Deep Reinforcement Learning Based Robotic Arm's Target Reaching Performance Enhancement

Ldet Honelign , Yoseph Abebe , Young Seok Jung , Abera Tullu , Sunghun Jung *

Posted Date: 26 January 2025

doi: 10.20944/preprints202501.1917.v1

Keywords: Deep reinforcement learning; Robotic arm manipulator; OpenAI Gym; PyBullet; Panda Gym

Article

Deep Reinforcement Learning Based Robotic Arm's Target Reaching Performance Enhancement

Ldet Honelign [1,†,‡], Yoseph Abebe [1,‡], Young Seok Jung [2,‡], Abera Tullu [3,‡] and Sunghun Jung [3,*]

[1] Department of Electromechanical Engineering, Addis Ababa Science and Technology University. Kilnto, 16417, Ethiopia
[2] D. AIR Co., Ltd., #220 & #221, 216 Munhwa-ro, Naju-so, Juollanam-do, 58325, Republic of Korea
[3] Faculty of Smart Vehicle System Engineering, Chosun University, Dong-gu, Gwangju 61452, Republic of Korea
* Correspondence: jungx148@chosun.ac.kr; Tel.: +82-62-230-7188 (S.J.)
† Current address: Department of Aerospace Engineering, Chosun University, Seoseok-dong, Gwangju, 61452, Republic of Korea
‡ These authors contributed equally to this work.

Abstract: This work presents the implementation of Deep Deterministic Policy Gradient (DDPG) algorithm to enhance target reaching capability of the seven Degree-of-Freedom (7-DoF) Franka Panda robot arm. A simulated environment is established by employing OpenAI Gym, PyBullet, and Panda Gym. Upon completion of 100,000 training time steps, the DDPG algorithm attains a success rate of 100% and an average reward of -1.8. The actor loss and critic loss values are 0.0846 and 0.00486, respectively, indicating improved decision-making and accurate value function estimations. The simulation results demonstrate the efficiency of DDPG in improving robotic arm performance, highlighting its potential for application to improve robot arm manipulation.

Keywords: deep reinforcement learning; robotic arm manipulator; OpenAI gym; PyBullet; panda gym

1. Introduction

The enhancement of precision for robot arm manipulation remains a core research area for acquiring full autonomy of robots in various sectors such as industrial manufacturing and assembly processes. The next logical progression in this field is to achieve complete autonomy of robotic manipulators through the use of machine learning (ML), artificial neural networks (ANNs), and artificial intelligence (AI) as a whole, given the maturity of image recognition and vision systems [1,2]. Numerous attempts have been made to create intelligent robots that can take tasks and execute accordingly [3–10].

Although developing a system with intelligence close to that of humans is still a long way off, robots that can perform specialized autonomous activities, such as intelligent facial emotion recognition [11], fly in natural and man-made environments [12], drive a vehicle [13], swim [14], carry boxes and material in different terrains [15], and pick up and place objects [16,17] is already actualized.

However, some challenges must be overcome to achieve this goal. For instance, the mapping complexity from Cartesian space to the joint space of a robot arm increases with the number of joints and linkages that the manipulator has. This is problematic because the tasks assigned to a robotic arm are in Cartesian space, whereas the commands (velocity or torque) are in joint space [18,19]. Therefore, if full autonomy of robotic manipulators is the objective, the target-reaching problem is probably one of the most crucial factors that must be addressed.

The field of reinforcement learning, as described in [20,21], is a type of machine learning that aims to maximize the outcome of a given system using a dynamic and autonomous trial-and-error approach. It shares a similar objective with human intelligence, which is characterized by the ability to perceive and retain information as knowledge to be used for environment-adaptive behaviors. Central to the reinforcement learning framework are trial-and-error search and delayed rewards, which allow the learning strategy to interact with the environment by performing actions and discovering rewards [22].

 doi:10.20944/preprints202501.1917.v1

Through this approach, software agents and machines can automatically select the most effective course of action to take in a given circumstance, thus improving performance. Reinforcement learning offers a framework and set of tools for designing sophisticated and challenging-to-engineer behaviors in robotics [23,24]. In contrast, the challenges presented by robotic issues serve as motivation, impact, and confirmation of advances in reinforcement learning. Multiple previous works on the implementation of reinforcement learning in the field of robotics depict this fact [25–29].

2. Modeling of Robotic Arm

2.1. Direct Kinematic Model of Robot Arm

The rotation matrices in the DH coordinate frame represent the rotations about the X and Z axes. The rotation matrices for these axes are, respectively, given as:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\delta_i & -S\delta_i \\ 0 & S\delta_i & C\delta_i \end{bmatrix} \tag{1}$$

and

$$R_z = \begin{bmatrix} C\phi_i & -S\phi_i & 0 \\ S\phi_i & C\phi_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

The homogeneous transformation matrix (T^i_{i-1}) that accounts for rotation and translation is given as:

$$\begin{aligned} T^i_{i-1} &= \begin{bmatrix} R^1_{i-1} & p^1_{i-1} \\ 0 & 1 \end{bmatrix} \\ &= \text{Rot}_x(\phi_i) \cdot \text{trans}_x(d_i) \cdot \text{trans}_z(\Delta_i) \cdot \text{Rot}_z(a_i) \\ \implies T^1_{i-1} &= \begin{bmatrix} C\phi_i & -S\phi_i C\delta_i & S\phi_i S\delta_i & a_i C\phi_i \\ S\phi_i & C\phi_i C\delta_i & -C\phi_i S\delta_i & a_i S\phi_i \\ 0 & S\delta_i & C\delta_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3}$$

where:

- The rotation matrix (R^1_{i-1}) represents the orientation of the i-th frame relative to the $(i-1)$-th frame.
- P^1_{i-1} represents the center of the link frame with components (P_x, P_y, and P_z).

