JJEE

Volume 12 | Number 1 | March 2026 | Pages 1-18

Jordan Journal of Electrical Engineering

ISSN (print): 2409-9600, ISSN (online): 2409-9619

Homepage: jjee.ttu.edu.jo

# Evaluating the Performance and Robustness of Inverse Kinematics Problem for a 7-DoF Cobot Arm: A Systematic Comparison Using Multiple DRL Algorithms

**Dhaval R. Vyas[1], Parth S. Thakar[2], Anilkumar Markana[3*], Nitin Padhiyar[4]**

[1, 3] Electrical Engineering Department, Pandit Deendayal Energy University, Gandhinagar, India.
[*]E-mail: anil.markana@spt.pdpu.ac.in
[2] Electronics & Communication Engineering Department, Pandit Deendayal Energy University, Gandhinagar, India.
[4] Chemical Engineering Department, Indian Institute of Technology, Gandhinagar, India.

*Received: Nov 02, 2025* *Revised: Jan 06, 2026* *Accepted: Jan 11, 2026* *Available online: Mar 19, 2026*

***Abstract*—** Classical methods for solving Inverse Kinematics (IK) problems result in non-unique solutions, especially for higher degrees of freedom robotic arm. Also, these methods are computationally intensive and non-trivial. To circumvent this, Deep Reinforcement Learning (DRL) methods have shown potential to address these challenges in the recent literature. Thus, in this work, we undertake a DRL approach for a representative 7-DoF Panda Franka Emika robot. Moreover, the literature misses out on a comprehensive study of multiple DRL methods for examining the key performance parameters of the IK problem for a 7-DoF cobot. The performance of three popular DRL algorithms- Proximal Policy Optimization (PPO), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Deep Deterministic Policy Gradient (DDPG) is performed for this purpose. We evaluate their performance using key indicators such as accuracy in terms of average position error, success rate, and convergence time. Moreover, all these methods are tested against the parametric uncertainties applied in multiple links for the IK problem, inferring its practicality and robustness in real-world scenarios. In due course, the training framework is also proposed through which a particular DRL algorithm is trained with reward logs that are used to evaluate the training stability and learning progress. Finally, for each DRL algorithm, we remark on a few quantitative metrics that suggest their selection guidelines for a given scenario in specific IK problem.

***Keywords*—** Inverse Kinematics; Deep Reinforcement Learning; PPO; TD3; DDPG; Robotic Manipulation.

## 1. INTRODUCTION

A robotic arm can be defined as multiple rigid bodies interconnected through revolute or prismatic joints, depending upon the degree of freedom (DoF) [1]. It is controlled directly by the operator or logical controllers. Generally, in industries like manufacturing, military, agriculture, health, and consumer services, a robotic manipulator is used in application like pick and place. One of the main challenges in robotic manipulation is to derive a solution for the inverse kinematics problem: to determine the robotic manipulator joint parameters to achieve the desired position of its end effector in a 3-D workspace. This problem has a unique solution, multiple solutions, or no solution at all [2]. Moreover, some solution may reach to singularity issue, which leads to operational problem for robotic hardware. Complexity will further increase with the increase in degrees of freedom. Also, the IK problem is more complicated when there are obstacles in the path of movement of the robot. So it is quite

* Corresponding author
DOI: https://doi.org/10.5455/jjee.204-1762063106

difficult to derive the joint parameters of the robotic manipulator to make its end effector reach to desired location from the current location in a 3-D workspace with constraints like obstacles, an increased number of DoF, and singularity issues. The need for higher DoF robotics arms is increasing due to their flexibility and ability to replicate human-like movements. Also, during co-operation with humans, higher DoF cobots are a more popular choice for industries. These all scenarios show the need to derive an effective and accurate solution for the inverse kinematics problem. This method should work for robots with any number of degrees of freedom and should handle applied constraints. This motivates the development of an inverse kinematics solution that uses Deep Reinforcement Learning (DRL). This method is well-known for solving complex problems through proper training under a controlled environment.

Conventional methods for solving IK problems usually rely on geometric, algebraic, or numerical approaches. These methods can be costly in terms of computation and may not always work well for real-time applications such as shown in [3, 4]. Damped least square (DLS) based IK solution proposed by [5] possesses the problem of singularity in solutions for some poses. Also, DLS doesn't guarantee a unique solution for the desired pose of the end effector. Few meta-heuristic approaches like genetic algorithm [6, 7] and particle swarm optimization [8], are proposed for inverse kinematics solution of robotic manipulator. These methods have problems like a large computational cost and are not suitable for highly dynamic environment like the application of cobots.

A neural network-based approach [9] is adopted to find the IK solution based on the trained neural network. Application of a recurrent neural network to find IK solution for a robotic manipulator is reviewed by [10]. Recently, researchers proposed a method that enables multimodality in a neural network model trained to predict the joint angles in the constrained workspace [11]. The problem with such neural network approaches in deriving the inverse kinematics solution requires a large amount of data to train the neural network. Also, if the data is not filtered well, some singularity points in the data can lead the network to learn from incorrect information. Therefore, there is a need for a method which is data independent and that can train the network from scratch with ability to perform in a dynamic environment. Also, the simulation environment should be such that it can handle parametric uncertainty during the training process.

Recently, DRL has shown promising results for addressing complex control problems [12, 13] in various fields. It teaches optimal strategies by interacting with the environment in simulations. DRL methods offers a robust data driven approach for the solution of the IK problem. Recent advancements in physics-based simulation engines offer a stable environment to implement DRL algorithms for policy training, which provides real world experiences to the system. Frameworks like PyBullet [14, 15] and Isaac Sim [16] offer robust environments for training models to learn policies for solving the inverse kinematics of higher DoF robotic arms.

On a Sawyer robotic arm, IK solution is obtained using Deep Q-Networks (DQN) in [17]. It shows that DQN based IK solution can generate multiple joint-space trajectories for the end-effector. Zhao et al. in [18] introduced a learning framework, based on multi-agent PPO, to consider the collision problem for the end effector. Recently, a hybrid model with Multi-step PPO-DLS approach for the IK problem in a 7-DoF robotic arm was discussed in [19]. Although, all these approaches achieve some of the key performance indicators like accuracy, convergence speed, and stability; however, a systematic comparison of the various DRL agents for such a

higher DoF robotic system was missing in these studies. Additionally, they did not evaluate the performance against parametric uncertainties. This motivates us to address these problems.

For 7-DoF Panda Franka Emika cobot arm, the proposed work evaluates three well known DRL algorithms, PPO, TD3, and DDPG for the inverse kinematics solution. The environment is created using the Gym library in Python. Using the stable baseline3 library, all three algorithms are trained for the IK solution. In the subsequent sections, we formulate the problem clearly and approach for the DRL based training methods. Before that, we crisply outline the key contributions of this work and define the paper structure.

### 1.1. Key Contributions

The following are three key contributions of the paper.

- **Comparison of DRL methods for 7-DoF Panda cobot arm:** This paper extensively evaluates three main DRL algorithms, namely PPO, TD3, and DDPG. The comparison is carried out for solving the inverse kinematics of the 7-DoF Franka Emika Panda arm, using key performance indicators like accuracy in terms of average position error, success rate, and convergence time.
- **Robustness evaluation:** All three methods of DRL are tested against the parametric uncertainties applied in multiple links for the IK problem and evaluated on the basis of mean error and success rate.
- **Training framework:** We judicially articulate the training framework, through which a particular DRL algorithm is trained with algorithm-specific training parameters. Further, if a pre-trained model is available, we start the training with the pre-trained model; otherwise, training begins from scratch. Both culminate into a trained agent with reward logs that are used to evaluate the training stability and learning progress.

### 1.2. Paper Structure

The remainder of the paper is organized as follows. Section 2 defines the IK problem as a continuous control task to comply with the DRL algorithm structure. Section 3 then briefly illustrates the three DRL algorithms-PPO, TD3, and DDPG. Further, section 4 explains the common training procedure along with the details of the training parameters chosen for each algorithm. The performance evaluation and robustness are compared in section 5. Finally, section 6 concludes the findings with clear guidelines on choosing the DRL agents for IK problem.

## 2. PROBLEM FORMULATION

In this study, the inverse kinematics (IK) problem of a 7-DoF Franka Emika Panda robotic arm, as shown in Fig. 1, is addressed. The problem is considered as a continuous control task using deep reinforcement learning (DRL).

The objective is to compute the appropriate joint angles for all seven joints that allow the end-effector to reach a randomly generated target position in 3D space. Unlike traditional conventional methods, DRL methods learn a policy that directly maps observed states to planning command actions in an end-to-end manner [12].

