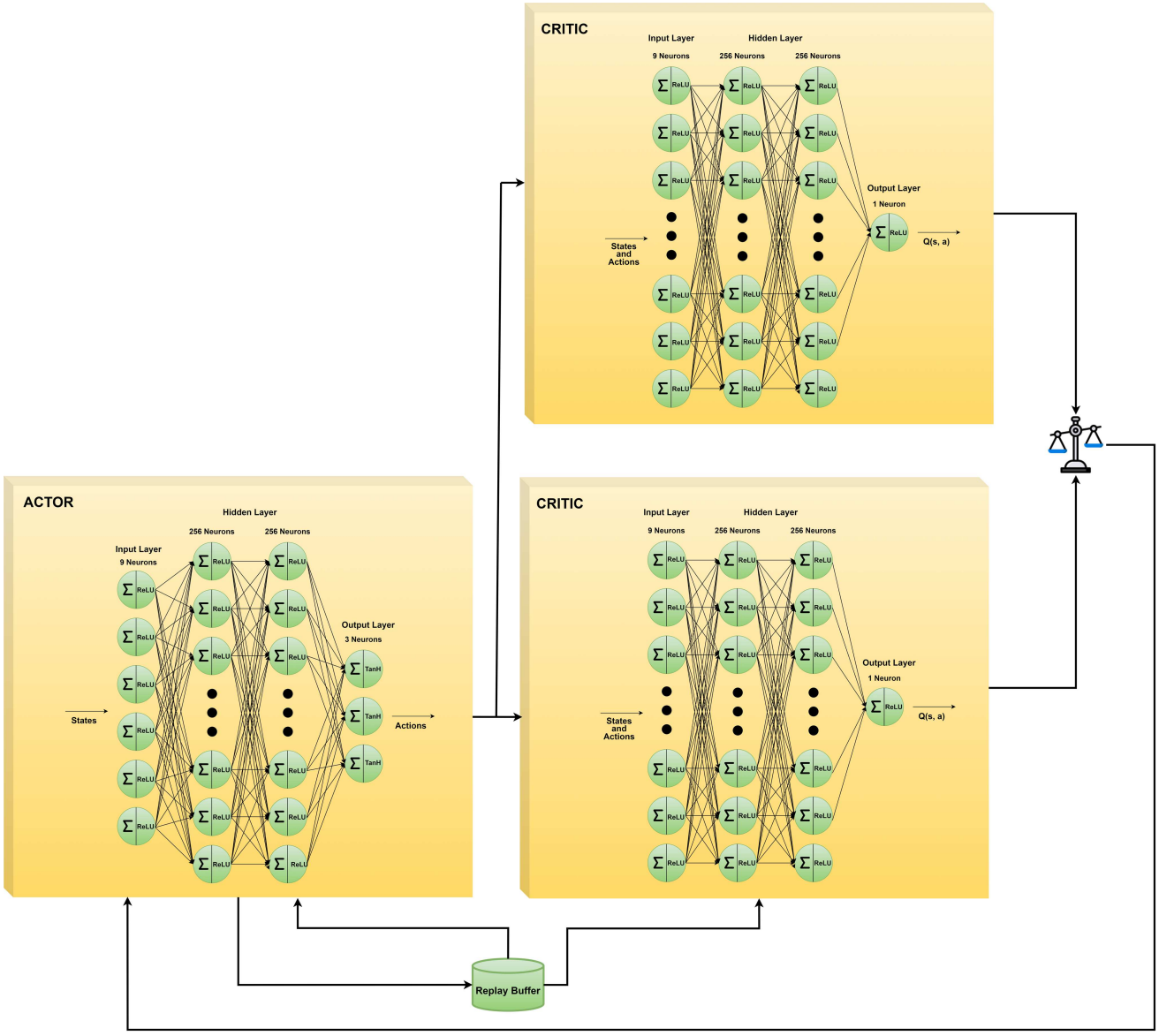
It will be important to set specific requirements in the performance analysis of our agent. For the case of the trained model, performance assessment is obtained through the evaluation of the success rate, defined as the proportion of successful episodes out of the total number of episodes, given by Equation (7).

$$Success\ Rate = \frac{Number\ of\ Successful\ Episodes}{Total\ Number\ of\ Episodes} \tag{7}$$

The second is the mean square error, defined as the average squared difference between the values estimated and the true values. It measures how well the model performed, as it describes whether the model's predictions fit the correct outcomes or not, given by Equation (8).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(e_i - e'_i)^2 \tag{8}$$

Figure 7
Model architecture of TD3 agent



where e_i represents the ground truth joint angle values, e'_i are the predicted joint angle values, and n is the total number of observations.

The third criterion is Euclidean distance. It is a measure of the two-dimensional, linear distance between two points within Euclidean space. This is the measure oftentimes utilized to state measures of similarity or dissimilarity between pairs of points, as described in Equation (9).

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{9}$$

where $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_n)$ are the coordinates of the two points in n - dimensional space. In reinforcement learning for manipulator control, Euclidean distance defines the measure of how close the end-effector goes to the target destination. This metric is used principally in simulation with the specially designed manipulator. A trial was considered successful if the Euclidean distance between the end-effector position and the target position was less than 0.01 m at the end of the episode.

6.2. Hyperparameter tuning

Tuning hyperparameters in reinforcement learning is crucial for optimizing algorithm performance, enhancing convergence speed, and improving stability. In this work, a systematic approach was employed to tune the hyperparameters. The hyperparameters considered include Number of Neurons, Hidden Layers, Learning Rate (α), Discount Factor (γ), Exploration Rate (ε), Soft Update Rate (τ), Batch Size, and Max Steps. The hyperparameters tuned for the DDPG algorithm are explained below. The initial values of the hyperparameters were randomly selected and shown in Table 3.

Figure 8
3D view of manipulator's reachable points

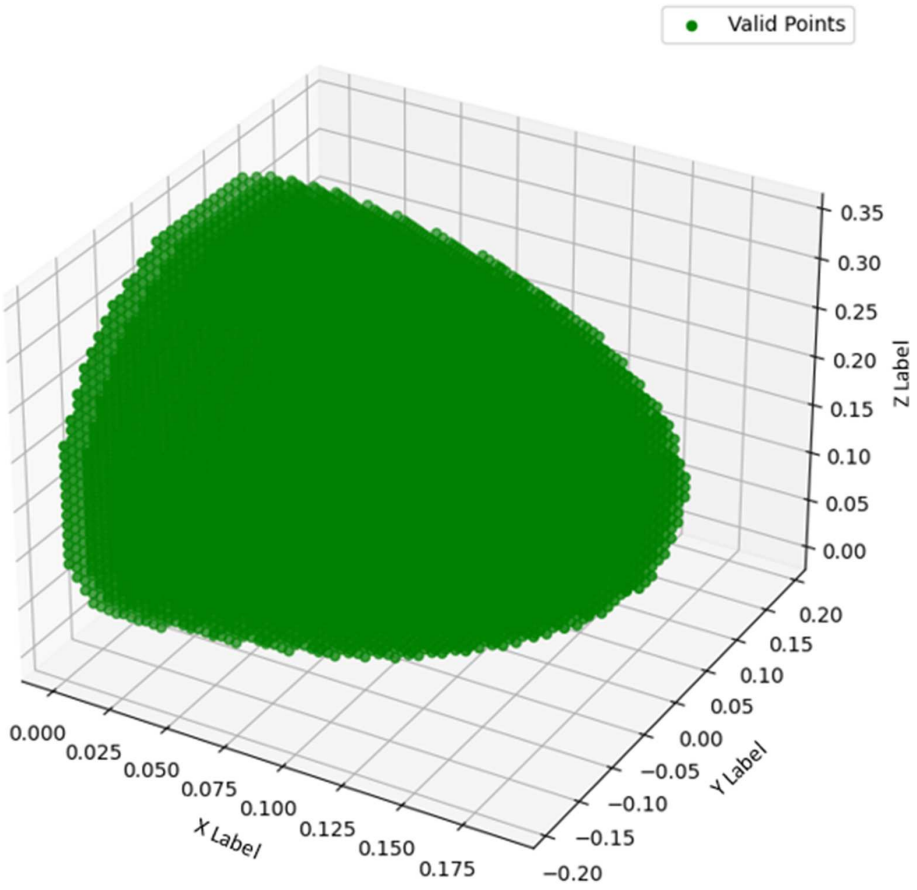


Figure 9
2D view of manipulator's reachable points

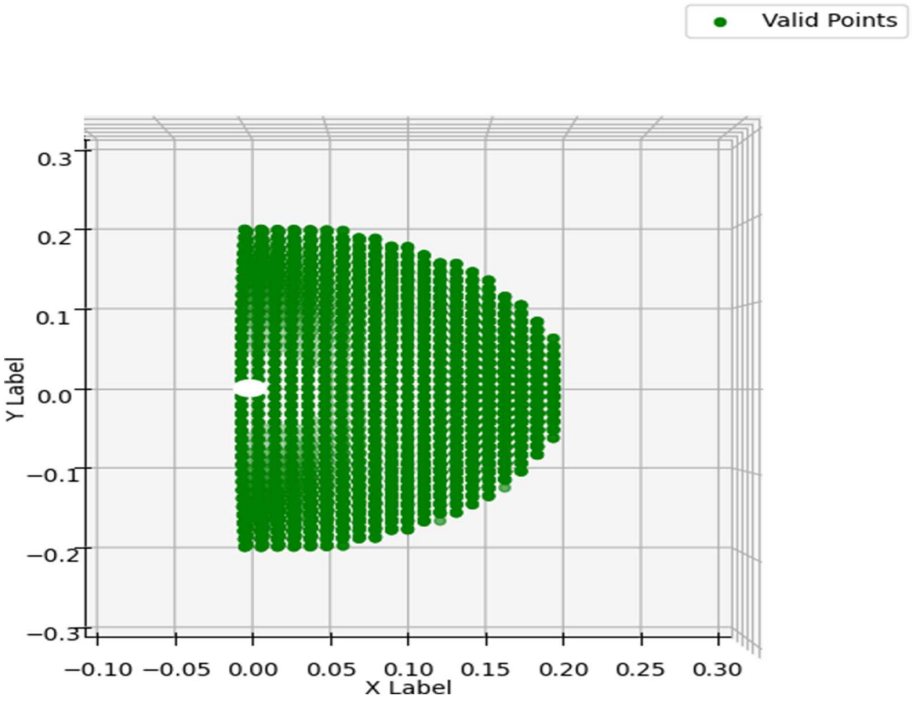


Table 3
Initial hyperparameter values

Hyperparameter	Values
Neurons	64
Hidden Layers	64–64
Actor Learning Rate and Critic Learning Rate	0.001–0.001
Gamma	0.9
Epsilon	0.3
Tau	0.003
Replay Buffer	10000
Batch Size	64
Max Step	64

6.2.1. Number of neurons

The tuning process began with adjusting the number of neurons in a single hidden layer. Keeping all other hyperparameters the same as in Table 3, initially, 64 neurons were used, and while the success rate increased, it quickly plateaued, indicating no further capacity for learning. Next, 128 neurons were tested, showing some initial promise but ultimately achieving the same success rate as 64 neurons. Subsequently, 256 neurons were evaluated, and this configuration outperformed the previous two, exhibiting a steady improvement in performance. As a result, 256 neurons were selected. These results are illustrated in Figure 10.