2.1.1. DH Axis Representation

The four DH parameters describe the translation and rotation relationship between two consecutive coordinate frames as follows:

- d: a distance between the current frame and the previous frame along the Z-axis,
- (ϕ): an angle between the X-axis of the previous frame and the X-axis of the current frame about the previous z-axis,
- a: a distance between the Z-axes of the current and previous frames.
- δ: an offset of the previous frame from the current frame along the Z-axis of the current frame.

The DH parameters for the Franka Panda robot, shown in Figure 1 are given in Table 1. From the above DH parameters and based on the homogeneous transformation matrix 3, the transformation matrix for the Franka Panda robot is derived as.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 26 January 2025 doi:10.20944/preprints202501.1917.v1

Table 1. DH Axis Representation.

Joint	a(m)	d(m)	δ(m)	ϕ(rad)
1	0	0.333	0	ϕ_1
2	0	0	-90	ϕ_2
3	0	0.316	90	ϕ_3
4	0.0825	0	90	ϕ_4
5	-0.0825	0.384	-90	ϕ_5
6	0	0	90	ϕ_6
7	0.088	0	90	ϕ_7
Flange	0	0.107	0	0

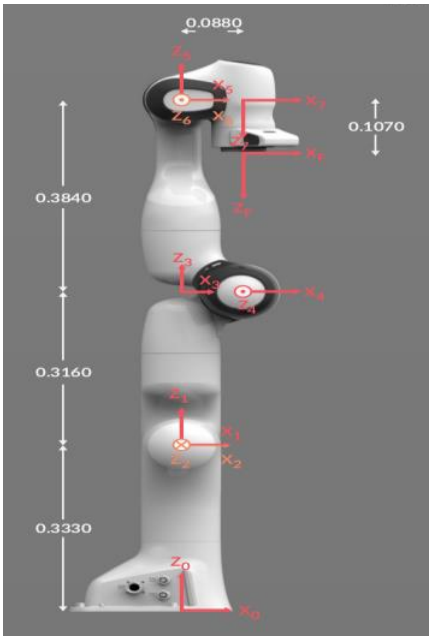

Figure 1. DH Axis Representation of Franka Panda robot [30].

$$T_{i-1}^{1} = \begin{bmatrix} C\phi_i & -S\phi_i C\delta_i & S\phi_i S\delta_i & a_i C\phi_i \\ S\phi_i & C\phi_i C\delta_i & -C\phi_i S\delta_i & a_i S\phi_i \\ 0 & S\delta_i & C\delta_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{0}^{1} = \begin{bmatrix} C\phi_1 & -S\phi_1 & 0 & 0 \\ S\phi_1 & C\phi_1 & 0 & 0 \\ 0 & 0 & 1 & 0.33 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$T_{1}^{2} = \begin{bmatrix} C\phi_2 & 0 & -S\phi_2 & 0 \\ S\phi_2 & 0 & C\phi_2 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$T_{2}^{3} = \begin{bmatrix} C\phi_3 & 0 & S\phi_3 & 0 \\ S\phi_3 & 0 & -C\phi_3 & 0 \\ 0 & 1 & 1 & 0.316 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

 doi:10.20944/preprints202501.1917.v1

$$T_3^4 = \begin{bmatrix} C\phi_4 & 0 & S\phi_4 & 0.0825C\phi_4 \\ S\phi_4 & 0 & -C\phi_4 & 0.0825S\phi_4 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$T_4^5 = \begin{bmatrix} C\phi_5 & 0 & -S\phi_5 & 0.0825C\phi_5 \\ S\phi_5 & 0 & C\phi_5 & 0.0825S\phi_5 \\ 0 & -1 & 0 & 0.384 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$T_5^6 = \begin{bmatrix} C\phi_6 & 0 & S\phi_6 & 0 \\ S\phi_6 & 0 & -S\phi_6 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

$$T_6^7 = \begin{bmatrix} C\phi_7 & 0 & S\phi_7 & 0.088C\phi_7 \\ S\phi_7 & 0 & -C\phi_7 & 0.088S\phi_7 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

$$T_7^0 = T_0^1 \cdot T_1^2 \cdot T_2^3 \cdot T_3^4 \cdot T_4^5 \cdot T_5^6 \cdot T_6^7 \tag{11}$$

$$T_7^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

The orientation and position of the end-effector, respectively, are given by:

$$R_{11} = -S\phi_1 S\phi_3 C\phi_2 + C\phi_1 C\phi_3 C\phi_4 \tag{13}$$
$$R_{12} = -S\phi_1 C\phi_2 C\phi_3 C\phi_4 - S\phi_3 C\phi_1 \tag{14}$$
$$R_{13} = S\phi_2 C\phi_1 \tag{15}$$
$$R_{21} = S\phi_1 C\phi_3 C\phi_4 + S\phi_3 C\phi_1 C\phi_2 \tag{16}$$
$$R_{22} = -S\phi_1 S\phi_3 C\phi_2 + C\phi_1 C\phi_3 C\phi_4 \tag{17}$$
$$R_{23} = -S\phi_2 S\phi_1 \tag{18}$$
$$R_{31} = -S\phi_2 C\phi_3 C\phi_4 \tag{19}$$
$$R_{32} = S\phi_2 S\phi_3 C\phi_4 \tag{20}$$
$$R_{33} = C\phi_2 \tag{21}$$