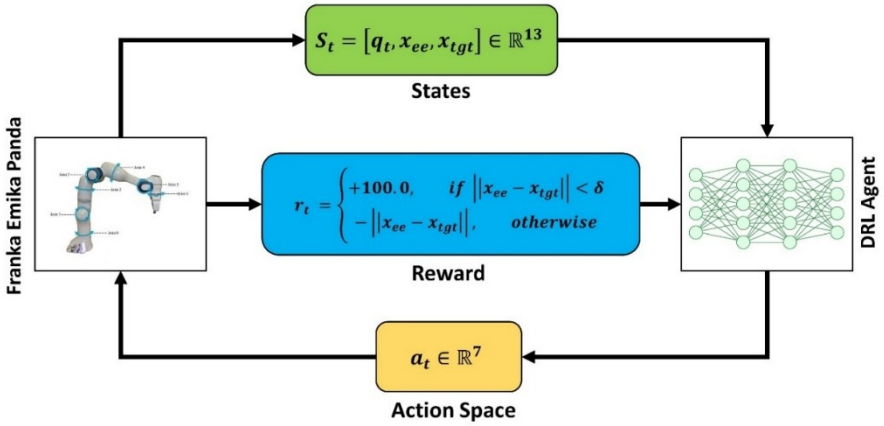


Fig. 1. Inverse Kinematics problem setup for the 7-DoF Franka Emika Panda robot arm.

Let $\boldsymbol{q}_t \in \mathbb{R}^7$ denote the current joint configuration of the robot and let $x_{tgt} \in \mathbb{R}^3$ be the target end-effector position, sampled randomly within a reachable workspace. The current end-effector position, $x_{ee} \in \mathbb{R}^3$ is a function of the current joint angles $\boldsymbol{q}_t$. The inverse kinematics problem seeks a joint configuration $\boldsymbol{q}$ such that the end-effector's position $x_{ee}$ is as close as possible to the target position $x_{tgt}$.

Let's define a state vector, $\boldsymbol{s}_t$ as follows,

$$\boldsymbol{s}_t = [\boldsymbol{q}_t, x_{ee}, x_{tgt}] \in \mathbb{R}^{13} \quad (1)$$

The above state vector Eq. (1) includes current joint states, $\boldsymbol{q}_t \in \mathbb{R}^7$, the current end effector position, $x_{ee} \in \mathbb{R}^3$, and the target end effector position $x_{tgt} \in \mathbb{R}^3$. This composite state vector gives all the necessary information to the DRL agent to predict the next action state $\boldsymbol{a}_t$.

The process begins with the observation of state $\boldsymbol{s}_t$ for each time step. Based on these observed states, the policy determines the action vector $\boldsymbol{a}_t$, which specifies the incremental adjustments to the seven joints of the robot. This action is then executed by Franka Emika Panda arm, resulting in new states. The effectiveness of the predicted actions is evaluated using the reward function $r_t$ as specified below,

$$r_t = \begin{cases} +100.0, & if \ \|x_{ee} - x_{tgt}\| < \delta \\ -\|x_{ee} - x_{tgt}\|, & otherwise \end{cases} \quad (2)$$

The term, $\|x_{ee} - x_{tgt}\|$ represents the Euclidean distance between the end-effector and the target. In 3D space, the Euclidean distance is computed as,

$$\|x_{ee} - x_{tgt}\| = \sqrt{(X_{ee} - X_{tgt})^2 + (Y_{ee} - Y_{tgt})^2 + (Z_{ee} - Z_{tgt})^2} \quad (3)$$

The agent receives a positive reward if the Euclidean distance between the end-effector and the target is less than the threshold value $\delta$. Otherwise, a negative reward is assigned based on the magnitude of the position error. This would ultimately encourage the agent to minimize the distance between the end effector and the target. This iterative process of state observation, action prediction, and reward evaluation allows the DRL agent to learn an end-to-end policy for solving the complex and highly nonlinear IK problem. An episode is terminated if the Euclidean distance between the end-effector and the target is less than the defined threshold limit of $\delta$ or the agent reaches the maximum number of steps per episode.

## 3. DRL ALGORITHMS

This section presents a brief overview of the Deep Reinforcement Learning (DRL) algorithms employed to solve the inverse kinematics problem for the 7-DoF Panda robot arm.

The algorithms discussed include PPO, TD3, and DDPG. Additionally, pseudocode for each algorithm is presented to facilitate easier understanding for the reader.

### 3.1. PPO

In this optimization method, the agent is trained by optimizing policy functions using reinforcement learning. These policy functions are critical in decision making. The main attribute of the PPO algorithm is to maintain stability during policy exploration via incorporating a clipped objective function. This would prevent noticeable deviations in policy updates. For example, in a robotic application, accurate movement of the robotic arm and maintaining the stability of the entire assembly is crucial and essential.

The process of optimization begins with initialization of policy parameters $\theta$ and value function parameters $\phi$. In each iteration using current policy action $a_t = \pi\theta_{old}$, the agent interacts with the environment for a number of time steps $T$. During this interaction, agent collects batch of data states, $s_{t+1}$, and rewards $r_t$. These data are then used to compute the rewards-to-go $\hat{R}_t$, the measure of total discounted future rewards, and advantage estimates $\hat{A}_t$, which is a measure of how much better an action is compared to expectation. Using the collected data, it runs several epochs to update both the policy and the value function. The policy updates through a clipped surrogate objective function, which keeps the new policy from straying too far from the old policy. Simultaneously, the value function is updated by minimizing the mean-squared error between its predictions and the computed rewards-to-go, which helps the critic to accurately estimate the states. Therefore, this iterative process allows the PPO algorithm to achieve robust and cautious policy refinement. Algorithm 1 illustrates the standard procedure for proximal policy optimization.

**Algorithm 1. Proximal Policy Optimization (PPO) [20]**

Initialize policy parameters $\theta_0$, value function parameters $\phi_0$, Total number of timesteps $T_{st}$, Number of timesteps in each episode $T$, No. of epochs $K$, $\gamma$, and $\lambda$.
**for** iteration = 1: $\frac{T_{st}}{T}$ **do**
  Collect data using policy $\pi\theta_{old}$ for T timesteps:
  **for** 1: $T$ **do**
    Observe state $s_t = [q_t, x_{ee}, x_{tgt}]$
    Select action $a_t = \pi\theta_{old}(s_t)$
    Execute $a_t$, observe new state $s_{t+1}$ and reward $r_t$
    Store $(s_t, a_t, r_t, s_{t+1})$
  Compute rewards-to-go $\widehat{R} = \sum_{l=0}^{T-t-1} \gamma^l r_{t+1}$
  Compute advantage estimates: $\widehat{A} = \sum_{t=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}$, where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
  **for** epoch = 1,2, … , $K$ **do**
    **for** mini-batch of collected data **do**
      Compute ratio: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi\theta_{old}(a_t|s_t)}$
      Compute surrogate loss:
      $L^{clip}(\theta) = \mathbb{E}[\min(r_t(\theta)\hat{A}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$
      Update $\theta$ by maximizing $L^{clip}(\theta)$
      Update value function $\phi$ by minimizing mean-squared error:
      $$\left(\widehat{R}_t - V_\phi(s_t)\right)^2$$
  Set $\theta_{old} \leftarrow \theta$

For further details on the theoretical foundations of PPO, readers are encouraged to refer work by Schulman et al. (2017) [20].

### 3.2. TD3

Algorithm 2 refers to the standard TD3 procedure with an advance actor-critic method designed to address the continuous control task. The TD3 framework features both actor $\mu_\theta$ and critics $(Q_{\phi_1}, Q_{\phi_2})$ neural networks, initialized with random parameters for effective exploration at the start of training.

**Algorithm 2. Twin Delayed Deep Deterministic Policy Gradient (TD3) [21], [22]**

1: Initialize two critic networks as $Q_{\phi_1}$ and $Q_{\phi_2}$, actor network $\mu_\theta$, target network weights $\phi_1, \phi_2$ *and* $\theta$, the discount factor $\gamma \in (0,1)$, delay parameter $d = 2$, replay buffer $\mathcal{B}$, Total number of timesteps $T_{st}$, Number of timesteps in each episode $T$.

2: **for** iteration = 1: $\frac{T_{st}}{T}$ **do**

3: Select action with Gaussian perturbed noise $a_t \sim \mu_\theta(s_t) + \epsilon, \epsilon \sim \mathcal{N}_t(0, \sigma)$

4: Observe $r_t$ and $s_{t+1}$

5: Store $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{B}$

6: Randomly sample mini-batch of $N$ transitions $(s, a, r, s')$ from reply buffer $\mathcal{B}$

7: Add clipped exploration noise to the target policy's action as :

$$\tilde{a} \leftarrow \mu_\theta(s) + \epsilon, \epsilon \sim clip(\mathcal{N}_t(0, \tilde{\sigma}), -c, c)$$

8: Obtained the target $Q$ value, $y \leftarrow r + \gamma \min_{i=1,2} Q_{\phi'}(s', \mu_{\theta'}(s'))$

9: Train and update critics $\phi_i \leftarrow argmin_{\phi_i} \frac{1}{N} \sum \left(y - Q_{\phi_i}(s, a)\right)^2$ for $i = 1,2$

10: **if** d=0 **then**

11: Update $\theta$ by the deterministic policy gradient:

$$\nabla_\theta J(\mu_\theta) = \frac{1}{N} \sum \nabla_a Q_{\phi_1}(s, a)||_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)$$

12: Update target networks:

$$\phi_i' \leftarrow \tau\phi_i + (1-\tau)\phi_i' \text{ for } i = 1,2$$

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

In each episode, TD3 proceeds with the current observed state and selects action $a_t$ based on the current state augmented with exploration noise $\mathcal{N}$. Once this action is executed, received rewards and observed states are stored in the replay buffer $\mathcal{B}$. TD3 samples mini batches from the replay buffer to update its model to improve data efficiency. The clipped noise function $clip(\epsilon, -c, c)$ is used to improve exploration and to mitigate overestimation bias to compute the target action $\tilde{a}$. This action is used to compute the target Q-value $y$ with the twin critic mechanism, which helps to reduce the overestimation. These critics are updated by minimizing the mean squared error between their Q-value predictions. The actor network is updated by maximizing the expected Q-value of the first critic. Finally, updates are applied to target networks, moving them slowly towards convergence.