6.2.2. Hidden layers

Having the number of neurons in the first hidden layer fixed at 256 and all the other hyperparameters with the same value as in Table 3, different neuron counts were tested in the second hidden layer: 64, 128, and 256. Once again, the configuration with 256 neurons in the second layer outperformed the others, showing a significant increase in the success rate. Consequently, the model with two hidden layers, each containing 256 neurons, was selected. These results are illustrated in Figure 11.

6.2.3. Learning rate (α)

The learning rate in RL controls the magnitude of updates to model parameters during training, balancing speed and stability in convergence. In DDPG and TD3, separate learning rates for the actor and critic networks help optimize policy and value estimation independently; a lower rate for the critic stabilizes value estimates, while a higher rate for the actor allows faster policy improvement based on updated critiques. Having fixed the number of neurons and hidden layers and all other hyperparameters with the same value as in Table 3, different learning rates for actor

Figure 10
Hyperparameter tuning for number of neurons

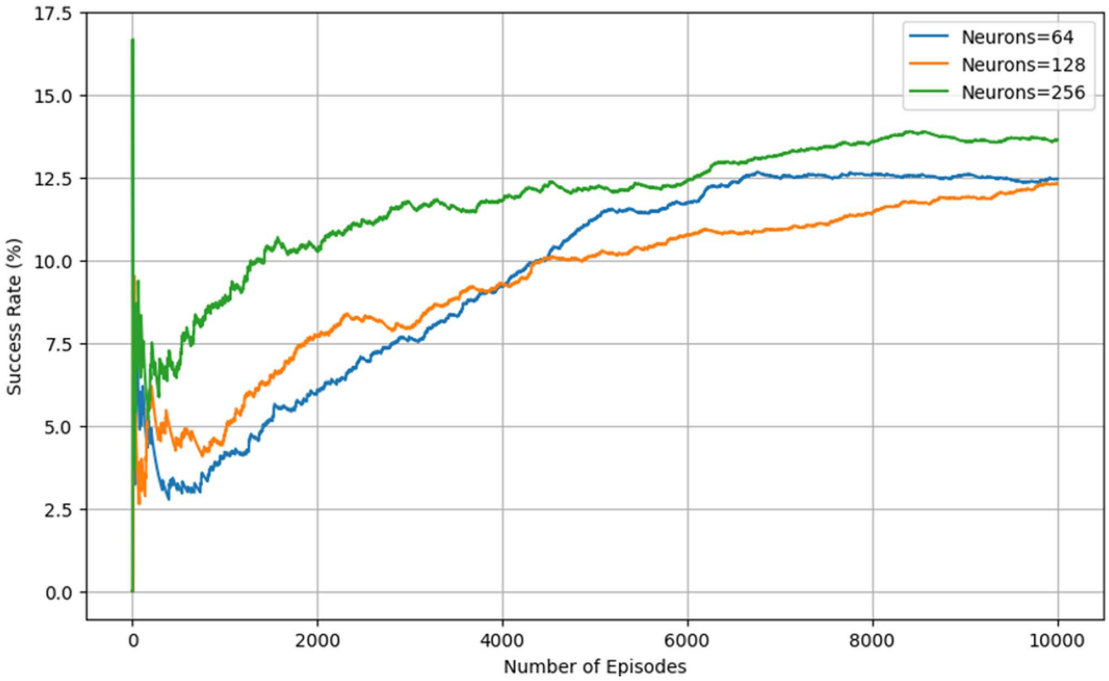
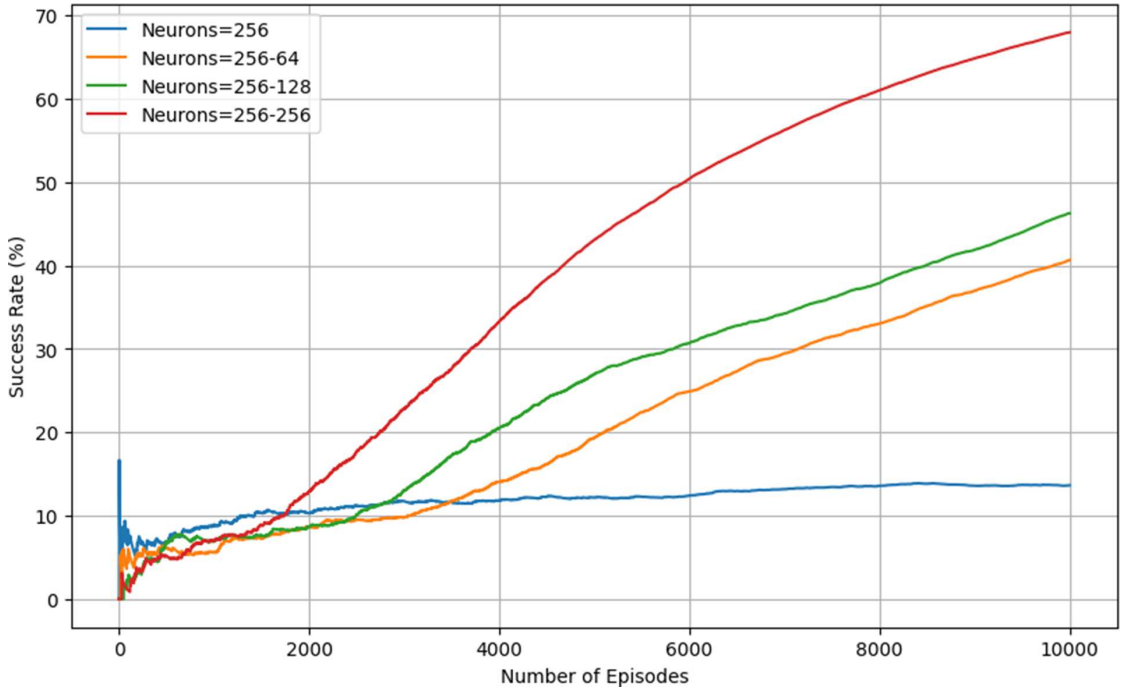


Figure 11
Hyperparameter tuning for hidden layers

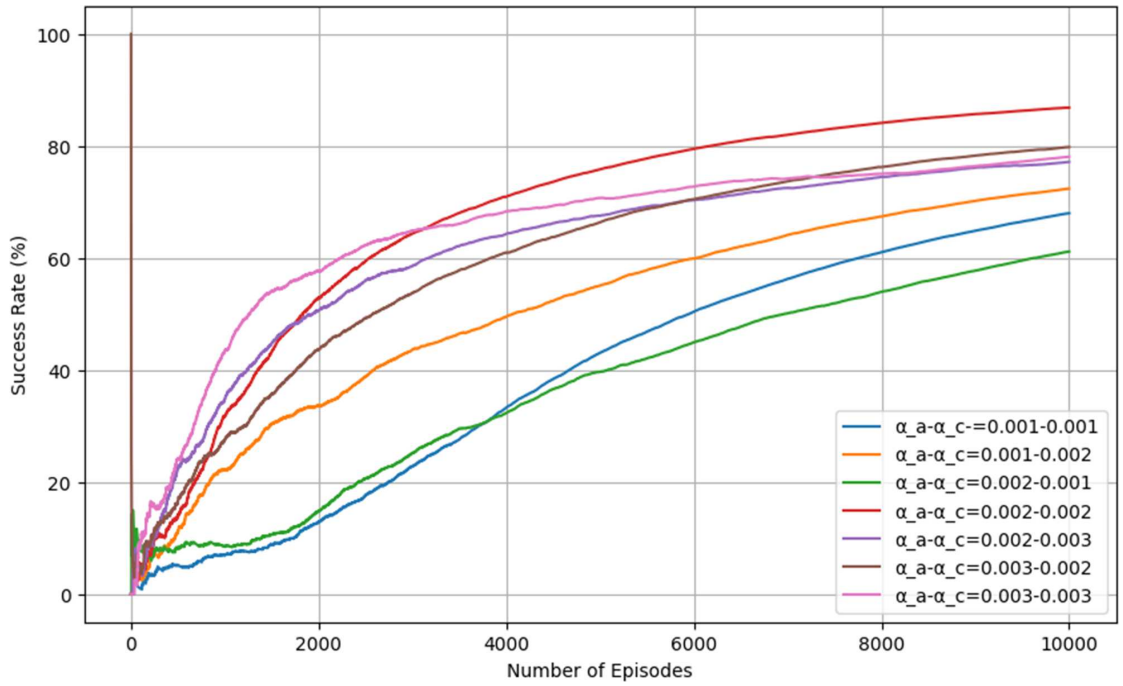


(α_a) and critic (α_c) were tested as shown in Figure 12. A learning rate of 0.002 for both the actor and the critic yielded the best performance in terms of success rate compared to other values.