 doi:10.20944/preprints202501.1917.v1

and

$$\begin{aligned} p_x = & -0.107 S\phi_2 C\phi_1 + 0.088 C\phi_1 \\ & + 0.384(-S\phi_1 S\phi_3 C\phi_2 + C\phi_1 C\phi_3 C\phi_4) \\ & + 0.316 C\phi_1 C\phi_2 + 0.333 C\phi_1 \end{aligned} \tag{22}$$

$$\begin{aligned} p_y = & -0.107 S\phi_2 S\phi_1 + 0.088 S\phi_1 \\ & + 0.384(S\phi_1 C\phi_3 C\phi_4 + S\phi_3 C\phi_1 C\phi_2) \\ & + 0.316 S\phi_1 C\phi_2 + 0.333 S\phi_1 \end{aligned} \tag{23}$$

$$\begin{aligned} p_z = & 0.107 C\phi_2 \\ & + 0.384 S\phi_2 C\phi_3 C\phi_4 \\ & + 0.316 S\phi_2 + 0.333 S\phi_2 \end{aligned} \tag{24}$$

2.2. Incremental Inverse Kinematics of Robot Arm

The 3D position vector $\mathbf{x} \in \mathbb{R}^6$ of the end-effector (EE) is given by:

$$\mathbf{y} = \mathbf{f}(\mathbf{q}) \tag{25}$$

In case of the Panda robot, there are $n = 7$ active joint angles $\mathbf{q} = [\phi_1, \ldots, \phi_7]^\mathsf{T}$. With these joint angles(q) and direct kinematics $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^6$, the goal is to solve the inverse kinematics $\mathbf{q} = \mathbf{f}^{-1}(\mathbf{y})$ using incremental inverse kinematics. In incremental inverse kinematics, a direct kinematics is linearized around the current joint angle configuration q^* as

$$\delta\,\mathbf{x}|_{\mathbf{x}^*} \propto \delta\,\mathbf{q}|_{\mathbf{q}^*}\,. \tag{26}$$

The goal is to find the change in joint angles ($\delta\mathbf{q}$) that corresponds to a desired change in the end-effector position ($\delta\mathbf{y}$). The linearized form of the direct kinematics equations is expressed as 26, where

- $\delta\,\mathbf{x}|_{\mathbf{y}^*}$ represents the change in end-effector position around a reference point $\mathbf{x}^*$ and
- $\delta\,\mathbf{q}|_{\mathbf{q}^*}$ represents the change in joint angles around a reference point $\mathbf{q}^*$.

The proportionality symbol ($\propto$) indicates that the change in end-effector position is directly related to the change in joint angles. To solve for the change in joint angles, the Jacobian matrix, denoted as $\mathbf{J_f}$, is utilized. The Jacobian matrix is a matrix of partial derivatives that describes how the end-effector position $\mathbf{f}$ depends on the joint angles $\mathbf{q}$. Specifically, the Jacobian matrix is defined as

$$J_\mathbf{f} := \left(\frac{\partial f_i}{\partial q_j}\right)_{i,j}, \tag{27}$$

- where $\frac{\partial f_i}{\partial q_j}$ represents the partial derivative of the *ith* component of the end-effector position with respect to the *jth* joint angle.

By multiplying the Jacobian matrix $\mathbf{J_f}$ by the change in joint angles $\delta\,\mathbf{q}|_{\mathbf{q}^*}$, an approximation of the change in the end-effector position $\delta\,\mathbf{x}|_{\mathbf{y}^*}$ is obtained. The joint angles are iteratively updated to minimize the difference between the current end-effector position and the desired end-effector position.This can be efficiently solved around $(\mathbf{x}^*, \mathbf{q}^*)$using the Jacobian.

$$\mathbf{J_f} := (\partial f_i / \partial q_j)_{i,j}$$

$$\Delta\,\mathbf{x}|_{\mathbf{x}^*} = \mathbf{J_f}(\mathbf{q}^*)\delta\,\mathbf{q}|_{\mathbf{q}^*} \tag{28}$$

2.2.1. Steps of Incremental Inverse Kinematics

Given: target pose $\mathbf{y}^{(t)}$

 doi:10.20944/preprints202501.1917.v1

Required: joint angles $\mathbf{q}^{(t)}$

1. Define the starting pose $(\mathbf{x}^{(0)}, \mathbf{q}^{(0)})$ and set up the Incremental inverse kinematics from (3.29)

$$\delta \mathbf{x}|_{\mathbf{x}^{(0)}} = \mathbf{J}(\mathbf{q}^{(0)})\delta \mathbf{q}|_{\mathbf{q}^{(0)}}$$

2. Determine the deviation $\delta \mathbf{y}^{(r)}$ relative to the target pose; e.g. $(\mathbf{y}^{(t)} - \mathbf{y}^{(0)})$
3. Check for termination; e.g.$\max(|\delta y_{i,j}^{(r)}|) \leq \epsilon$.
4. Solve

$$\delta \mathbf{x}^{(k)} = \mathbf{J}(\mathbf{q}^{(k)})\delta \mathbf{q}^{(k)} \tag{29}$$