### 3.3. DDPG

Algorithm 3 illustrates the Deep Deterministic Policy Gradient (DDPG). It is a lighter version of TD3 with one critic network. Actor network $\mu_\theta$ and critic network $Q_\phi$ are initialized

with random parameters. DDPG observed initial state $s_0$ and action $a_t$ with augmented exploration noise $\mathcal{N}_t$ is selected. By executing action $a_t$, the network received a reward and observed the next state. Again mini-batch is sampled from the replay buffer $\mathcal{B}$ to compute the target Q-value $y_i$. The critic will be updated by minimizing the mean squared error. Also, the actor policy is updated by maximizing the prediction of the Q-value. Finally, updates are applied to both the actor and critic networks.

**Algorithm 3. Deep Deterministic Policy Gradient (DDPG) [23]**

1: Initialize actor network $\mu_\theta$ and critic network as $Q_\phi$ with random parameters,
target network weights $\theta' and\ \phi'$, the discount factor $\gamma \in (0,1)$, replay buffer $\mathcal{B}$ ,
Total number of timestep $T_{st}$, Number of timestep in each episode $T$ .

2: **for** iteration = 1: $\frac{T_{st}}{T}$ **do**

3: Initialize a random process $\mathcal{N}_t$ for exploration

4: Receive initial state $s_0$

5: **for** each step t **do**

6: Select action $a_t = \mu_\theta(s_t) + \mathcal{N}_t$

7: Execute $a_t$, observe reward $r_t$ and next state $s_{t+1}$

8: Store $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{B}$

9: Randomly sample mini-batch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from reply buffer $\mathcal{B}$

10: Compute target Q-value :

$$y_i = r_i + \gamma Q_{\phi'}(s_{i+1}, \mu_{\theta'}(s_{i+1}))$$

11: Update critic by minimizing:

$$\mathcal{L}(\phi) = \frac{1}{N}\sum_i \left(Q_{phi}(s_i, a_i) - y_i\right)^2$$

12: Update actor policy by maximizing:

$$\mathcal{J}(\theta) = \frac{1}{N}\sum_i Q_\phi(s_i, \mu_\theta(s_i))$$

13: Update target networks:

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$
$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

## 4. REINFORCEMENT LEARNING TRAINIG WORKFLOW

In this section, we present a generalized approach for network training in Reinforcement Learning. The proposed framework is designed to support all three DRL algorithms discussed in the previous section. The flowchart given by Fig. 2 outlines the unified training pipeline developed for training continuous control policies of the 7-DoF Franka Emika Panda manipulator within a simulated environment. These algorithms are implemented using the Stable-Baseline3 (SB3) library integrated with the *pybullet* physics engine in Python. The framework ensures consistent environmental conditions to facilitate fair comparison among the algorithms.

The training begins with the initialization of a custom environment $E$, consists of Panda 7-DoF robotic arm on a plane with all seven joints at their home position. To optimize computational efficiency of GPU simulation rendering kept off during training. All three DRL algorithm training were carried out on a computational system equipped with an Intel 13th generation i7-13700HX processor, 48 GB of RAM, and an NVIDIA RTX 4060 GPU with 8 GB of

dedicated VRAM. To comply with the *OpenAI Gym*, the environment is wrapped with GymWrapper, which ensures compatibility with multiple DRL algorithms.

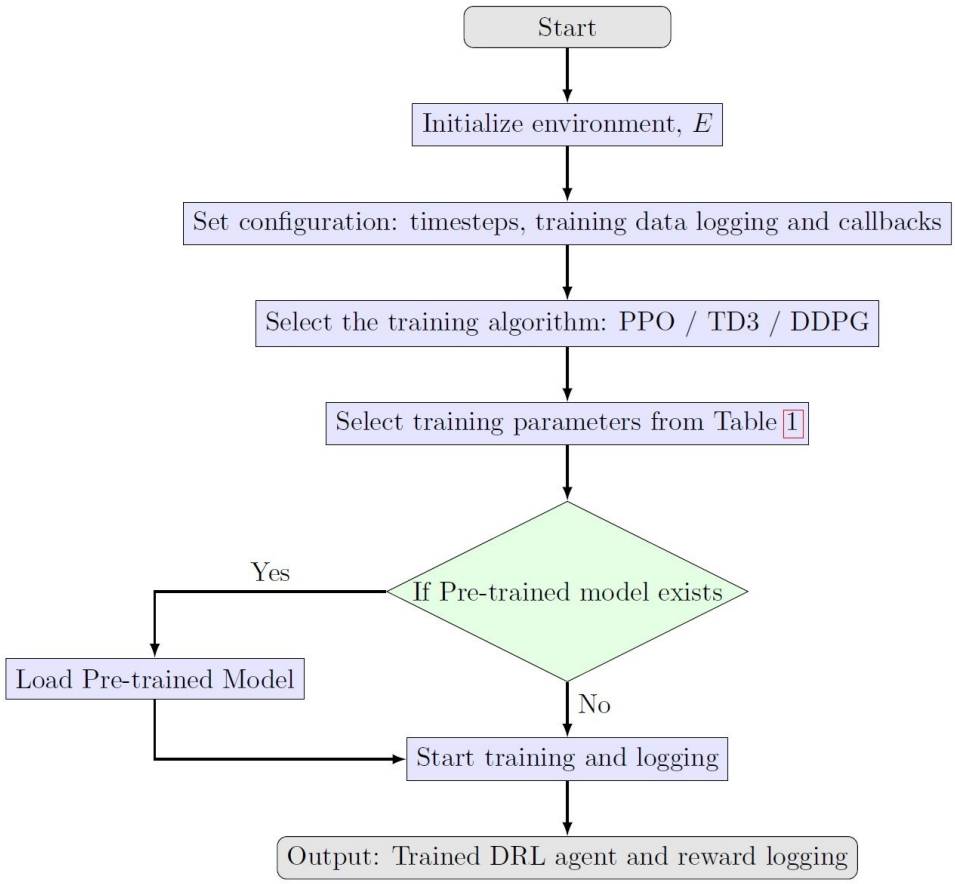


Fig. 2. Training framework.

Table 1. Training parameters for PPO, TD3 and DDPG algorithms.

| Parameter | PPO | TD3 | DDPG |
|---|---|---|---|
| Learning rate, $\alpha$ | 0.0003 | 0.001 | 0.001 |
| Batch size | 64 | 256 | 100 |
| No. of epochs, $K$ | 10 | - | - |
| Discount factor, $\gamma$ | 0.99 | 0.99 | 0.98 |
| Generalized advantage estimation, $\lambda$ | 0.95 | - | - |
| Clipping parameter, $\epsilon$ | 0.2 | - | - |
| Buffer size | - | 1000000 | 1000000 |
| Update rate, $\tau$ | - | 0.005 | 0.005 |
| Learning starts, $N_{starts}$ | - | 100 | 100 |
| Exploration noise, $\mathcal{N}_t$ | - | Normal(0,0.1) | Normal(0,0.1) |
| Network Architecture | [64,64] | [400,300] | [400,300] |
| Total timestep, $T_{st}$ | 2000000 | 3500000 | 2000000 |
| $T$ | 100 | 100 | 100 |

Algorithm-specific training parameters are usually selected based on observed convergence behavior in the environment. Empirical tuning, along with insights from existing literature and best practices, is commonly employed to determine these settings. Although there is no universal rule, people usually take a systematic approach. In this approach,

parameters like network architecture, learning rates, buffer sizes, batch sizes, discount factors, update rates, and exploration noise are often selected based on standard benchmarks or default values from DRL libraries such as Stable Baselines3. For more details on choosing training parameters, readers can look at the discussion in [24]. While in this work, we selected training parameter values as listed in Table 1. For PPO and DDPG, training occurs over a total of 2000000 timesteps, while TD3 is trained for 3500000 timesteps because it has a slower learning curve. These considerations directly motivate the algorithm-specific parameter choices summarized in Table 1. Each episode's reward is recorded to review the training behavior of each algorithm. Before training begins, the system checks for an existing trained model. If one is found, it is loaded for continued training or evaluation. Finally, upon completion of training, the trained policy is saved on a local resource for further evaluation. This modular training workflow allows training IK solution policies using multiple DRL algorithms.