6.2.4. Discount factor (γ)

The discount factor in RL determines the importance of future rewards by exponentially reducing their impact on current decisions. In DDPG and TD3, it balances immediate and long-term gains, guiding the critic network's value estimation by prioritizing rewards within a relevant time horizon, which indirectly influences the actor's policy updates. With the number of neurons, hidden layers, and learning rate held in the selected values and all the other hyperparameters with the same value as in Table 3, the discount factor was tested using values of 0.5, 0.7,

Figure 12
Hyperparameter tuning for learning rate (α)



0.9, 0.95, and 0.99. As illustrated in Figure 13, a discount factor of 0.9 resulted in a higher success rate compared to 0.99, 0.95, 0.7, and 0.5, although the differences were marginal. Therefore, a discount factor of 0.9 was selected.

6.2.5. Exploration rate (ε)

With the number of neurons, hidden layers, learning rate, and discount factor held in the selected values and all the other hyperparameters with the same value as in Table 3, the exploration rate values of 0.2, 0.3, and 0.4 were tested, with an exploration rate of 0.3 yielding the highest success rate compared to the others. This is illustrated in Figure 14. As a result, an epsilon of 0.3 was selected. The exploration rate in RL defines the probability with which the agent chooses to explore new actions rather than exploit known rewarding actions in epsilon-greedy strategies. A lower epsilon encourages exploitation, while a higher epsilon

Figure 13
Hyperparameter tuning for discount factor (γ)

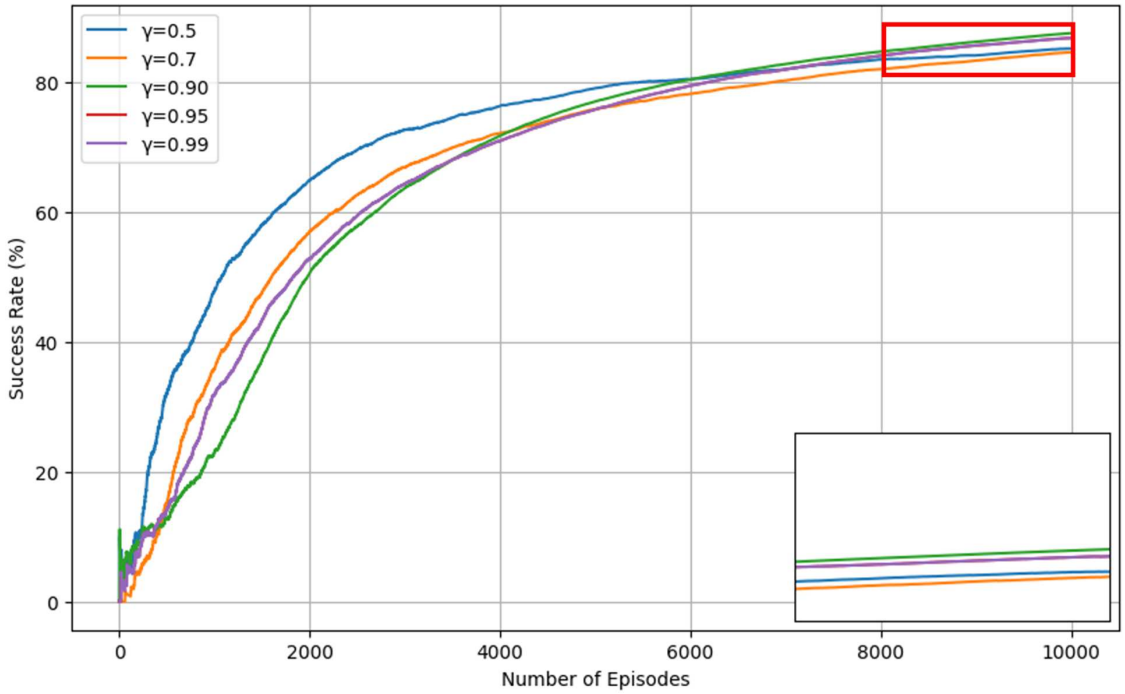
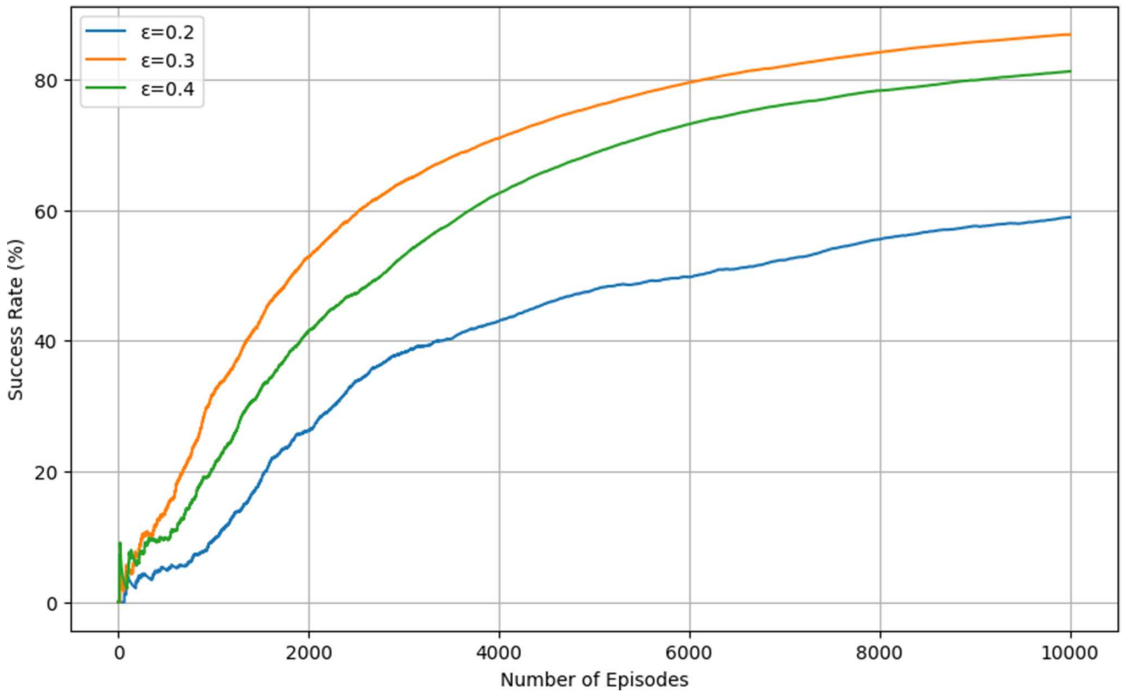


Figure 14
Hyperparameter tuning for epsilon (ϵ)



increases exploration. Balancing exploration and exploitation is crucial to ensure that the agent does not prematurely converge on suboptimal policies.

6.2.6. Soft update rate (τ)

The number of neurons, hidden layers, learning rate, discount factor, and exploration rate were held in the selected values, and all the other hyperparameters were set to the same value as in Table 3. The soft update rate was tuned, with a value of 0.003 delivering the best performance compared to other values; therefore, 0.003 was chosen, and this is illustrated in Figure 15. Here, the soft update rate controls the rate at which the target network is updated with the learned network's parameters. A smaller tau results in slower updates, ensuring smoother transitions and more stable learning, while a larger tau leads to quicker updates but may cause instability.

6.2.7. Batch size

The number of neurons, hidden layers, learning rate, discount factor, exploration rate, and soft update rate were held in the selected values, and all the other hyperparameters were set to the same value as in Table 3. Batch size in RL refers to the number of samples used to compute updates during training, affecting the stability and efficiency of learning. In DDPG and TD3, a larger batch size can lead to more accurate gradient estimates for both the actor and critic networks by averaging over diverse experiences, while smaller batch sizes may enable quicker updates and exploration but can introduce noise, potentially destabilizing training. The batch size was tested with values of 32, 64, and 128. As shown in Figure 16, a batch size of 64 yielded the highest success rate compared to 128 and 32, and therefore, 64 was selected.

6.2.8. Max step

The maximum number of steps per episode was tuned, with 256 steps yielding the highest success rate, although 128 steps were also very close, as shown in Figure 17. The maximum step limit determines how long an agent is allowed to interact with the environment in a single episode. A higher maximum step count can provide the agent with more opportunities to learn from its actions, while a lower count may expedite learning but can also limit the agent's ability to explore its environment thoroughly. Therefore, a maximum step count of 256 was selected for optimal performance.

The summary of the selected values for all the hyperparameters is shown in Table 4.

6.3. Training results

6.3.1. Deep reinforcement learning using DDPG

The DDPG algorithm was trained with a precision requirement of 0.01 m, aiming for the manipulator to accurately reach target positions within this tolerance. The agent demonstrated robust performance, achieving the desired destinations reliably within the specified tolerance. The accumulated rewards obtained over 20,000 training episodes are presented in Figure 18, where positive y-axis values denote the reward points accumulated by the agent during training. Evaluation of performance also included

Figure 15
Hyperparameter tuning for tau (τ)