5. Calculate new joint angles

$$\mathbf{q}^{(r+1)} = \mathbf{q}^{(r)} + \delta \mathbf{q}^{(r)} \tag{30}$$

6. $r \leftarrow r + 1$

3. Deep Reinforcement Learning Algorithm Design

3.1. Policy Gradient Algorithm

The policy gradient theorem states that the expected return to the policy parameters can be calculated as the sum of the action value function $q^{\pi}(s, a)$ multiplied by the policy function $\pi_{\varrho}(s, a)$, summed over all states s and actions a, and weighted by the stationary distribution of states $d^{\pi}(s)$.

$$O(\varrho) = \sum_{s} d^{\pi}(s) \sum_{a} Q^{\pi}(s, a)\pi_{\varrho}(s, a) \tag{31}$$

where:

- **Objective function(J_{ϱ})**: Represents the expected cumulative reward obtained by following the policy π_{ϱ} in the given environment. The objective function is optimized by adjusting the parameter ϱ to maximize the expected cumulative reward.
- **Discounted state distribution($d^{\pi}(s)$)**: Represents the probability of being in a particular state s under the policy π. Mathematically,

$$d^{\pi}(s) = \lim_{t \to \infty} P(s_t = s | s_0, \pi_{\varrho}) \tag{32}$$

  where $s_t = s$ when starting from s_0 and following policy π_{ϱ} for t time steps.
- **Action-value function (($Q^{\pi}(s, a)$)**: Represents the expected cumulative reward obtained by taking action a in state s and following the policy π thereafter.
- **Policy function($\pi_{\varrho}(s, a)$)**: Represents the probability of taking action a in state s under the parameterized policy ϱ.

The policy gradient theorem helps to solve this problem by providing a formula for the gradient of the expected return to the policy parameters. This formula involves the stationary distribution of the Markov chain and is given by:

$$\nabla_{\varrho} J_{\varrho} = \sum_{s} d^{\pi}(s) \sum_{a} q^{\pi}(s, a) \nabla_{\varrho} \pi_{\varrho}(a|s) \tag{33}$$

where $q^{\pi}(s, a)$ is the state-action value function for policy π_{ϱ}.

$$\propto \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} Q^{\pi}(s, a) \nabla_{\varrho} \pi_{\varrho}(s, a) \tag{34}$$

 doi:10.20944/preprints202501.1917.v1

3.1.1. Derivation of Policy Gradient Theorem

$$\nabla_\varrho V^\pi(s) = \nabla_\varrho \left[\sum_{a \in A} \pi_\varrho(a|s) Q^\pi(s,a) \right] \tag{35}$$

using derivative product rule:

$$= \sum_{a \in A} (\nabla_\varrho \pi_\varrho(a|s) Q^\pi(s,a) + \pi_\varrho(a|s) \nabla_\varrho Q^\pi(s,a))$$

The steps of the derivation using, the derivative product rule from equation (4.6) are as follows: Step 1: Apply the derivative product rule:

$$\nabla_\varrho V^\pi(s) = \nabla_\varrho \sum_{a \in A} (\pi_\varrho(a|s) Q^\pi(s,a))$$

Step 2: Distribute the derivative operator inside the summation:

$$\nabla_\varrho V^\pi(s) = \sum_{a \in A} \nabla_\varrho (\pi_\varrho(a|s) Q^\pi(s,a))$$

Step 3: Apply the chain rule to differentiate the product of $\pi_\varrho(a|s)$ and $Q^\pi(s,a)$ to ϱ:

$$\nabla_\varrho V^\pi(s) = \sum_{a \in A} \left(\nabla_\varrho \pi_\varrho(a|s) Q^\pi(s,a) + \pi_\varrho(a|s) \nabla_\varrho Q^\pi(s,a) \right)$$

Step 4: Simplify the expression by rearranging the terms.

After applying the derivative product rule, the derivative of the state-value function $V^\pi(s)$ to the policy parameter ϱ is:

$$\nabla_\varrho V^\pi(s) = \sum_{a \in A} \left(\nabla_\varrho \pi_\varrho(a|s) Q^\pi(s,a) + \pi_\varrho(a|s) \nabla_\varrho Q^\pi(s,a) \right) \tag{36}$$

Extend $Q^\pi(s,a)$ by incorporating the future state value. This can be done by considering the state-action pair (s,a) and summing over all possible future states s' and corresponding rewards r:

$$\begin{aligned} = \sum_{a \in A} \Big[& \nabla_\varrho \pi_\varrho(a|s) Q^\pi(s,a) \\ & + \pi_\varrho(a|s) \nabla_\varrho \sum_{s',r} P(s',r|s,a)(r + V^\pi(s')) \Big]; \end{aligned} \tag{37}$$

Since $P(s',r|s,a)$ and r are not functions of ϱ, the derivative operator ∇_ϱ can be moved inside the summation over s', r without affecting these terms.