Fig. 3(a-c) depict the training progress for PPO, TD3, and DDPG agents, respectively. All three deep reinforcement learning algorithms show an overall upward trend in reward as training progresses, indicating that the agent is learning to solve the inverse kinematic problem.

The PPO training progress graph, as shown in Fig. 3(a), indicates a rapid and stable learning curve. As shown in the figure, initially, the episode rewards within the light blue shaded area are quite broad, reflecting high variance in performance during the early exploration phase. However, as the algorithm converges to a successful policy, this shaded area narrows considerably, indicating more consistent performance. For better visualization, we also plot the 100-episode average reward curves for all three agents together in a single graph as shown in Fig. 4. It can be observed from Fig. 4 that the 100-episode average reward curve for PPO increases rapidly from a negative value to a high positive reward within the first 5,000 episodes and then remains consistently at a high level. The TD3 training progress graph, depicted in Fig. 3(b), exhibits a similar learning trend to that of the PPO algorithm. However, the convergence period for TD3 is considerably longer, requiring approximately 10,000 episodes, compared to around 5,000 episodes for PPO. The DDPG training progress is shown in Fig. 3(c). During the initial 9,000 episodes, most of the episode rewards are negative, indicating suboptimal policy exploration during this early phase. Subsequently, from around 9,000 episodes onward, there is a noticeable increase in episode rewards, resulting in a positive cumulative reward. The 100-episode average reward curve for DDPG rises after 9,000 episodes, transitioning from negative to high positive values, and stabilizes after approximately 17,000 episodes. In summary, regarding the training progress of all three algorithms, the PPO algorithm achieves convergence to the maximum reward more quickly than the other two. Among them, DDPG is the slowest in reaching maximum reward. This is clearly illustrated in Fig. 4, which displays the 100-episode moving average rewards.

Fig. 5 shows a comparative study of PPO, TD3, and DDPG trained agents, focusing on stability, learning dynamics, and sensitivity over a 100-episode window. The standard deviation of episodic rewards within this rolling window duration are shown in the left side subplot. This indicates the reward variance for each algorithm as the training progresses. During the early stage of the training, all algorithms exhibit high reward variance. This indicates suboptimal policy exploration and instability. The reward variance for the PPO algorithm decreases rapidly and remains at a low level. This indicates faster convergence and greater stability. In contrast, DDPG maintains a high level of reward variance throughout the first approximately 20,000 episodes. After which, we can observe that the reward variance

decreases. This indicates that the DDPG agent experiences a delay in variance decay, indicating potential stability issues during training. The right-side subplot depicts the reward difference between consecutive 100-episode windows throughout training. Further, this plot indicates the vulnerability and volatility in the agent's learning process. Note that larger fluctuations indicate instability in policy updates and increased sensitivity to feedback from the environment.

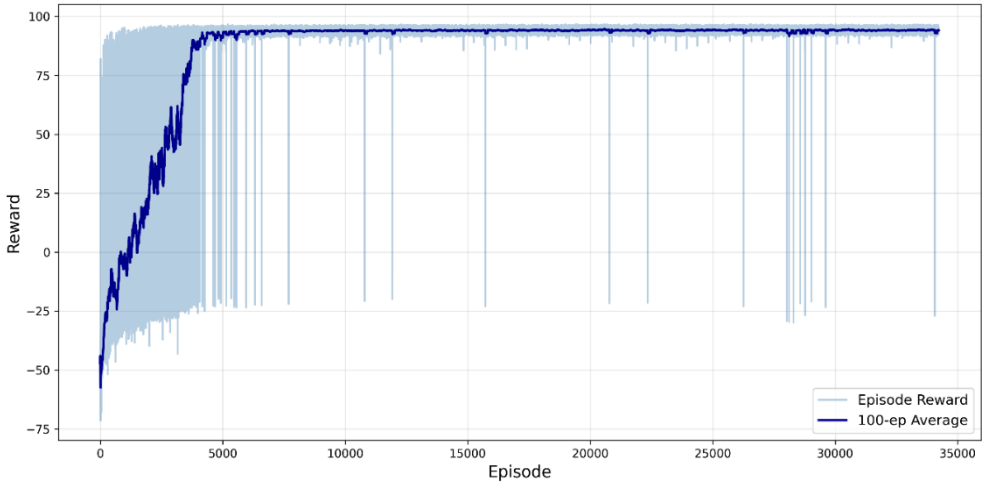


Fig. 3. **(a)** PPO training progress.

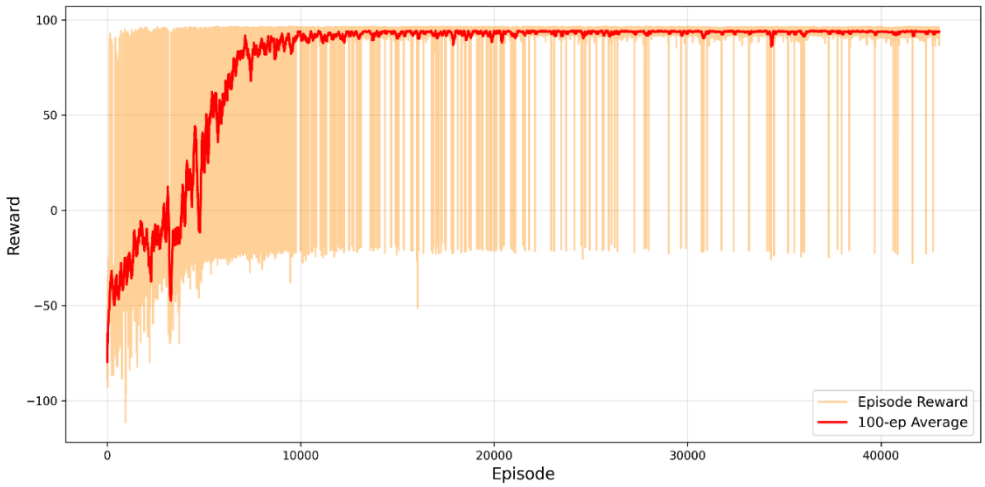


Fig. 3. **(b)** TD3 training progress.

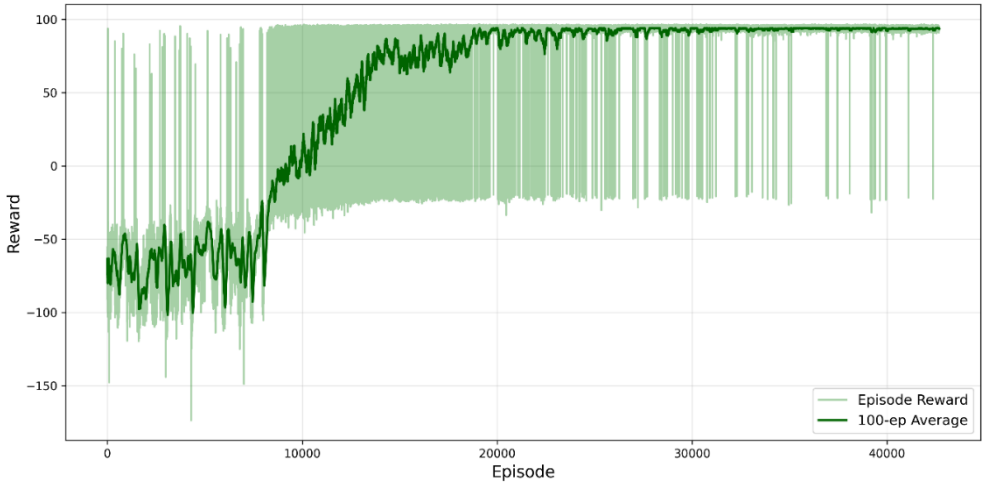


Fig. 3. **(c)** DDPG training progress.

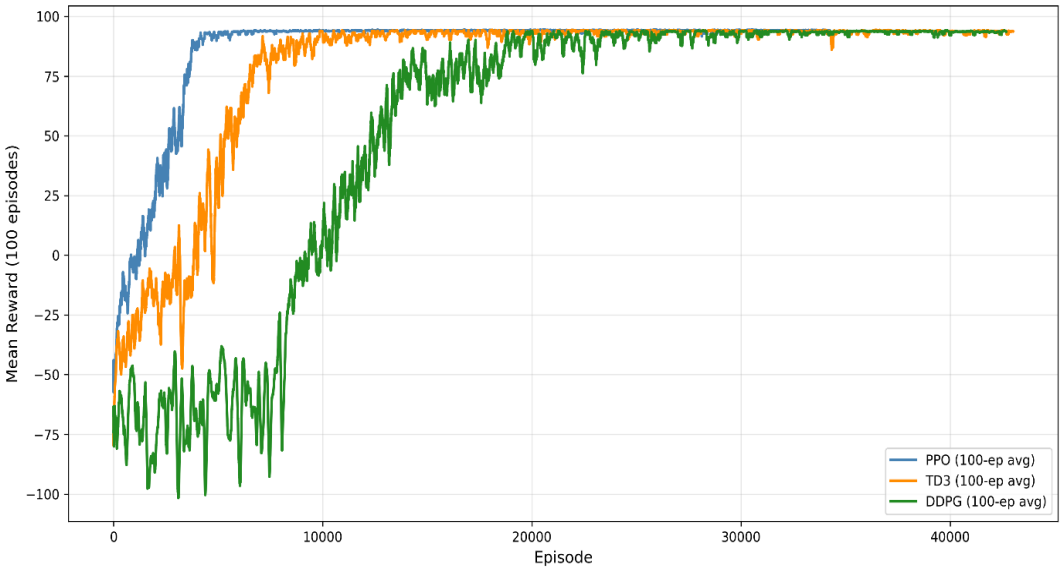


Fig. 4. Comparison of 100-episode average reward for training agents PPO, TD3, and DDPG.