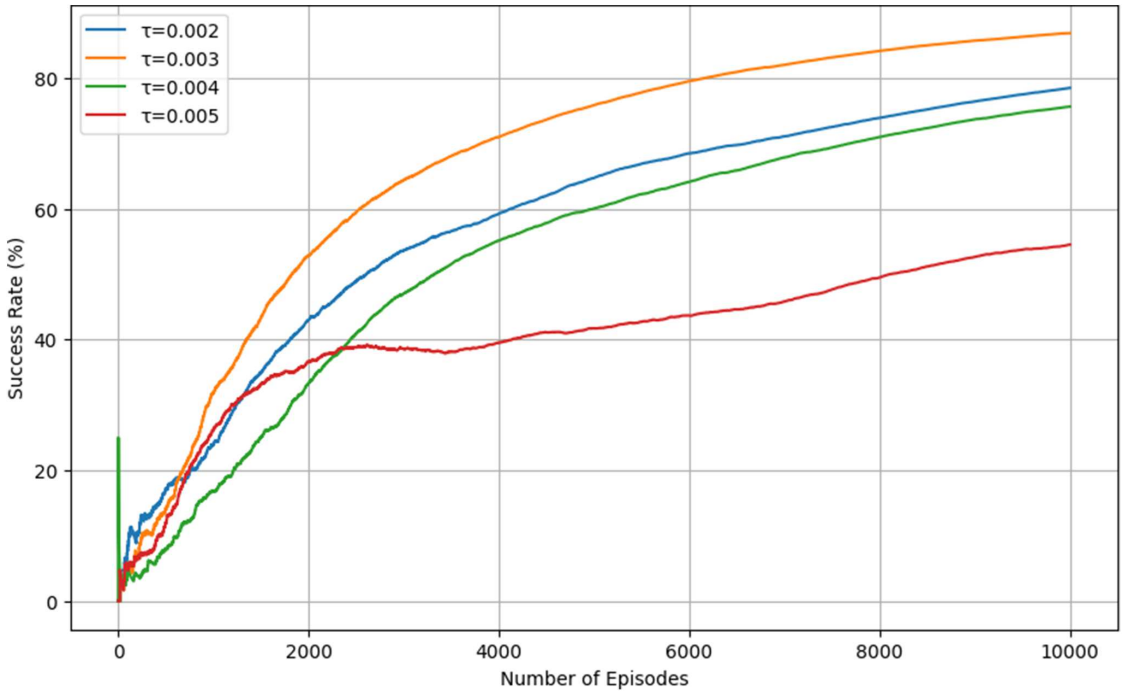


Figure 16
Hyperparameter tuning for batch size

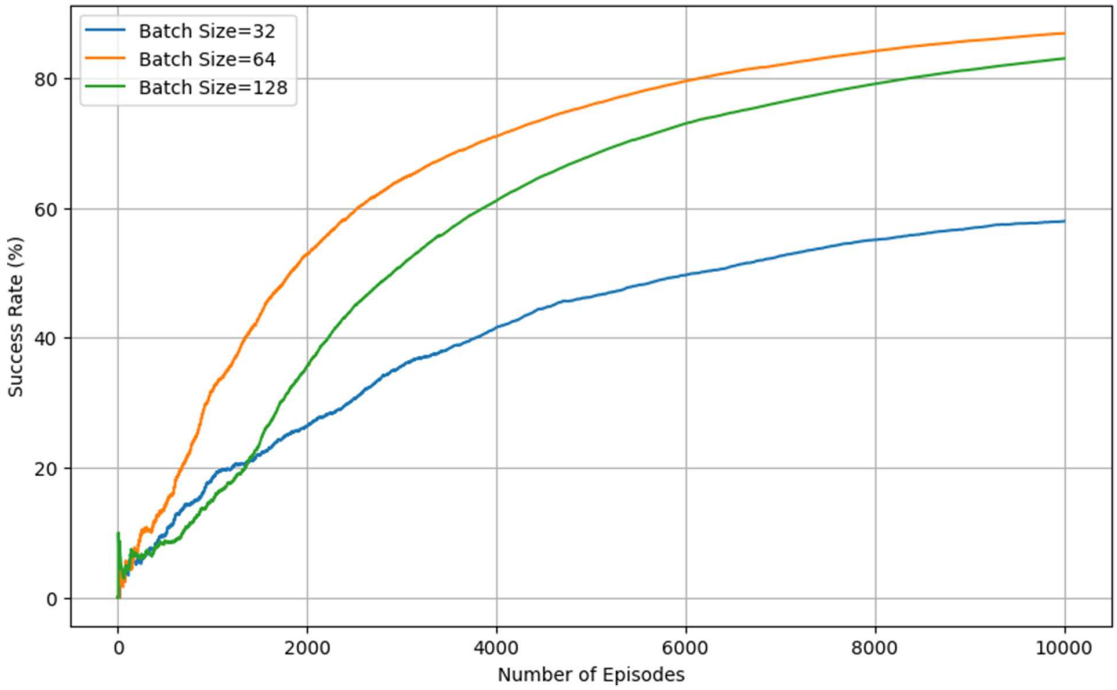


Figure 17
Hyperparameter tuning for max step

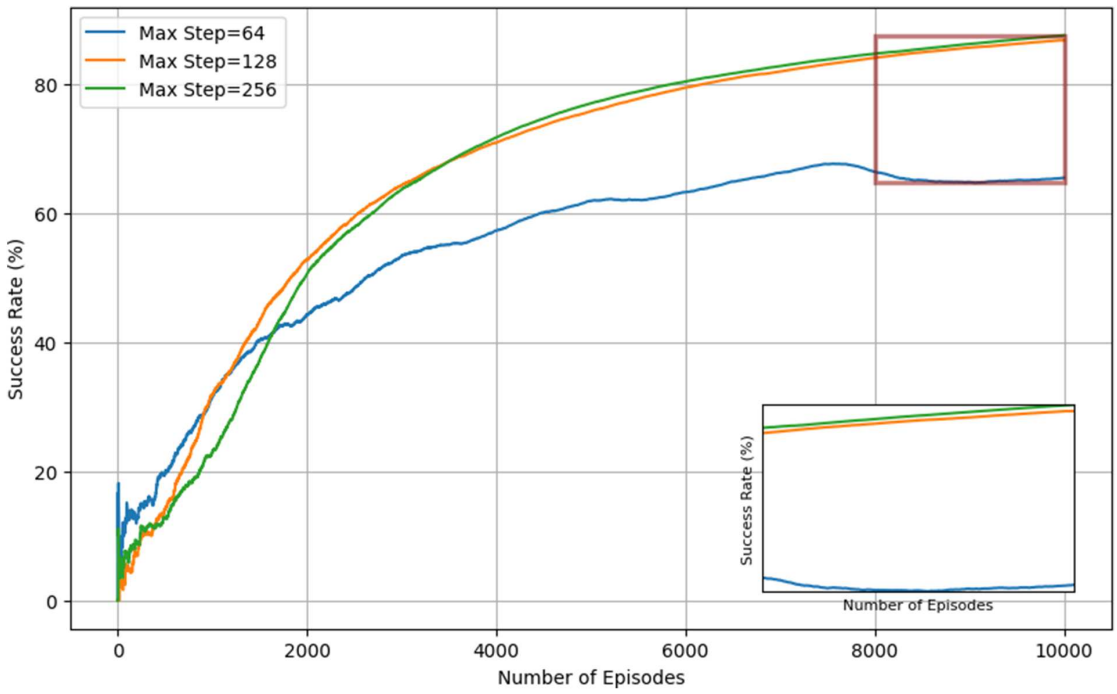
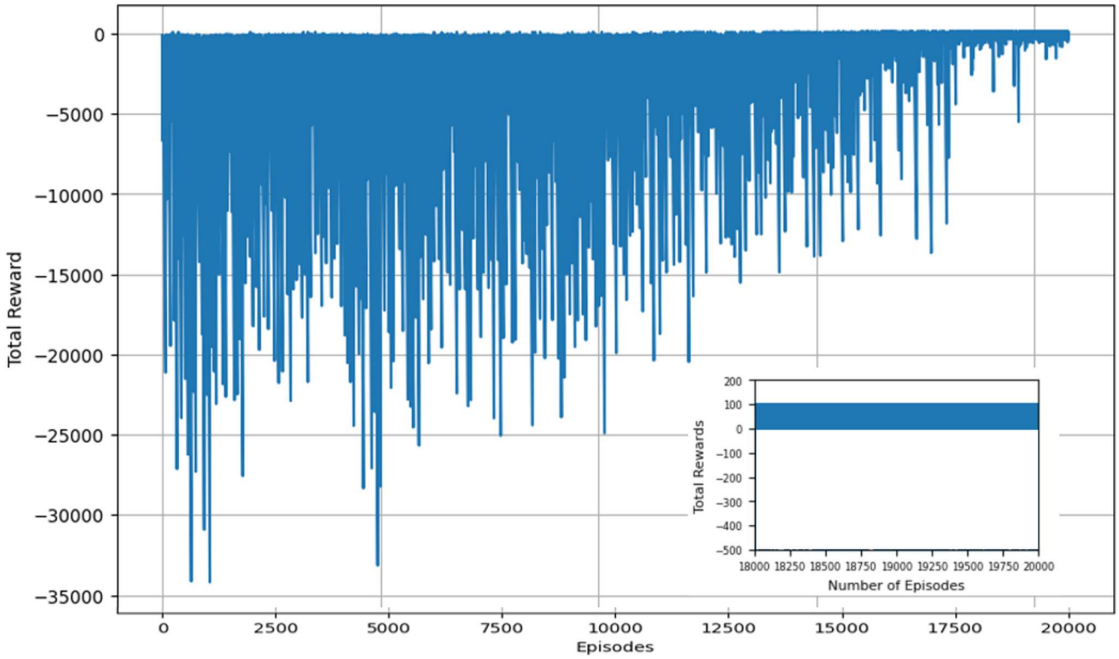


Table 4
Tuned hyperparameter values

Hyperparameter	Values
Neurons	256
Hidden Layers	256-256
Actor Learning Rate and Critic Learning Rate	0.002-0.002
Gamma	0.9
Epsilon	0.3
Tau	0.003
Replay Buffer	10000
Batch Size	64
Max Step	256

Figure 18
Accumulated rewards (DDPG algorithm)



a success rate analysis, as depicted in Figure 19, which indicated a commendable success rate of 93.72%. This metric underscores the model's effective learning and adaptation to its environment. Additionally, Figures 20 and 21 illustrate the Actor Mean Square Error and Critic Mean Square Error, respectively. In DDPG, the objective of the actor network is to learn a policy that maximizes the expected return by producing actions that lead to high Q-values as evaluated by the critic. To achieve this, the actor's loss function is defined as the negative Q-value of the chosen action, which the critic estimates given the current state, as shown in Equation (10).

$$L_{actor} = -Q(s, a) \tag{10}$$

where $-Q(s, a)$ represents the critic's estimate of the expected return for state s and action a chosen by the actor. Minimizing this loss function allows the actor to indirectly maximize the expected return, aligning with the goal of policy improvement. As training progresses, the actor's loss typically becomes more negative, reflecting enhanced policy performance through higher expected returns. Meanwhile, the critic's loss trends toward zero as training progresses, indicating effective learning and convergence of the model. Together, these findings demonstrate that the DDPG model has learned the task effectively and achieved proficient performance in controlling the manipulator within a challenging continuous state space.