$$\begin{aligned} \nabla_\varrho V^\pi(s) = \sum_{a \in A} \Big[& \nabla_\varrho \pi_\varrho(a|s) Q^\pi(s,a) \\ & + \pi_\varrho(a|s) \sum_{s',r} P(s',r|s,a) \nabla_\varrho V^\pi(s') \Big] \end{aligned} \tag{38}$$

 doi:10.20944/preprints202501.1917.v1

Next, it is observed that $\nabla_\varrho V^\pi(s')$ is the derivative of the state-value function to the policy parameter ϱ at state s'. This can be rewritten as $\nabla_\varrho V^\pi(s') = \nabla_\varrho V^\pi(s') \sum_{s'} P(s'|s,a)$. Here, $P(s'|s,a)$ represents the probability of transitioning to state s' given the current state-action pair (s,a). The following substitution is now made:

$$= \sum_{a\in A}\left[\nabla_\varrho \pi_\varrho(a|s)Q^\pi(s,a) + \pi_\varrho(a|s)\sum_{s',r} P(s',r|s,a)\nabla_\varrho V^\pi(s')\right]$$

$$= \sum_{a\in A}\left[\nabla_\varrho \pi_\varrho(a|s)Q^\pi(s,a) + \pi_\varrho(a|s)\sum_{s'} P(s'|s,a)\nabla_\varrho V^\pi(s')\right]$$

Where $P(s'|s,a) = \sum_r P(s'|s,a)$

Now there is,

$$\begin{aligned} = \nabla_\varrho V^\pi(s) = \sum_{a\in A}\Big[&\nabla_\varrho \pi_\varrho(a|s)Q^\pi(s,a) \\ &+ \pi_\varrho(a|s)\sum_{s'} P(s'|s,a)\nabla_\varrho V^\pi(s')\Big] \end{aligned} \tag{39}$$

Consider the following visiting sequence and identify the probability of changing state s to state x using policy π_ϱ after k steps as $\rho(s \to x, k)$

$$s \xrightarrow{a\sim\pi_\varrho(.|s)} s' \xrightarrow{a\sim\pi_\varrho(.|s')} s'' \xrightarrow{a\sim\pi_\varrho(.|s'')} \dots$$

- When $k = 0 : \rho(s \to s, k = 0) = 1$.
- When $k = 1$, consider every action that might be taken and add up the probabilities of reaching the desired state:

$$\rho(s \to x, k = 1) = \sum_a \pi_\varrho(a|s)P(s'|s,a) \tag{40}$$

- The goal is to move from s to x after $k+1$ steps, by following π_ϱ.The agent can first move from s to intermediate state $s'(s' \in S)$ going to final state x in the last steps after the k stages. This allows to recursively update the visitation probability.

$$\rho(s \to x, k+1) = \sum_{s'} \rho^\pi(s \to s', k)\rho(s' \to x, 1) \tag{41}$$

After discussing the probability $\rho(s \to x, k)$ for transitioning from state s to state x after a certain number of steps k, the next step is to drive a recursive formulation for $\nabla_\varrho V^\pi(s)$.

To accomplish this, a function $\phi(s)$ is introduced, defined as:

$$\phi(s) = \sum_{a\in A} \nabla_\varrho \pi_\varrho(a|s)Q^\pi(s,a) \tag{42}$$

Here, $\Phi(s)$ represents the sum of the gradients of the policy π_ϱ to ϱ, weighted by the corresponding action-value function $Q^\pi(s,a)$.

To simplify (4.10)

$$\nabla_\varrho V^\pi(s) = \Phi(s) + \left[\sum_{a\in A} \pi_\varrho(a|s)\sum_{s'} P(s'|s,a)\nabla_\varrho V^\pi(s')\right] \tag{43}$$

 doi:10.20944/preprints202501.1917.v1

$$= \phi(s) + \sum_{a} \pi_{\varrho}(a|s) \sum_{s'} P(s'|s,a) \nabla_{\varrho} V^{\pi}(s') \tag{44}$$

$$= \Phi(s) + \sum_{s'} \sum_{a \in A} \pi_{\varrho}(a|s) P(s'|s,a) \nabla_{\varrho} V^{\pi}(s') \tag{45}$$

$$= \Phi(s) + \sum_{s'} \sum_{a \in A} \pi_{\varrho}(a|s) P(s'|s,a) \nabla_{\varrho} V^{\pi}(s') \tag{46}$$

$$= \Phi(s) + \sum_{s'} \rho^{\pi}(s \to s', 1) \nabla_{\varrho} V^{\pi}(s') \tag{47}$$

$$= \Phi(s) + \sum_{s'} \rho^{\pi}(s \to s', 1) \left[\phi(s') + \sum_{s''} \rho^{\pi}(s' \to s'', 1) \nabla_{\varrho} V^{\pi}(s'') \right]$$

Consider s' as the middle point for $s' \to s''$

$$\begin{aligned} &= \Phi(s) + \left[\sum_{s'} \rho^{\pi}(s \to s', 1) \phi(s) \right] \\ &+ \left[\sum_{s'} \rho^{\pi}(s' \to s'', 2) \nabla_{\varrho} V^{\pi}(s'') \right] \end{aligned} \tag{48}$$

The expression for $\nabla_{\varrho} V^{\pi}(s)$ can be unrolled as follows:

$$\begin{aligned} &= \Phi(s) + \left[\sum_{s'} \rho^{\pi}(s \to s', 1) \Phi(s') \right] \\ &+ \left[\sum_{s''} \rho^{\pi}(s \to s'', 2) \Phi(s'') \right] \\ &+ \left[\sum_{s'''} \rho^{\pi}(s \to s''', 3) \nabla_{\varrho} V^{\pi}(s''') \right] \end{aligned} \tag{49}$$

$$\nabla_{\varrho} V^{\pi}(s) = \sum_{x \in S} \sum_{k=0} \rho^{\pi}(s \to x, k) \Phi(x) \tag{50}$$