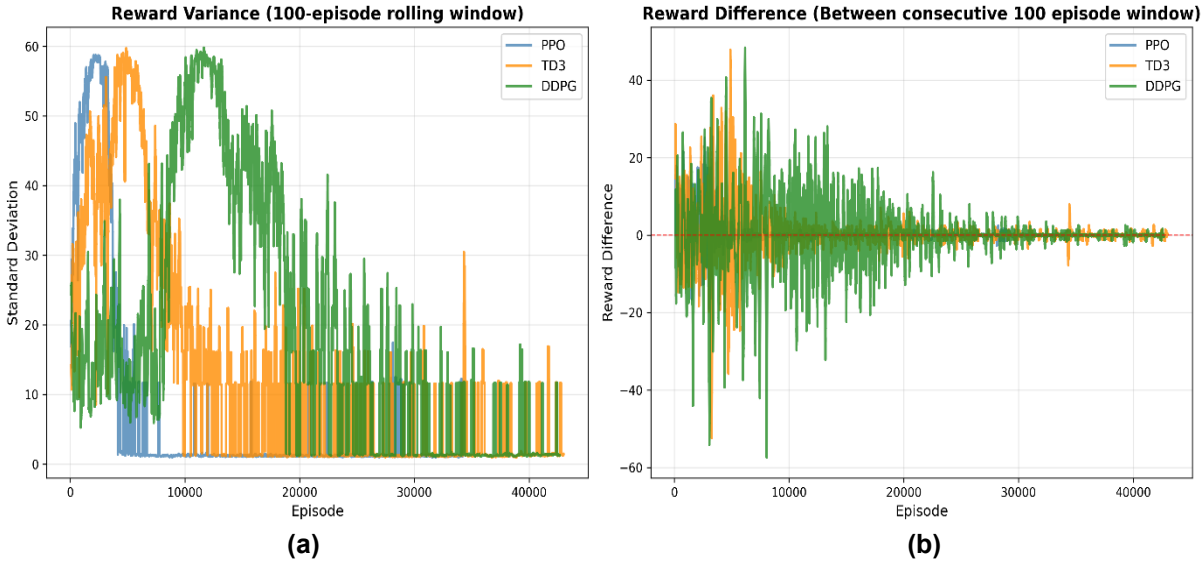


Fig. 5. Comparative analysis of the training agents PPO, TD3, and DDPG, focusing on stability, learning dynamics and sensitivity.

## 5. PERFORMANCE EVALUATION OF TRAINED DRL-AGENTS

In this section, we describe the performance evaluation of all three trained DRL agents, discussed in the previous section, for solving IK. Subsequently, we assess their robustness against parametric uncertainties.

To simulate the practical scenarios, parametric variations are introduced during the evaluation in the simulation environment, allowing for a comprehensive analysis of each agent's effectiveness and stability under uncertain conditions.

Simulations were conducted to evaluate the performance of all three trained Deep Reinforcement Learning (DRL) agents within the simulation environment for the Panda 7-DoF robotic arm, as illustrated in Fig. 6.

Throughout all experimental trials, constraints are applied to joint limits. These limits and the reference home positions for all seven joints are listed in Table 2.

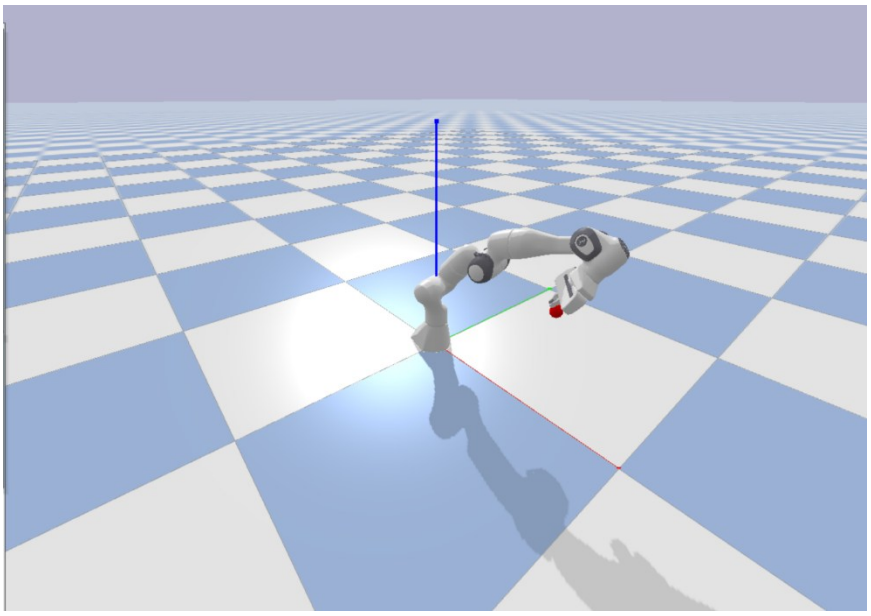

Fig. 6 Franka Emika Panda 7-DoF simulation environment.

Table 2. Joint limits and origin positions of franka emika panda 7-DoF.

| Joint Index | Lower Limit | Upper Limit | Posistion |
|---|---|---|---|
| 1 | -2.9671 | 2.9671 | (0.0,0.0,0.333) |
| 2 | -1.8326 | 1.8326 | (0.0,0.0,0.0) |
| 3 | -2.9671 | 2.9671 | (0.0,-0.316,0.0) |
| 4 | -3.1416 | 0.0873 | (0.0825,0.0,0.0) |
| 5 | -2.9671 | 2.9671 | (-0.0825,0.384,0.0) |
| 6 | -0.0873 | 3.8223 | (0.0,0.0,0.0) |
| 7 | -2.9671 | 2.9671 | (0.088,0.0,0.0) |

### 5.1. Performance evaluation based on success rate and position error

In this subsection, we present a comprehensive analysis of a trained DRL agent based on success rate and positional error as key performance parameters. The agent's consistency and accuracy in solving the inverse kinematics (IK) problem are evaluated using these parameters.

The performance evaluation process starts by defining the number of target positions, $N = 20$. These positions are points in 3-D space that the robotic end effector tries to reach. A threshold, $\delta = 0.03$ cm, is also set. This threshold is based on the Euclidean distance between the Panda arm's end effector and each target point. It helps to determine if the reach is successful. An experiment is considered successful when the Euclidean distance between the end effector and the target point is within the specified threshold of $\delta$. The trained agent model is loaded, along with pre-generated random target positions. These target positions are generated while keeping the constraints of the reachable workspace of the arm. Table 3 lists the 20 target positions in 3D space employed in this study. For each episode in the experiment, a specific target $x_{tgt}$ is designated as the goal position. After resetting the environment, the trained DRL agent predicts the action $a_t$ based on the current observed state $s_t$ . Using this action, the environment then returns to the next state, reward, and termination flag. The total reward, number of steps, and final positional error are recorded for further analysis. This procedure will be carried out for an evaluation count of $N = 20$, testing all 20 target positions.

Towards the end, the average positional error and overall success rate from all 20 episodes will be calculated to evaluate the effectiveness of the trained agent in solving the inverse kinematics (IK) problem for the 7-DoF arm.

Table [4-6] show the quantitative analysis of the IK solution for PPO, TD3, and DDPG DRL agents, respectively. Fig. 7 and Fig. 8 compare the end-effector position errors for the trained PPO, TD3, and DDPG agents.

Table 7 shows the performance of all three agents. It lists their average error and success rate percentages in solving the inverse kinematics (IK) problem for all 20 target positions. All agents achieved a 100% success rate, demonstrating their accuracy. The DDPG algorithm performs better than the others, with the lowest average error of 0.0198. PPO's performance is slightly lower than that of TD3 and DDPG.

In summary, for this performance evaluation, all three trained agents perform reliably within the threshold value of 0.03 cm.

To further investigate, we reduce the threshold limit to 0.02 cm. As a result, the success rates decreased to 45% for DDPG and 40% for both PPO and TD3 trained agents. Detailed results for this case are not included in this work.

The performance of all three trained agents regarding precision and accuracy in solving the IK problem is similar and remains comparable. DDPG shows a slight advantage over the other two algorithms.