6.3.2. Deep reinforcement learning using TD3

The TD3 algorithm was trained with a precision requirement of 0.01 m, necessitating the agent to accurately position the manipulator. Figure 22 illustrates the accumulated rewards obtained over 30,000 training episodes, where positive y-axis values represent the reward points accumulated by the agent during its learning process. Evaluation of performance included an analysis of success rate, as depicted in Figure 23, which indicated a notable success rate of 93.62%. This metric underscores the algorithm's effective learning and adaptation capabilities within its operational environment. While both algorithms achieved high success rates, the DDPG-based agent converged in fewer episodes compared to TD3, suggesting faster policy stabilization under our

Figure 19
Success rate (DDPG algorithm)

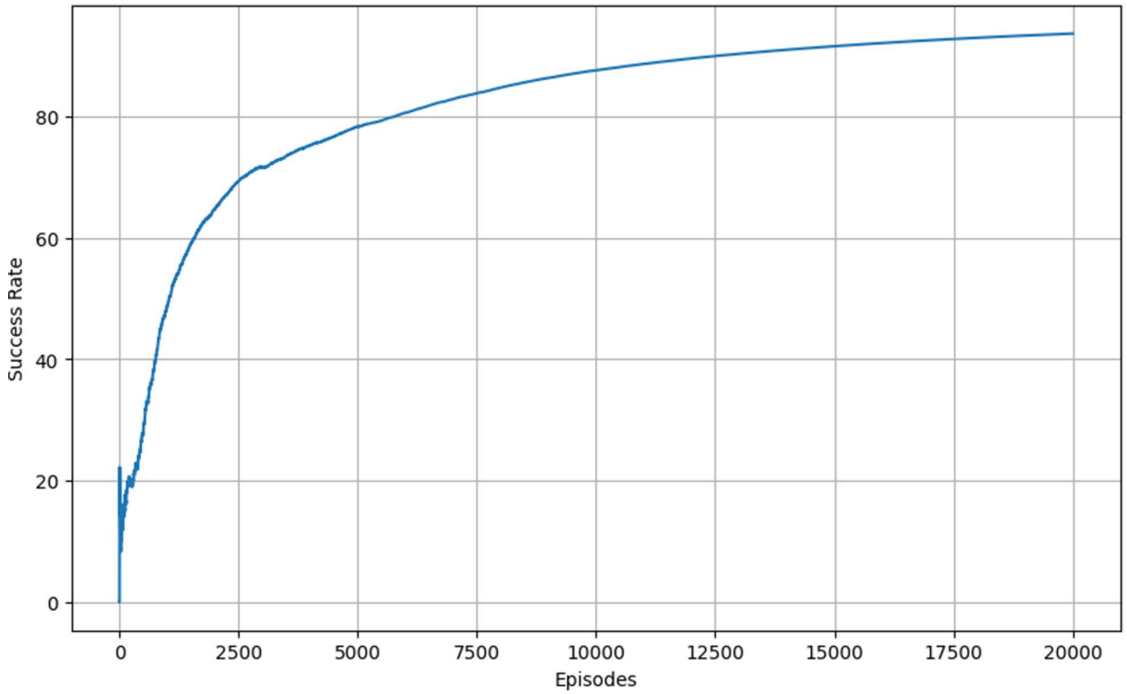


Figure 20
Actor network Mean Squared Error (MSE)

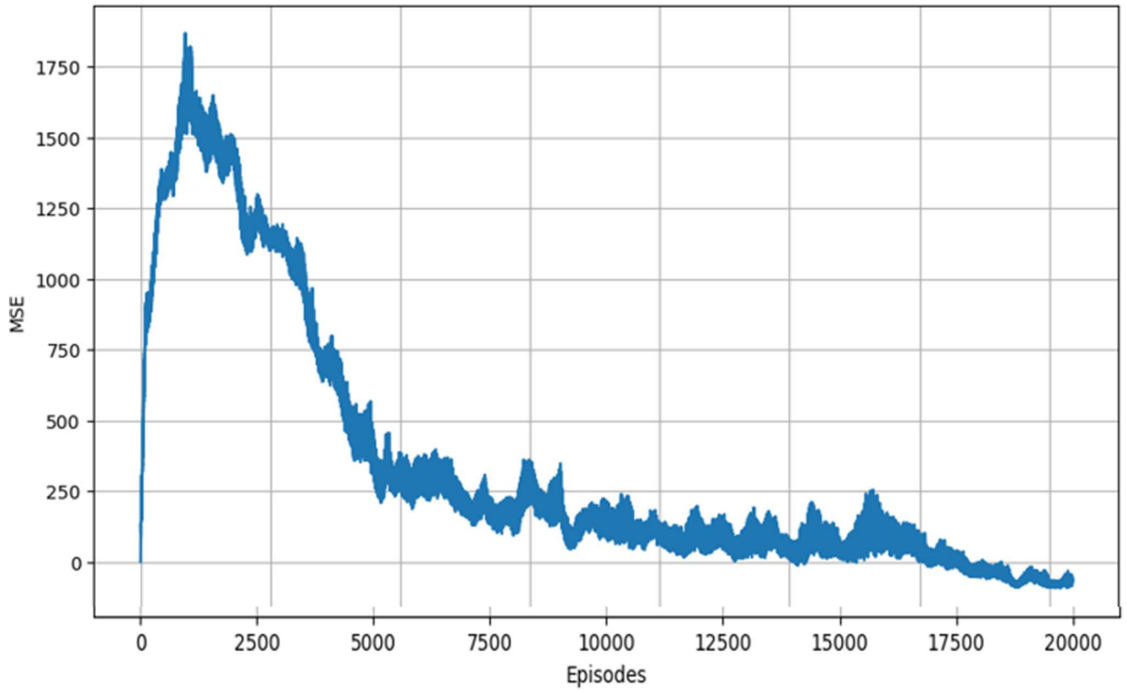


Figure 21
Critic network MSE

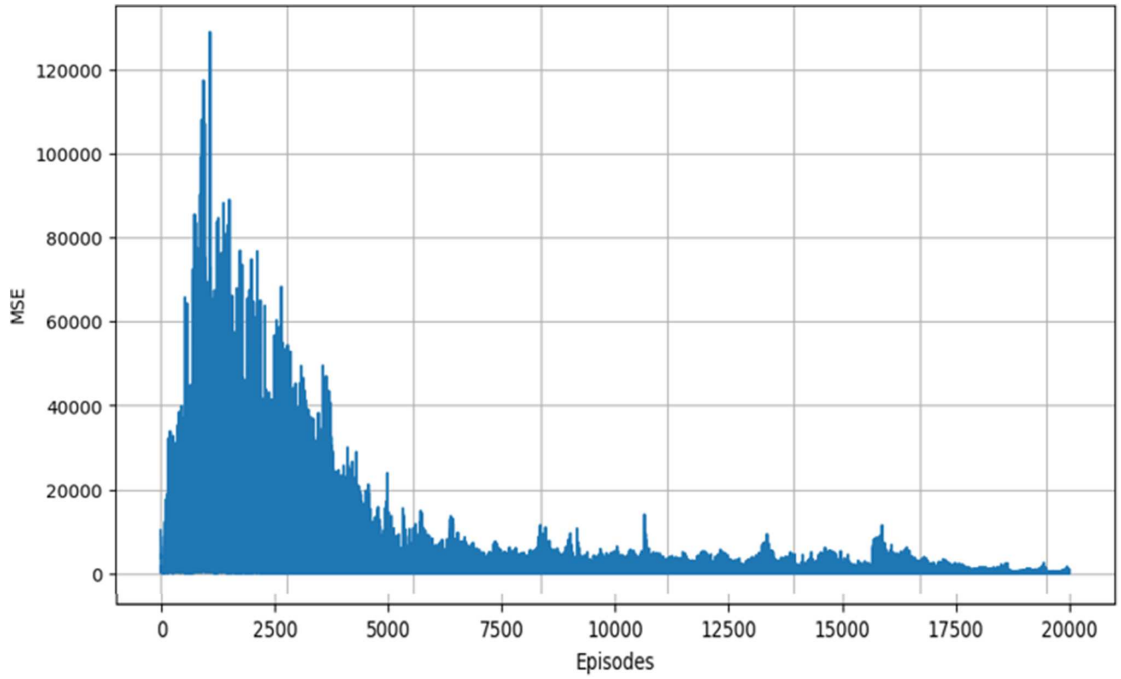


Figure 22
Accumulated rewards (TD3 algorithm)