Eliminating the derivatives of the Q-value function $\nabla_{\varrho} Q^{\pi}(s)$ and inserting objective function $O(\varrho)$ in (4.5), starting from state S_0

$$\nabla_{\varrho} O(\varrho) = \nabla_{\varrho} V^{\pi}(S_0) \tag{51}$$

$$= \sum_{s} \sum_{k=0} \rho^{\pi}(s \to x, k) \phi(s); \tag{52}$$

Let, $\eta(s) = \sum_{s} \rho^{\pi}(s, \to x, k)$

 doi:10.20944/preprints202501.1917.v1

$$\nabla_{\varrho} O(\varrho) = \sum_{s} \eta(s)\phi(s) \tag{53}$$

Substituting $\eta(s) = \sum_s \rho^{\pi}(s, \rightarrow x, k)$ in to eqn(4.24)

$$\nabla_{\varrho} O(\varrho) = \sum_{s} \eta(s)\phi(s) \tag{54}$$

Normalize $\eta(s)$ in (4.27), $s \in S$ to be probability distribution as show down bellow:

$$\nabla_{\varrho} O(\varrho) = \left(\sum_{s} \eta(s)\right) \sum_{s} \left[\frac{\eta(s)}{\sum_s \eta(s)} \phi(s)\right] \tag{55}$$

Since the $\sum_s \eta(s)$ is constant, the gradient of the objective function is proportional to the normalized $\eta(s)$ and $\phi(s)$

$$\nabla_{\varrho} O(\varrho) \propto \sum_{s} \left[\frac{\eta(s)}{\sum_s \eta(s)} \phi(s)\right] \tag{56}$$

Where $d^{\pi}(s) = \frac{\eta(s)}{\sum_s \eta(s)}$ is stationary distribution.

In the episodic case, the constant proportionality $\sum_s \eta(s)$ is the average length of an episode; In the continuing case, it is one(1) [31].

$$= \sum_{s} d^{\pi}(s) \sum_{a \in A} \nabla_{\varrho} \pi_{\varrho}(a|s) Q^{\pi}(s, a) \tag{57}$$

$$\nabla_{\varrho} O(\varrho) \propto \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} Q^{\pi}(s, a) \nabla_{\varrho} \pi(\varrho)(a|s) \tag{58}$$

$$\nabla_{\varrho} O(\varrho) = \sum_{s \in S} d^{\pi}(s) \frac{\nabla_{\varrho} \pi_{(}\varrho)(a|s)}{\pi_{(}\varrho)(a|s)} \tag{59}$$

$$= \sum_{s \in S} d^{\pi}(s) \frac{\nabla_{\varrho} \pi_{(}\varrho)(a|s)}{\pi_{(}\varrho)(a|s)} \tag{60}$$

$$\nabla_{\varrho} O(\varrho) = E_{\pi}\{Q^{\pi}(s, a) \nabla_{\varrho} \ln(\pi_{\varrho}(a|s))\} \tag{61}$$

Where E_{π} refers to $E_{s \sim d_{\pi}, a \sim \pi_{\varrho}}$ when the distribution of states and actions follows policy π_{ϱ}(on policy).

3.1.2. Off-Policy Policy Gradient

Since DDPG is one of the off-policy Policy gradient Algorithms, First let's talk about off-policy Policy gradient Algorithms in great detail.

The behavior policy for collecting samples is known and labeled as $\alpha(a|s)$ the objective function sums up the reward over the state distribution defined by this behavior policy:

$$O(\varrho) = \sum_{s \in \mathcal{S}} d^{\alpha}(s) \sum_{a \in \mathcal{A}} Q^{\pi}(s, a) \pi_{\varrho}(a|s)$$

$$= \mathbb{E}_{s \sim d^{\alpha}} \left[\sum_{a \in \mathcal{A}} Q^{\pi}(s, a) \pi_{\varrho}(a|s) \right]$$

 doi:10.20944/preprints202501.1917.v1

where $d^{\alpha}(s)$ is the stationary distribution of the behavior policy α since $d^{\alpha}(s) = \lim_{t\to\infty} P(S_t = s|S_0, \alpha)$and Q^{π} is the action-value function estimated about the target policy π. Given that the training observations are sampled by $a \sim \alpha(a|s)$, the gradient can be rewritten as:

$$\nabla_{\varrho} O(\varrho) = \nabla_{\varrho} \mathbb{E}_{s\sim d^{\alpha}} \Big[\sum_{a\in\mathcal{A}} Q^{\pi}(s,a)\pi_{\varrho}(a|s) \Big] \tag{62}$$

By Derivative product rule.

$$= \mathbb{E}_{s\sim d^{\alpha}} \Big[\sum_{a\in\mathcal{A}} \big(Q^{\pi}(s,a)\nabla_{\varrho}\pi_{\varrho}(a|s) + \pi_{\varrho}(a|s)\nabla_{\varrho} Q^{\pi}(s,a) \big) \Big] \tag{63}$$