Table 3. Target positions.

| Point | Target_x | Target_y | Target_z |
|---|---|---|---|
| 1 | 0.302 | 0.371 | 0.436 |
| 2 | 0.094 | -0.321 | 0.425 |
| 3 | 0.176 | -0.048 | 0.511 |
| 4 | 0.253 | 0.270 | -0.057 |
| 5 | 0.174 | 0.324 | 0.122 |
| 6 | 0.180 | -0.099 | 0.190 |
| 7 | -0.191 | 0.089 | 0.594 |
| 8 | -0.178 | -0.214 | 0.438 |
| 9 | 0.195 | 0.199 | 0.174 |
| 10 | 0.113 | -0.370 | 0.225 |
| 11 | -0.395 | 0.388 | 0.449 |
| 12 | 0.133 | -0.153 | 0.435 |
| 13 | 0.395 | -0.295 | -0.063 |
| 14 | 0.201 | -0.330 | 0.190 |
| 15 | -0.062 | -0.102 | 0.051 |
| 16 | -0.351 | 0.376 | 0.365 |
| 17 | -0.036 | 0.321 | 0.360 |
| 18 | 0.335 | -0.391 | 0.005 |
| 19 | 0.290 | 0.134 | 0.132 |
| 20 | 0.146 | -0.117 | 0.294 |

Table 4. IK solution for PPO.

| Point | x | y | z | Error | $\theta_1$ | $\theta_2$ | $\theta_3$ | $\theta_4$ | $\theta_5$ | $\theta_6$ | $\theta_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.301 | 0.372 | 0.465 | 0.029 | -1.253 | -0.112 | 2.347 | -2.376 | -2.807 | 0.670 | -0.625 |
| 2 | 0.094 | -0.321 | 0.448 | 0.022 | 0.039 | -1.001 | -1.408 | -2.722 | 1.906 | 1.204 | 0.509 |
| 3 | 0.176 | -0.048 | 0.533 | 0.022 | -1.093 | -1.703 | 1.102 | -2.982 | -2.025 | 1.889 | 1.211 |
| 4 | 0.253 | 0.270 | -0.039 | 0.018 | -2.807 | -1.578 | -2.792 | -1.532 | 1.233 | 0.375 | -2.151 |
| 5 | 0.174 | 0.325 | 0.139 | 0.017 | -2.503 | -1.017 | -2.807 | -2.067 | 1.313 | 0.318 | -2.503 |
| 6 | 0.179 | -0.098 | 0.215 | 0.025 | 0.207 | -1.703 | -2.619 | -2.795 | 0.514 | 1.619 | 0.080 |
| 7 | -0.192 | 0.090 | 0.619 | 0.025 | -1.465 | 1.823 | -2.262 | -2.829 | -2.218 | 2.006 | -1.664 |
| 8 | -0.179 | -0.214 | 0.457 | 0.019 | 1.438 | 0.555 | 2.555 | -2.982 | 2.627 | 0.641 | 1.967 |
| 9 | 0.194 | 0.200 | 0.202 | 0.028 | -1.124 | 1.823 | 0.841 | -2.585 | 0.834 | 1.508 | -1.522 |
| 10 | 0.112 | -0.369 | 0.250 | 0.025 | 0.512 | 1.823 | -0.984 | -2.399 | -0.966 | 1.528 | 2.241 |
| 11 | -0.395 | 0.388 | 0.457 | 0.009 | -2.807 | 0.639 | -0.945 | -2.075 | -2.333 | 0.885 | -1.792 |
| 12 | 0.132 | -0.153 | 0.464 | 0.029 | 0.170 | -1.703 | -1.370 | -2.982 | 1.755 | 1.858 | 0.081 |
| 13 | 0.395 | -0.294 | -0.044 | 0.019 | -0.601 | 1.697 | 0.099 | -1.378 | 2.953 | 0.128 | 0.538 |
| 14 | 0.200 | -0.329 | 0.220 | 0.030 | 0.539 | -1.703 | -2.040 | -2.347 | 1.102 | 1.467 | 0.235 |
| 15 | -0.062 | -0.101 | 0.074 | 0.023 | -2.807 | 1.407 | 0.257 | -2.320 | 0.368 | 0.721 | 0.035 |
| 16 | -0.352 | 0.376 | 0.375 | 0.010 | -2.807 | 0.834 | -0.903 | -2.154 | -1.981 | 0.821 | -1.527 |
| 17 | -0.037 | 0.321 | 0.373 | 0.013 | -2.436 | -0.139 | -2.208 | -2.788 | 2.379 | 0.287 | -2.202 |
| 18 | 0.334 | -0.390 | 0.033 | 0.029 | -0.058 | 1.665 | -0.474 | -1.466 | -1.439 | 0.654 | 1.836 |
| 19 | 0.289 | 0.135 | 0.158 | 0.026 | 1.157 | 1.080 | -0.203 | -2.209 | -0.926 | 0.383 | 1.325 |
| 20 | 0.145 | -0.116 | 0.322 | 0.028 | -0.584 | 0.425 | 0.090 | -2.683 | 0.088 | 0.087 | -2.380 |

Table 5. IK solution for TD3.

| Point | x | y | z | Error | $\theta_1$ | $\theta_2$ | $\theta_3$ | $\theta_4$ | $\theta_5$ | $\theta_6$ | $\theta_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.301 | 0.372 | 0.465 | 0.021 | 2.943 | -0.229 | -1.961 | -2.361 | 2.770 | 0.694 | -1.128 |
| 2 | 0.094 | -0.321 | 0.448 | 0.030 | -2.467 | -1.703 | 1.620 | -2.710 | -1.454 | 1.855 | 2.298 |
| 3 | 0.176 | -0.048 | 0.533 | 0.015 | -1.099 | -1.703 | 1.139 | -2.982 | -1.987 | 1.887 | 1.216 |
| 4 | 0.253 | 0.270 | -0.039 | 0.026 | -2.807 | -1.553 | -2.790 | -1.560 | 1.230 | 0.378 | -2.161 |
| 5 | 0.174 | 0.325 | 0.139 | 0.015 | -2.504 | -1.026 | -2.807 | -2.059 | 1.308 | 0.320 | -2.506 |
| 6 | 0.179 | -0.098 | 0.215 | 0.018 | -0.281 | 0.858 | 0.046 | -2.449 | -0.071 | 0.287 | -2.807 |
| 7 | -0.192 | 0.090 | 0.619 | 0.010 | -1.407 | 1.823 | -2.225 | -2.869 | -2.190 | 1.974 | -1.695 |
| 8 | -0.179 | -0.214 | 0.457 | 0.010 | 1.477 | 0.517 | 2.503 | -2.982 | 2.587 | 0.601 | 1.939 |
| 9 | 0.194 | 0.200 | 0.202 | 0.029 | 2.907 | 1.823 | -0.662 | -2.588 | -0.656 | 1.510 | 0.100 |
| 10 | 0.112 | -0.369 | 0.250 | 0.019 | 2.457 | -0.672 | 2.953 | -2.319 | -1.577 | 0.253 | 2.522 |
| 11 | -0.395 | 0.388 | 0.457 | 0.025 | -2.807 | 0.587 | -0.951 | -2.074 | -2.404 | 0.894 | -1.837 |
| 12 | 0.132 | -0.153 | 0.464 | 0.030 | 1.433 | 1.823 | -1.597 | -2.982 | -1.581 | 1.858 | 1.817 |
| 13 | 0.395 | -0.294 | -0.044 | 0.021 | -0.601 | 1.694 | 0.099 | -1.382 | 2.943 | 0.127 | 0.538 |
| 14 | 0.200 | -0.329 | 0.220 | 0.023 | -2.416 | -1.703 | 2.233 | -2.332 | -0.914 | 1.452 | 1.994 |
| 15 | -0.062 | -0.101 | 0.074 | 0.027 | 0.664 | 1.799 | -0.032 | -2.468 | -0.041 | 1.250 | 2.584 |
| 16 | -0.352 | 0.376 | 0.375 | 0.029 | -2.807 | 0.761 | -0.916 | -2.172 | -2.075 | 0.808 | -1.578 |
| 17 | -0.037 | 0.321 | 0.373 | 0.022 | -2.546 | -0.086 | -2.059 | -2.817 | 2.555 | 0.287 | -2.071 |
| 18 | 0.334 | -0.390 | 0.033 | 0.021 | -0.827 | 1.484 | 0.104 | -1.571 | 2.946 | 0.148 | 0.761 |
| 19 | 0.289 | 0.135 | 0.158 | 0.018 | 0.598 | 1.091 | 0.096 | -2.004 | -0.216 | 0.072 | 2.414 |
| 20 | 0.145 | -0.116 | 0.322 | 0.028 | 0.571 | -0.854 | -2.033 | -2.982 | 1.157 | 0.969 | -0.314 |

Table 6. IK solution for DDPG.