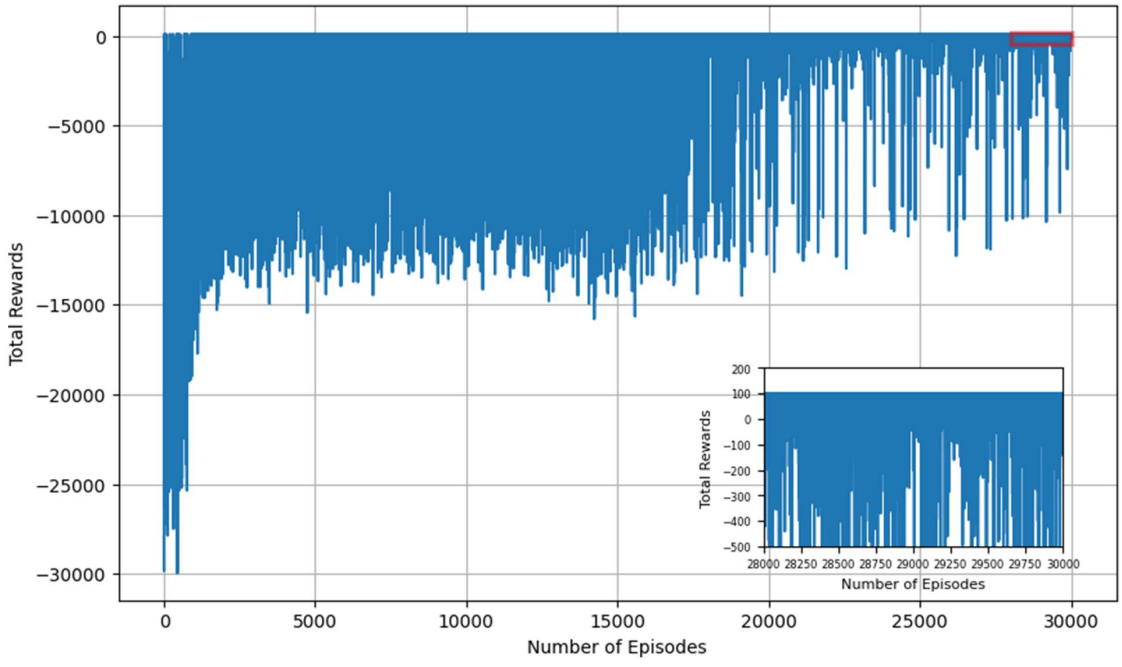
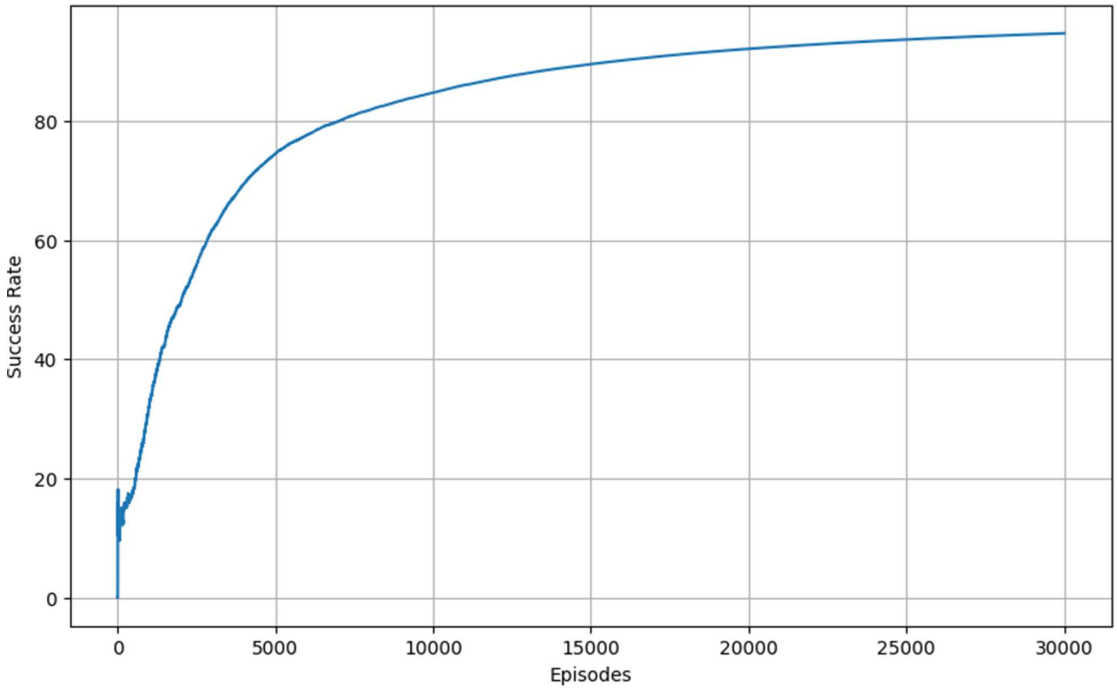


Figure 23
Success rate (TD3 algorithm)



simplified network and reward settings. This observation aligns partially with the findings in References [38, 39], where TD3 generally provides greater robustness but at the cost of longer training due to its dual critic structure and policy delay mechanisms. Our empirical results demonstrate that for this specific 3-DoF task, DDPG offers a better trade-off between performance and training efficiency.

6.4. Testing results

Following training with DDPG and TD3, the models were evaluated in the manipulator's workspace environment, and the testing success rates are summarized in Table 5. In training, the actor–critic networks learn by interacting with the environment, using experience replay and target networks for stability. In testing, only the trained actor network is used to select deterministic actions without exploration noise. The success rate for both algorithms was tested over 100,000 trials. The end-effector of the manipulator was able to reach the target destination 95.77% of the time within a threshold of 0.01 m for the DDPG algorithm. The remaining were within a threshold of 0.015–0.020 m. The end-effector of the manipulator was able to reach the target destination 94.71% of the time within a threshold of 0.01 m for the TD3 algorithm. The remaining were within a threshold of 0.015–0.020 m. Therefore, during testing, DDPG exhibited a marginally higher success rate compared to TD3. Even though the difference is marginal, ultimately, DDPG was chosen as the final algorithm due to its comparative simplicity relative to TD3, its lesser time taken to train the agent, and its strong performance in the test scenarios. This selection underscores DDPG's effectiveness in achieving desired outcomes in the manipulator control task, aligning with practical considerations of simplicity and performance efficacy.

Table 5
Success rate of the DRL algorithms

Algorithm	Training (%)	Testing (%)
DDPG	93.72	95.77
TD3	93.62	94.71

6.5. Simulation

The Gazebo simulator was utilized to simulate the manipulator. The trained agent model was integrated into the control node to choose the next action of the manipulator. Demonstrated in Figures 24 and 25, the manipulator exhibits its capability to navigate from any random initial state to a predefined target point. It achieves this with precision, typically within 0.01 m of the target position. The target that the manipulator has to reach is represented as a yellow-highlighted box.

6.6. Hardware

To assess the real-time performance of the trained model, a physical 3-DoF manipulator was constructed and integrated with hardware components. The robot is controlled through an Arduino UNO board and servo motors rated at 83.3 RPM with a torque capacity of 1 kg/cm. A task was considered successful when the end-effector reached the target position within an error threshold of 0.01 m from the desired destination. This success criterion was consistently applied during both simulation and real-world experimental evaluation. The manipulator's joint angles are sent by an Arduino microcontroller, interfacing with a Python script that executes the trained model's predictions. Figures 26 and 27 visually depict the manipulator in operation during these real-time tests. The model successfully maneuvered the manipulator to designated target destinations, achieving

Figure 24
Manipulator moving to target: [0.07, −0.18, 0.16]. (a) Initial position of the manipulator and (b) final position of the manipulator

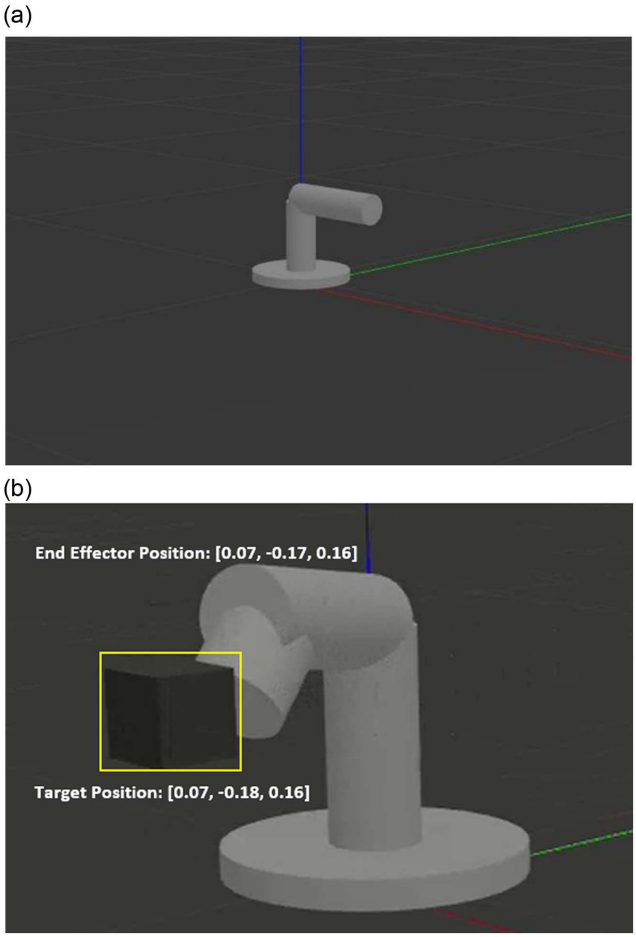
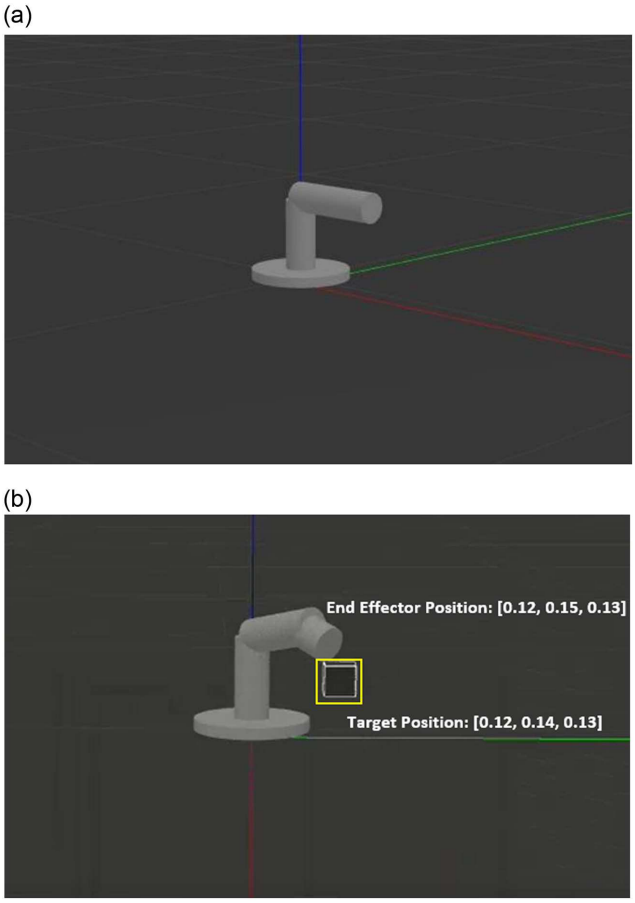


Figure 25
Manipulator moving to target: [0.12, 0.14, 0.13]. (a) Initial position of the manipulator and (b) final position of the manipulator



positional accuracy within a tolerance of 0.01 m. Verification of these results was conducted using accompanying graph sheets, confirming the manipulator's precise movements and validating the efficacy of the trained model in practical hardware implementation. To evaluate the positioning accuracy of the manipulator in the physical setup, a calibrated graph sheet with a 0.01 m grid resolution was placed beneath the workspace to provide a spatial reference for the end-effector position. The target positions generated by the DRL policy were visually aligned with the grid intersections, allowing the positional deviation between the commanded target and the achieved end-effector location to be estimated. Multiple trials were conducted across different target locations within the reachable workspace to assess the consistency of the learned policy. The observed positioning error remained within approximately 0.01 m, confirming that the trained agent was able to reproduce the target positions with satisfactory accuracy in the real-world environment.