By ignoring $\pi_{\varrho}(a|s)\nabla_{\varrho} Q^{\pi}(s,a)$

$$\overset{(i)}{\approx} \mathbb{E}_{s\sim d^{\alpha}} \Big[\sum_{a\in\mathcal{A}} Q^{\pi}(s,a)\nabla_{\varrho}\pi_{\varrho}(a|s) \Big] \tag{64}$$

$$= \mathbb{E}_{s\sim d^{\alpha}} \Big[\sum_{a\in\mathcal{A}} \alpha(a|s) \frac{\pi_{\varrho}(a|s)}{\alpha(a|s)} Q^{\pi}(s,a) \frac{\nabla_{\varrho}\pi_{\varrho}(a|s)}{\pi_{\varrho}(a|s)} \Big] \tag{65}$$

$$= \mathbb{E}_{\alpha} \Big[\frac{\pi_{\varrho}(a|s)}{\alpha(a|s)} Q^{\pi}(s,a)\nabla_{\varrho} \ln \pi_{\varrho}(a|s) \Big] \tag{66}$$

Where $\left[\frac{\pi_{\varrho}(a|s)}{\alpha(a|s)}\right]$ is the importance weight. Since Q^{π} is a function of the target policy and, consequently, a function of the policy parameter ϱ, the derivative $\nabla_{\varrho} Q^{\pi}(s,a)$ must also be computed using the product rule. However, computing $\nabla_{\varrho} Q^{\pi}(s,a)$ directly is challenging in practice. Fortunately, by approximating the gradient and ignoring the gradient of Q^{π}, policy improvement can still be guaranteed, and eventual convergence to the true local minimum is achieved.

In summary, when applying policy gradient in the off-policy setting, it can be adjusted by a weighted sum, where the weight is the ratio of the target policy to the behavior policy, $\left[\frac{\pi_{\varrho}(a|s)}{\alpha(a|s)}\right]$ [31].

*3.2. Deterministic Policy Gradient (**DPG**)*

The policy function $\pi(\cdot|s)$ is typically represented as a probability distribution over actions A based on the current state, making it inherently stochastic. However, in the case of the Deterministic Policy Gradient (DPG), the policy is modeled as a deterministic decision, denoted as $a = \mu(s)$. Instead of selecting actions probabilistically, DPG directly maps states to specific actions without uncertainty. Let:

- $\rho_0(s)$ The initial distribution over states
- $\rho^{\mu}(s \to s', k)$: Starting from state s, the visitation probability density at state s' after moving k steps by policy μ.
- $\rho^{\mu}(s')$: Discounted state distribution, defined as $\rho^{\mu}(s') = \int_{\mathcal{S}} \sum_{k=1}^{\infty} \gamma^{k-1}\rho_0(s)\rho^{\mu}(s \to s', k)d_s$

The objective function to optimize for is listed as follows:

$$O(\varrho) = \int_{\mathcal{S}} \rho^{\mu}(s) Q(s, \mu_{\varrho}(s))ds \tag{67}$$

According to the chain rule, first, take the gradient of Q with respect to the action a and then take the gradient of the deterministic policy function μ w.r.t.ϱ :

 doi:10.20944/preprints202501.1917.v1

$$\nabla_{\varrho} O(\varrho) = \int_{\mathcal{S}} \rho^{\mu}(s) \nabla_a Q^{\mu}(s,a) \nabla_{\varrho} \mu_{\varrho}(s)|_{a=\mu_{\varrho}(s)} ds \tag{68}$$

$$= \mathbb{E}_{s\sim\rho^{\mu}}[\nabla_a Q^{\mu}(s,a) \nabla_{\varrho} \mu_{\varrho}(s)|_{a=\mu_{\varrho}(s)}] \tag{69}$$

*3.3. Deep Deterministic Policy Gradient (**DDPG**)*

By combining DQN and DPG, DDPG leverages the power of deep neural networks to handle high-dimensional state spaces and complex action spaces, making it suitable for a wide range of reinforcement learning tasks. The original DQN works in discrete space, and DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy. In order to do better exploration, an exploration policy μ' is constructed by adding noise $\mathcal{N}$:

$$\mu'(s) = \mu_{\varrho}(s) + \mathcal{N} \tag{70}$$

Moreover, the **DDPG** algorithm integrates a sophisticated technique known as soft updates, or conservative policy iteration, to update the parameters of both the actor and critic networks. This revised methodology utilizes a small parameter, denoted as τ, which is much smaller than $1(\tau \ll 1)$.

The soft update equation is formulated as

$$\varrho' \leftarrow \tau\varrho + (1-\tau)\varrho' \tag{71}$$

It guarantees that the target network values alter gradually over time, unlike the approach employed in DQN, where the target network remains static for a fixed period.