| Point | x | y | z | Error | $\theta_1$ | $\theta_2$ | $\theta_3$ | $\theta_4$ | $\theta_5$ | $\theta_6$ | $\theta_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.301 | 0.372 | 0.465 | 0.029 | 2.944 | -0.169 | -1.871 | -2.271 | 2.860 | 0.784 | -1.038 |
| 2 | 0.094 | -0.321 | 0.448 | 0.011 | -2.377 | -1.643 | 1.710 | -2.620 | -1.364 | 1.945 | 2.388 |
| 3 | 0.176 | -0.048 | 0.533 | 0.013 | -1.009 | -1.643 | 1.229 | -2.892 | -1.897 | 1.977 | 1.306 |
| 4 | 0.253 | 0.270 | -0.039 | 0.026 | -2.717 | -1.493 | -2.700 | -1.470 | 1.320 | 0.468 | -2.071 |
| 5 | 0.174 | 0.325 | 0.139 | 0.023 | -2.414 | -0.966 | -2.717 | -1.969 | 1.398 | 0.410 | -2.416 |
| 6 | 0.179 | -0.098 | 0.215 | 0.026 | -0.191 | 0.918 | 0.136 | -2.359 | 0.019 | 0.377 | -2.717 |
| 7 | -0.192 | 0.090 | 0.619 | 0.013 | -1.317 | 1.823 | -2.135 | -2.779 | -2.100 | 2.064 | -1.605 |
| 8 | -0.179 | -0.214 | 0.457 | 0.010 | 1.567 | 0.577 | 2.593 | -2.892 | 2.677 | 0.691 | 2.029 |
| 9 | 0.194 | 0.200 | 0.202 | 0.019 | 2.937 | 1.827 | -0.572 | -2.498 | -0.566 | 1.600 | 0.190 |
| 10 | 0.112 | -0.369 | 0.250 | 0.029 | 2.547 | -0.612 | 2.957 | -2.229 | -1.487 | 0.343 | 2.612 |
| 11 | -0.395 | 0.388 | 0.457 | 0.025 | -2.717 | 0.647 | -0.861 | -1.984 | -2.314 | 0.984 | -1.747 |
| 12 | 0.132 | -0.153 | 0.464 | 0.015 | 1.523 | 1.824 | -1.507 | -2.892 | -1.491 | 1.948 | 1.907 |
| 13 | 0.395 | -0.294 | -0.044 | 0.016 | -0.511 | 1.754 | 0.189 | -1.292 | 2.955 | 0.217 | 0.628 |
| 14 | 0.2 | -0.329 | 0.220 | 0.014 | -2.326 | -1.643 | 2.323 | -2.242 | -0.824 | 1.542 | 2.084 |
| 15 | -0.062 | -0.101 | 0.074 | 0.024 | 0.754 | 1.812 | 0.058 | -2.378 | 0.049 | 1.340 | 2.674 |
| 16 | -0.352 | 0.376 | 0.375 | 0.028 | -2.717 | 0.821 | -0.826 | -2.082 | -1.985 | 0.898 | -1.488 |
| 17 | -0.037 | 0.321 | 0.373 | 0.017 | -2.456 | -0.026 | -1.969 | -2.727 | 2.645 | 0.377 | -1.981 |
| 18 | 0.334 | -0.390 | 0.033 | 0.028 | -0.737 | 1.544 | 0.194 | -1.481 | 2.958 | 0.238 | 0.851 |
| 19 | 0.289 | 0.135 | 0.158 | 0.018 | 0.688 | 1.151 | 0.186 | -1.914 | -0.126 | 0.162 | 2.504 |
| 20 | 0.145 | -0.116 | 0.322 | 0.011 | 0.661 | -0.794 | -1.943 | -2.892 | 1.247 | 1.059 | -0.224 |

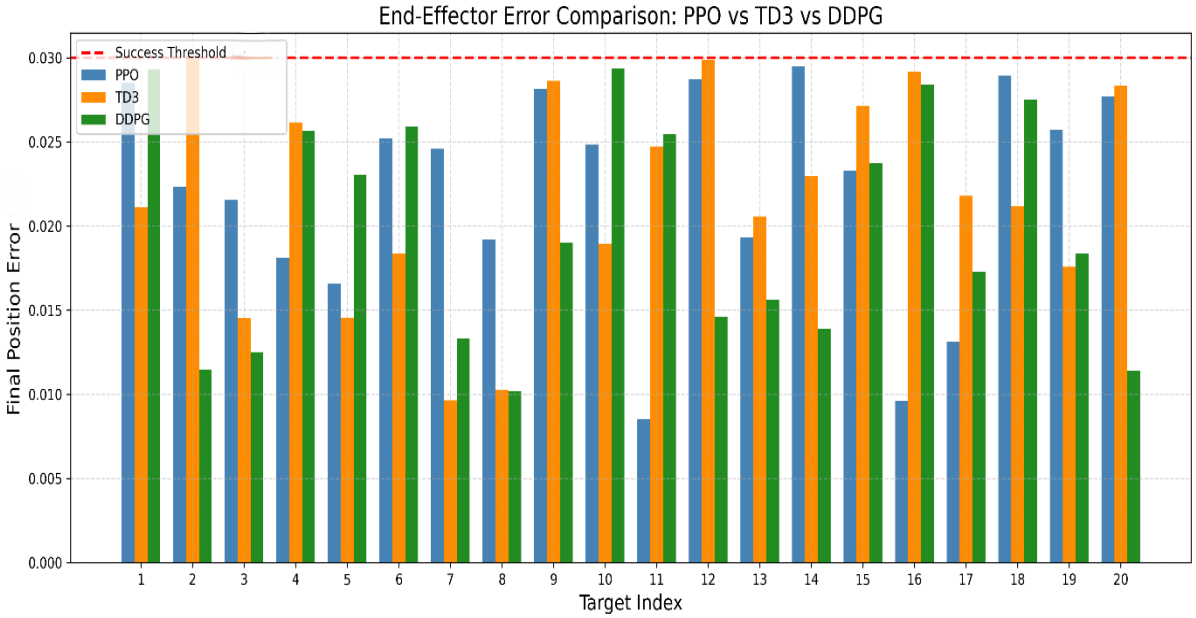


Fig. 7. Detailed position error comparison.

Table 7. Performance comparison of DRL Algorithms for IK solution.

| Algorithm | Average Error [cm] | Success Rate [%] |
|---|---|---|
| PPO | $0.0222 \pm 0.0064$ | 20/20 (100.0) |
| TD3 | $0.0218 \pm 0.0063$ | 20/20 (100.0) |
| DDPG | $0.0198 \pm 0.0067$ | 20/20 (100.0) |

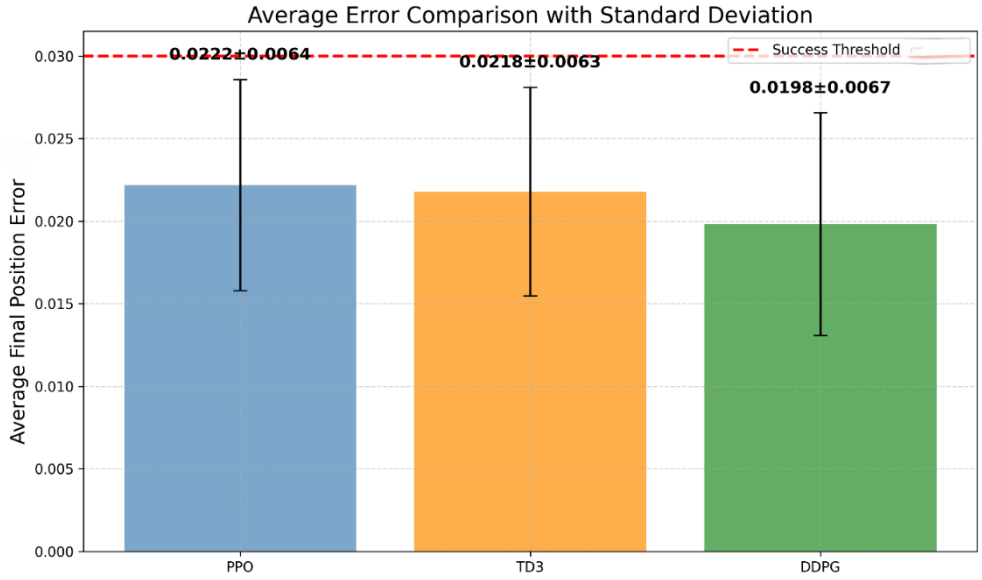


Fig. 8 Average position error comparison

## 5.2. Performance evaluation based on robustness against parametric uncertainty

Furthermore, in this subsection, we examine the robustness of DRL-trained agents by assessing their performance under parametric uncertainties in the physical parameters of the robotic arm. Such changes in physical parameters represent real-world situations where differences come from modeling simplifications, payload variations, manufacturing tolerances, and other factors. Testing DRL agents to perform under these conditions offers valuable insights into their practicality and strength in real-world applications.

To achieve this, we introduce parametric variations in the robot's Unified Robot Description Format (URDF) model by altering the link masses and inertial properties during the evaluation phase. Note that the agent is trained using the actual parameter values specified in the URDF model. Table 8 summarizes the robustness of all three trained agents under parametric uncertainties introduced in the robot model. The performance is evaluated for five variation cases, as detailed in the table. Note that case 6 represents the scenario where uncertainties are not considered and is included here for easy comparison with the other five cases involving parametric variations.