Although differences typically exist between simulation environments and physical robotic systems, explicit calibration or compensation strategies were not required in this study. The simulation environment was implemented in Gazebo using a robot model that closely matched the physical manipulator in terms of link dimensions, joint configuration, and kinematic structure. Because the system is a relatively simple 3-DoF manipulator operating in a controlled environment without external disturbances, the trained policies transferred directly from simulation to hardware with minimal performance degradation. Nevertheless, in more complex robotic systems or dynamic environments, additional techniques such as domain randomization, system identification, or sim-to-real adaptation strategies may be necessary to further reduce the simulation–reality gap.

Figure 26
Manipulator moving to target: [0.07, −0.18, 0.16]

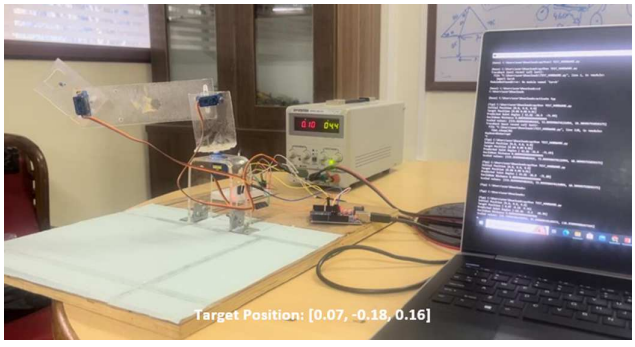
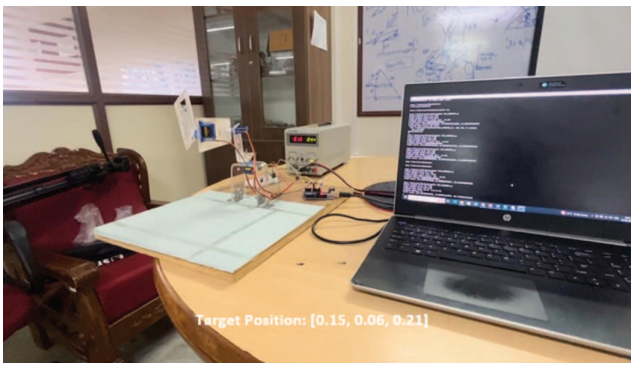


Figure 27
Manipulator moving to target: [0.15, 0.06, 0.21]



7. Discussion

This study explores the application of DRL as an alternative to traditional IK for manipulator control in continuous workspaces. The results obtained from both simulation and real-world experiments affirm the effectiveness of the proposed DRL framework as an alternative to traditional IK for manipulator control. The use of actor–critic methods, particularly DDPG and Twin Delayed DDPG (TD3), enabled continuous joint angle prediction with high accuracy, eliminating the need for explicit kinematic modeling. The performance metrics—task success rates of 95.77% for DDPG and 94.71% for TD3, and end-effector positioning accuracy consistently within 0.01 m—highlight the practical reliability of the learned policies.

One of the key findings was the superior convergence speed and marginally better accuracy achieved by DDPG compared to TD3. This contrasts with existing literature [17, 40], where TD3 is often preferred for its robustness and stability in high-dimensional, stochastic environments. In our experimental setup, however, the combination of a low-dimensional action space, a simple reward function, and a lightweight MLP architecture may have favored DDPG by reducing policy variance and improving sample efficiency. These observations reinforce the notion that algorithm selection should be task-specific and that increased architectural complexity does not always yield better outcomes.

The proposed piecewise-linear reward function contributed significantly to the learning stability and interpretability of the training process. Prior works [19, 20, 22, 35] commonly employ complex or compound reward structures involving trigonometric or exponential terms, which, while expressive, often require extensive tuning and can introduce instability. Our empirical results suggest that minimalist reward design, when well-aligned with task objectives, can offer a practical alternative for effective DRL training, particularly in scenarios where precise control (within 0.01 m of the target) is required without high computational cost.

The framework demonstrated robust sim-to-real transfer without relying on domain randomization, adversarial training, or extensive parameter tuning. The real-world manipulator closely mirrored the simulated policy behavior, with no significant degradation in task performance. This finding underscores the generalizability of the learned policies and validates the use of physics-consistent simulation environments (e.g., Gazebo + ROS2) as effective proxies for training reinforcement learning agents.

Although this study focuses on a 3-DoF manipulator, the use of a lower-dimensional system was intentional. It provides a sufficiently complex yet tractable platform for evaluating learning behavior, reward shaping, and sim-to-real performance of DRL-based control. This controlled setup enables clearer attribution of performance improvements to architectural and algorithmic choices while avoiding confounding factors associated with higher-dimensional redundancy. This approach is consistent with several benchmark DRL studies [15, 17, 22], which begin with low-DoF systems for algorithm development. Nevertheless, extending the framework to higher-DoF manipulators, more complex tasks, and dynamic environments remains necessary to fully establish its scalability.

Although TD3 was proposed to address instability issues in DDPG through techniques such as clipped double Q-learning and delayed policy updates, the experimental results in this study showed comparable performance between the two algorithms, with DDPG exhibiting slightly faster convergence. This outcome is likely influenced by the relatively low-dimensional control problem considered in this work, involving a 3-DoF manipulator operating in a controlled environment. Under such conditions, the overestimation bias and instability that TD3 is designed to mitigate may not be strongly manifested.

8. Conclusion and Future Scope

This study proposed a DRL-based control framework for a 3-DoF robotic manipulator operating in a continuous three-dimensional workspace. It serves as an alternative to traditional IK methods by using actor–critic DRL algorithms, specifically DDPG and Twin Delayed DDPG, to enable direct joint angle prediction based on end-effector target positions. A key aspect of the proposed method was the use of a simple, piecewise-linear reward function, which reduced training complexity while ensuring stable convergence and precise control. The proposed linear reward formulation reduced reward design complexity while achieving faster convergence of approximately 31% fewer training episodes compared with the evaluated nonlinear reward functions, demonstrating that simplified reward engineering can be effective for continuous manipulator control tasks.

The manipulator was modeled and trained in the Gazebo simulation environment using ROS2 and tested in both simulated and real-world conditions. Training was performed for 20,000 episodes for DDPG and 30,000 episodes for TD3, respectively. The trained agents achieved high task success rates: 95.77% for DDPG and 94.71% for TD3, with consistent end-effector positioning accuracy within 0.01 m. The DDPG agent exhibited faster convergence and slightly better accuracy than TD3 in this setup, which may be attributed to the simplicity of the reward function and the task environment. Despite TD3 being generally known for greater stability due to its clipped double Q-learning and delayed policy updates [17, 40], our empirical findings suggest that DDPG offers improved sample efficiency and faster stabilization in this specific 3-DoF scenario.

The proposed architecture, based on a simple MLP, demonstrated consistent performance across simulation and hardware, highlighting the effectiveness of the sim-to-real transfer without the use of domain randomization. This validates the robustness and applicability of the proposed approach in real-time control of robotic manipulators.

That said, several limitations remain. The system was tested in a static and obstacle-free environment, and the manipulator was constrained to a 3-DoF configuration. While this reduced-dimensional setup was intentionally chosen to isolate the effects of learning architecture and reward design, it does not represent

the challenges of higher-dimensional, redundant, or dynamically interacting manipulators. Furthermore, the impact of disturbances, sensor noise, and external environmental variability was not explored in the current scope.

In future work, several research directions will be explored to address the current limitations of the proposed framework. First, the approach will be extended to manipulators with higher degrees of freedom (e.g., 6-DoF and above), where redundancy and increased joint coupling introduce additional control complexity. Techniques such as hierarchical reinforcement learning or modular policy architectures may be investigated to manage the expanded action space efficiently. Second, the framework will be enhanced to operate in dynamic and uncertain environments by incorporating obstacle avoidance and disturbance handling. This could involve integrating perception modules such as computer vision or depth sensing to enable environment-aware decision-making. Third, improvements in sim-to-real transfer will be investigated using techniques such as domain randomization and adaptive policy learning to improve robustness against modeling inaccuracies. Finally, the proposed control strategy will be evaluated on more complex robotic tasks, including pick-and-place operations, assembly processes, and human–robot collaborative scenarios to assess its practical applicability in industrial and service robotics.

Funding Support

This research was funded by Amrita Vishwa Vidyapeetham under the guidance of Chancellor, Sri Mata Amritanandamayi Devi (Amma).

Ethical Statement

This study does not contain any studies with human subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data are available from the corresponding author upon reasonable request.

Author Contribution Statement

Shivkumar Sankaralingam: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **NippunKumaar Arulmani Angamuthu:** Conceptualization, Methodology, Formal analysis, Investigation, Writing – review & editing, Visualization, Supervision. **Suja Palaniswamy:** Writing – review & editing, Visualization, Project administration, Funding acquisition.

References

[1] Spong, M. W. (2022). An historical perspective on the control of robotic manipulators. *Annual Review of Control, Robotics, and Autonomous Systems*, *5*, 1–31. https://doi.org/10.1146/annurev-control-042920-094829

[2] Chotikunnan, P., & Chotikunnan, R. (2023). Dual design PID controller for robotic manipulator application. *Journal of Robotics and Control*, *4*(1), 23–34. https://doi.org/10.18196/jrc.v4i1.16990

[3] Şen, G. D., & Öke Günel, G. (2023). NARMA-L2–based online computed torque control for robotic manipulators. *Transactions of the Institute of Measurement and Control*, *45*(13), 2446–2458. https://doi.org/10.1177/01423312231153255

[4] Azeez, M. I., Abdelhaleem, A. M. M., Elnaggar, S., Moustafa, K. A., & Atia, K. R. (2023). Optimized sliding mode controller for trajectory tracking of flexible joints three-link manipulator with noise in input and output. *Scientific Reports*, *13*(1), 12518. https://doi.org/10.1038/s41598-023-38855-7

[5] Wang, Z., Zou, L., Su, X., Luo, G., Li, R., & Huang, Y. (2021). Hybrid force/position control in workspace of robotic manipulator in uncertain environments based on adaptive fuzzy control. *Robotics and Autonomous Systems*, *145*, 103870. https://doi.org/10.1016/j.robot.2021.103870

[6] Liu, Y., Jiang, D., Yun, J., Sun, Y., Li, C., Jiang, G., . . . , & Fang, Z. (2022). Self-tuning control of manipulator positioning based on fuzzy PID and PSO algorithm. *Frontiers in Bioengineering and Biotechnology*, *9*, 817723. https://doi.org/10.3389/fbioe.2021.817723

[7] Kasruddin Nasir, A. N., Ahmad, M. A., & Tokhi, M. O. (2022). Hybrid spiral-bacterial foraging algorithm for a fuzzy control design of a flexible manipulator. *Journal of Low Frequency Noise, Vibration and Active Control*, *41*(1), 340–358. https://doi.org/10.1177/14613484211035646

[8] Ajwad, S. A., Iqbal, J., Ullah, M. I., & Mehmood, A. (2015). A systematic review of current and emergent manipulator control approaches. *Frontiers of Mechanical Engineering*, *10*(2), 198–210. https://doi.org/10.1007/s11465-015-0335-0

[9] Liu, Z., Peng, K., Han, L., & Guan, S. (2023). Modeling and control of robotic manipulators based on artificial neural networks: A review. *Iranian Journal of Science and Technology, Transactions of Mechanical Engineering*, *47*(4), 1307–1347. https://doi.org/10.1007/s40997-023-00596-3

[10] Jin, L., Li, S., Yu, J., & He, J. (2018). Robot manipulator control using neural networks: A survey. *Neurocomputing*, *285*, 23–34. https://doi.org/10.1016/j.neucom.2018.01.002

[11] Rawat, D., Gupta, M. K., & Sharma, A. (2023). Intelligent control of robotic manipulators: A comprehensive review. *Spatial Information Research*, *31*(3), 345–357. https://doi.org/10.1007/s41324-022-00500-2

[12] Liu, A., Zhao, H., Song, T., Liu, Z., Wang, H., & Sun, D. (2021). Adaptive control of manipulator based on neural network. *Neural Computing and Applications*, *33*(9), 4077–4085. https://doi.org/10.1007/s00521-020-05515-0

[13] Sahu, V. S. D. M., Samal, P., & Panigrahi, C. K. (2022). Modelling, and control techniques of robotic manipulators: A review. *Materials Today: Proceedings*, *56*, 2758–2766. https://doi.org/10.1016/j.matpr.2021.10.009

[14] Ren, H., & Ben-Tzvi, P. (2020). Learning inverse kinematics and dynamics of a robotic manipulator using generative adversarial networks. *Robotics and Autonomous Systems*, *124*, 103386. https://doi.org/10.1016/j.robot.2019.103386

[15] Malik, A., Lischuk, Y., Henderson, T., & Prazenica, R. (2022). A deep reinforcement-learning approach for inverse kinematics solution of a high degree of freedom robotic

manipulator. *Robotics*, *11*(2), 44. https://doi.org/10.3390/robotics11020044

[16] Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274. https://doi.org/10.1177/0278364913495721

[17] Han, D., Mulyana, B., Stankovic, V., & Cheng, S. (2023). A survey on deep reinforcement learning algorithms for robotic manipulation. *Sensors*, *23*(7), 3762. https://doi.org/10.3390/s23073762

[18] Li, Q., Nie, J., Wang, H., Lu, X., & Song, S. (2021). Manipulator motion planning based on actor-critic reinforcement learning. In *2021 40th Chinese Control Conference*, 4248–4254. https://doi.org/10.23919/CCC52363.2021.9550010

[19] Li, X., Liu, H., & Dong, M. (2022). A general framework of motion planning for redundant robot manipulator based on deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, *18*(8), 5253–5263. https://doi.org/10.1109/TII.2021.3125447

[20] Joshi, S., Kumra, S., & Sahin, F. (2020). Robotic grasping using deep reinforcement learning. In *2020 IEEE 16th International Conference on Automation Science and Engineering, 1461-1466*. https://doi.org/10.1109/CASE48305.2020.9216986

[21] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., . . . , & Wierstra, D. P. (2020). *U.S. Patent No. 10,776,692*. U.S. Patent and Trademark Office.

[22] Luipers, D., Kaulen, N., Chojnowski, O., Schneider, S., Richert, A., & Jeschke, S. (2022). Robot control using model-based reinforcement learning with inverse kinematics. In *2022 IEEE International Conference on Development and Learning*, 244–249. https://doi.org/10.1109/ICDL53763.2022.9962215

[23] Honerkamp, D., Welschehold, T., & Valada, A. (2021). Learning kinematic feasibility for mobile manipulation through deep reinforcement learning. *IEEE Robotics and Automation Letters*, *6*(4), 6289–6296. https://doi.org/10.1109/LRA.2021.3092685

[24] Calderon-Cordova, C., & Sarango, R. (2023). A deep reinforcement learning algorithm for robotic manipulation tasks in simulated environments. *Engineering Proceedings*, *47*(1), 12. https://doi.org/10.3390/engproc2023047012

[25] Kim, S., Jang, I., Kim, H., Park, C.-W., & Park, J. H. (2020). Learning robot manipulation based on modular reward shaping. In *2020 International Conference on Information and Communication Technology Convergence*, 883–886. https://doi.org/10.1109/ICTC49870.2020.9289596

[26] Manoj, M., Ananthakrishnan, S., Sriharshitha, P., & Pandi, V. R. (2021). Four axis welding robot control using fuzzy logic. In *2021 2nd Global Conference for Advancement in Technology*, 1–5. https://doi.org/10.1109/GCAT52182.2021.9587770

[27] Mohan, S., & Bhanot, S. (2007). Comparative study of some new hybrid fuzzy algorithms for manipulator control. *Journal of Control Science and Engineering*, *2007*(1), 075653. https://doi.org/10.1155/2007/75653

[28] Danthala, S., Rao, S., Mannepalli, K., & Shilpa, D. (2018). Robotic manipulator control by using machine learning algorithms: A review. *International Journal of Mechanical and Production Engineering Research and Development*, *8*(5), 305–310. https://doi.org/10.24247/ijmperdoct201834

[29] Li, Z., & Li, S. (2023). Neural network model-based control for manipulator: An autoencoder perspective. *IEEE Transactions on Neural Networks and Learning Systems*, *34*(6), 2854–2868. https://doi.org/10.1109/TNNLS.2021.3109953

[30] Jagadish, G., Abhishek, S., Prakash, A., & Nair, A. R. (2024). Enhancing recycling efficiency: A SCARA robot and CNN model for waste segregation. *International Journal of Advanced Mechatronic Systems*, *11*(3), 145–153. https://doi.org/10.1504/IJAMECHS.2024.143147

[31] Ladosz, P., Weng, L., Kim, M., & Oh, H. (2022). Exploration in deep reinforcement learning: A survey. *Information Fusion*, *85*, 1–22. https://doi.org/10.1016/j.inffus.2022.03.003

[32] Kumaar, A. A. N., & Kochuvila, S. (2023). Mobile service robot path planning using deep reinforcement learning. *IEEE Access*, *11*, 100083–100096. https://doi.org/10.1109/ACCESS.2023.3311519

[33] S, S., & Kumaar, A. A. N. (2024). Manipulator control using federated deep reinforcement learning. In *2024 IEEE International Conference on Electronics, Computing and Communication Technologies*, 1–6. https://doi.org/10.1109/CONECCT62155.2024.10677205

[34] Aradi, S. (2022). Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *23*(2), 740–759. https://doi.org/10.1109/TITS.2020.3024655

[35] Bourdillon, A. T., Garg, A., Wang, H., Woo, Y. J., Pavone, M., & Boyd, J. (2023). Integration of reinforcement learning in a virtual robotic surgical simulation. *Surgical Innovation*, *30*(1), 94–102. https://doi.org/10.1177/15533506221095298

[36] Karimi, M., & Ahmadi, M. (2022). A reinforcement learning approach in assignment of task priorities in kinematic control of redundant robots. *IEEE Robotics and Automation Letters*, *7*(2), 850–857. https://doi.org/10.1109/LRA.2021.3133934

[37] Zheng, L., Wang, Y., Yang, R., Wu, S., Guo, R., & Dong, E. (2023). An efficiently convergent deep reinforcement learning-based trajectory planning method for manipulators in dynamic environments. *Journal of Intelligent & Robotic Systems*, *107*(4), 50. https://doi.org/10.1007/s10846-023-01822-5

[38] Alessi, C., Bianchi, D., Stano, G., Cianchetti, M., & Falotico, E. (2024). Pushing with soft robotic arms via deep reinforcement learning. *Advanced Intelligent Systems*, *6*(8), 2300899. https://doi.org/10.1002/aisy.202300899

[39] He, W., Gao, H., Zhou, C., Yang, C., & Li, Z. (2021). Reinforcement learning control of a flexible two-link manipulator: An experimental investigation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *51*(12), 7326–7336. https://doi.org/10.1109/TSMC.2020.2975232

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

[41] Dankwa, S., & Zheng, W. (2020). Twin-delayed DDPG: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, 1–5. https://doi.org/10.1145/3387168.3387199

How to Cite: Sankaralingam, S., Arulmani Angamuthu, N. K., & Palaniswamy, S. (2026). Manipulator Control Using DRL: Sim-to-Real Validation on a 3-DoF Arm. *Journal of Computational and Cognitive Engineering*. https://doi.org/10.47852/bonviewJCCE62027656