For all five cases considered here, both TD3 and DDPG achieve a 100% success rate. Both agents show greater strength against changes in link mass and inertial properties. In contrast, PPO suffers a slight drop in performance when the mass of link 7 changes in cases 1 and 3. Its success rate decreases to 95%. This indicates that the PPO agent is more sensitive to shifts in system dynamics. When it comes to accuracy, measured by mean error, TD3 consistently records the lowest mean error. This ranges from 0.019 to 0.022 across all cases of variation.

Table 8. Comparison of success rate and mean error under parametric variations for PPO, TD3, and DDPG

| Case | Link/Joint | Parameter | Actual | Uncertainty[%] | Success Rate [%] | | | Mean Error | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | PPO | TD3 | DDPG | PPO | TD3 | DDPG |
| 1 | Link7 | Mass | 0.2 | 10% | 95 | 100 | 100 | 0.033 | 0.020 | 0.022 |
| 2 | Link6 and Link7 | Mass | 1.3 | 10% | 100 | 100 | 100 | 0.021 | 0.021 | 0.020 |
| 3 | Link6 and Link7 | Mass | 0.2 | 20% | 95 | 100 | 100 | 0.030 | 0.019 | 0.020 |
| 4 | Hand | Mass | 0.81 | 20% | 100 | 100 | 100 | 0.022 | 0.019 | 0.024 |
| 5 | Link7 | Inertial(Ixx) | 0.1 | 20% | 100 | 100 | 100 | 0.021 | 0.020 | 0.022 |
| 6 | Without uncertainty | - | - | - | 100 | 100 | 100 | 0.022 | 0.022 | 0.020 |

## 6. CONCLUSIONS

This study comprehensively evaluated three DRL methods, PPO, TD3, and DDPG, for solving the IK problem of a 7-degree-of-freedom Panda Franka Emika robotic arm. The IK problem is defined as a continuous control task. Each algorithm was trained and tested in similar environmental conditions for fair comparison. All three DRL agents were then assessed on the basis of performance metrics such as accuracy, success rate, and robustness under parametric uncertainty. It is observed from the performance analysis of all three agents that although all of them achieve 100% success rate for reaching the target points, however they differ in convergence time for the training and the amount of parametric uncertainty that they can handle. For instance, we observed that the PPO agent takes 5000 episodes for training convergence, whereas the TD3 and DDPG agents take twice and thrice the amount of time, respectively, then the PPO agent. The faster convergence in the PPO agent is, however, achieved at the cost of losing 5% robustness performance when compared to TD3 and DDPG agents, which do not compromise on success rate for parametric uncertainty. Thus, based on the given performance indicator for a specific IK problem, the trade-off between the convergence time and robustness becomes evident, and an application specific choice can be made judicially using such analysis.

Overall, this work demonstrates the promising potential for solving IK using DRL for higher DoF cobots, positioning DRL as a compelling alternative to traditional IK approaches. Future efforts will focus on validating these DRL methods on physical hardware to address challenges related to sim-to-real transfer.

## REFERENCES

[1] M. Spong, M. Vidyasagar, *Robot Dynamics and Control*, John Wiley & Sons, 2008.

[2] A. Calzada-Garcia, J. Victores, F. Naranjo-Campos, C. Balaguer, "A review on inverse kinematics, control and planning for robotic manipulators with and without obstacles via deep neural networks," *Algorithms*, vol. 18, no. 1, p. 23, 2025, doi: 10.3390/a18010023.

[3] R. Murray, Z. Li, S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, CRC press, 2017.

[4] J. Craig, *Introduction to Robotics: Mechanics and Control*, Pearson Education India, 2009.

[5] S. Buss, J. Kim, "Selectively damped least squares for inverse kinematics," *Journal of Graphics Tools*, vol. 10, no. 3, pp. 37–49, 2025, doi: 10.1080/2151237X.2005.10129202.

[6] T. Nguyen T. Bui, W. Dai, T. Nguyen, L. Tao, "Apply some meta-heuristic algorithms to solve inverse kinematic problems of a 7-DoFs manipulator robot," *International Journal of Mechanical Engineering and Robotics Research*, vol. 10, no. 9, pp. 498-504, 2021, doi: 10.18178/ijmerr.10.9.498-504.

[7] S. Wen, J. Min, Z. Yu, Y. Li, X. Liu, H. Karimi, "Multiple population genetic algorithm-based inverse kinematics solution for a 6-DOF manipulator," *Journal of Field Robotics*, vol. 42, pp. 3440-3453, 2025, doi: 10.1002/rob.22585.

[8] S. Zhang, A. Li, J. Ren, R. Ren, "Kinematics inverse solution of assembly robot based on improved particle swarm optimization," *Robotica*, vol. 42, no. 3, pp. 833-845, 2024, doi: 10.1017/S0263574723001789.

[9] P. Srisuk, A. Sento, Y. Kitjaidure, "Inverse kinematics solution using neural networks from forward kinematics equations," 9th International Conference on Knowledge and Smart Technology, 2017, doi: 10.1109/KST.2017.7886084.

[10] A. Hassan, M. El-Habrouk, S. Deghedie, "Inverse kinematics of redundant manipulators formulated as quadratic programming optimization problem solved using recurrent neural networks: a review," *Robotica*, vol. 38, no. 8, pp. 1495-1512, 2024, doi: 10.1017/S0263574719001590.

[11] S. Tammishetty, V. Semwal, Y. Pathak, D. Joshi, "Inverse kinematic solution using neural networks for multimodal inputs and optimization in constrained workspace," *IEEE Sensors Letters*, vol. 9, no. 1, pp. 1-4, 2025, doi: 10.1109/LSENS.2024.3513418.

[12] R. Liu, F. Nageotte, P. Zanne, M. Mathelin, B. Dresp-Langley, "Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review," *Robotics*, vol. 10, no. 1, p. 22, 2021, doi: 10.3390/robotics10010022.

[13] D. Vyas, A. Markana, N. Padhiyar, "Robotic grasp synthesis using deep learning approaches: a survey," Mathematical Modeling, Computational Intelligence Techniques and Renewable Energy, Advances in Intelligent Systems and Computing, vol 1287, 2021, doi: 10.1007/978-981-15-9953-8_11.

[14] C. Mower, S. Theodoros, J. Moura, C. Rauch, L. Yan, N. Behabadi, M. Gienger, T. Vercauteren, C. Bergeles, S. Vijayakumar. "ROS-PyBullet Interface: A framework for reliable contact simulation and human-robot interaction," Conference on robot learning, 2023.

[15] J. Baumgärtner, M. Hansjosten, D. Schönhofen, I. Fleischer, "Pybullet industrial: a process-aware robot simulation," *Journal of Open-Source Software*, vol. 8, no. 85, p. 5174, doi: 10.21105/joss.05174.

[16] E. Berrocal, B. Sierra, H. Herrero, "Evaluating PyBullet and isaac sim in the scope of robotics and reinforcement learning," 7th Iberian Robotics Conference, 2024, doi: 10.1109/ROBOT61475.2024.10797383.

[17] A. Malik, Y. Lischuk, T. Henderson, R. Prazenica, "A deep reinforcement-learning approach for inverse kinematics solution of a high degree of freedom robotic manipulator," *Robotics*, vol. 11, no. 2, p. 44, 2024, doi: 10.3390/robotics11020044.

[18] C. Zhao, Y. Wei, J. Xiao, "Inverse kinematics solution and control method of 6-degree-of-freedom manipulator based on deep reinforcement learning," *Science Report,* vol. 14, p. 12467, 2024, doi: 10.1038/s41598-024-62948-6.

[19] S. Yu, G. Tan, "Inverse kinematics of a 7-degree-of-freedom robotic arm based on deep reinforcement learning and damped least squares," *IEEE Access*, vol. 13, pp. 4857-4868, 2025, doi: 10.1109/ACCESS.2024.3521539.

[20] J. Schulman, W. Filip, D. Prafulla, A. Radford, O. Klimov, "Proximal policy optimization algorithms," 2017, arXiv preprint arXiv:1707.06347.

[21] S. Fujimoto, H. Herke, M. David, "Addressing function approximation error in actor-critic methods," International Conference On Machine Learning, 2018.

[22] R. Dhaval, S. Parth, A. Markana, N. Padhiyar, "Cascade-trained deep reinforcement learning for PID gain optimization in precise joint position control of 6-DoF robotic arm," *Engineering Research Express*, vol. 7, no. 4, p. 045211, 2017, doi: 10.1088/2631-8695/ae090e.

[23] P. Lillicrap, J. Jonathan, P. Alexander, H. Nicolas, T. Erez, Y. Tassa, D. Silver, D. Wierstra, "Continuous control with deep reinforcement learning," 2015, arXiv preprint arXiv:1509.02971.

[24] A. Patterson, N. Samuel, W. Martha, W. Adam, "Empirical design in reinforcement learning," *Journal of Machine Learning Research*, vol. 25, no. 318, pp. 1-63, 2014.