3.4. Working of DDPG Algorithm

Algorithm 1: Deep Deterministic Policy Gradient

Randomly initialize critic network $Q(s,a|\varrho^Q)$ and actor $\mu(s|\varrho^{\mu}$ with weights ϱ^Q and ϱ^{μ};
Initialize target network Q' and μ' with weights $\varrho^{Q'} \leftarrow \varrho^Q, \varrho^{\mu'} \leftarrow \varrho^{\mu}$;
Initialize replay buffer $\mathbb{R}$;
for *episode* $= 1$ *to* M **do**
  initialize a random process $\mathcal{N}$ for action exploration;
  Receive initial observation state s_1;
  for $t = 1$ *to* T **do**
    Select action $a_t = \mu(s_t|\varrho^{\mu}) + \mathcal{N}_t$ according to the current policy and exploration noise;
    Execute action a_t and observe reward r_t and observe new state $s_t + 1$;
    Store transition $(s_t, a_t, r_t, s_t + 1)$ in R;
    Sample a random mini-batch of N transitions $(s_t, a_t, r_t, s_t + 1)$ from R;
    Set $y_i = r_i + \gamma Q'(s_i + 1, \mu'(s_i + 1|\varrho^{\mu'})|\varrho^{Q'})$;
    Update critic by minimizing the loss:$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\varrho^Q))^2$;
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\varrho_{\mu}} O \approx \frac{1}{N} \sum_i \nabla_a Q(s,a|\varrho^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\varrho\mu} \mu(s|\varrho^{\mu})|_{s_i}$$

    Update the target networks:

$$\varrho^{Q'} \leftarrow \tau\varrho^Q + (1-\tau)\varrho^{Q'}$$

$$\varrho^{\mu'} \leftarrow \tau\varrho^{\mu'} + (1-\tau)\varrho^{\mu'}$$

  end
end

 doi:10.20944/preprints202501.1917.v1

4. Results And Discussions

4.1. Software Configuration

The deep reinforcement learning agent was trained using Python within a Jupyter Notebook on a Linux Ubuntu 20.04 operating system. The training process spanned approximately 5.415 hours on a modest hardware configuration consisting of an Intel graphics card, 8 GB of RAM, and a 1.9 GHz processor.

4.2. Hyper Parameter Selection and Initial Search

Parameters given in Table 2 are selected in this work.

Table 2. Parameters and Hyper Parameters.

Parameter	Value
Policy	MultiInputPolicy
Replay buffer class	HerReplayBuffer
Verbose	1
Gamma	0.95
Tau (τ)	0.005
Batch size	512,1024,2048
Buffer size	100000
Replay buffer kwargs	rb kwargs
Learning rate	1e-3 , 2e-4
Action noise	Normal action noise
Policy kwargs	Policy kwargs
Tensorboard log	Log path

4.2.1. Batch Size Comparison

Upon completing the training process, it was observed that there was a minimal disparity in the success rate, Figure 2, and cumulative reward, Figure 3, achieved across the different batch sizes. Also, the decrease in critic loss values, Figure 4 and actor loss values, Figure 5 indicates an improvement in the actor and critic networks' ability to approximate the optimal policy and value functions. Although the success rate and cumulative reward were similar, the enhanced convergence demonstrated by the lower losses in the 2048 batch size suggests a more efficient learning process and a potentially higher quality of learned policies.

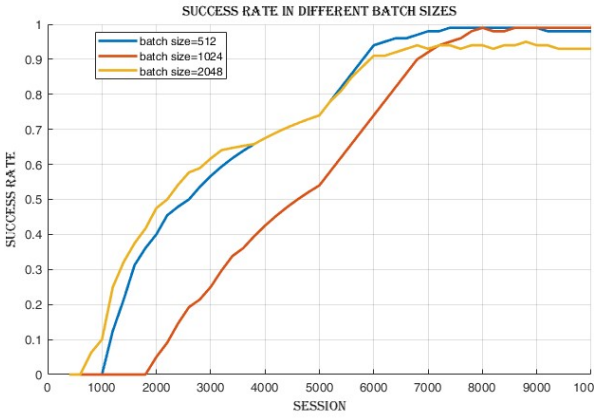

Figure 2. Success Rate In Different Batch Sizes.

 doi:10.20944/preprints202501.1917.v1

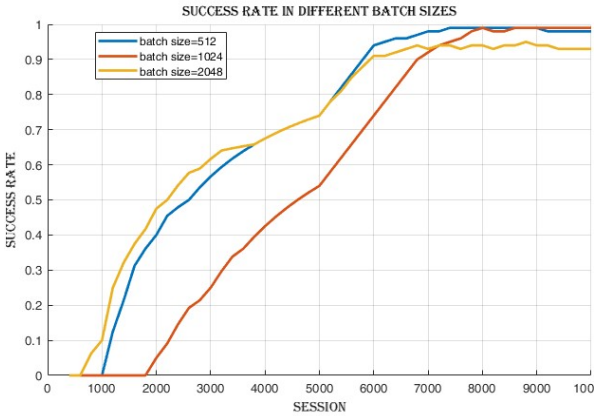

Figure 3. Cumulative Mean Reward In Different Batch Sizes.

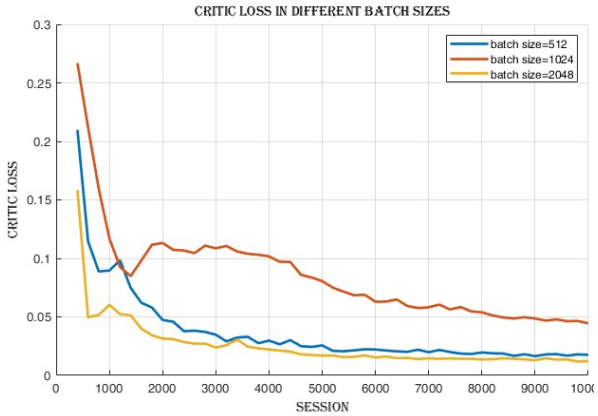

Figure 4. Critic Loss In Different Batch Sizes.

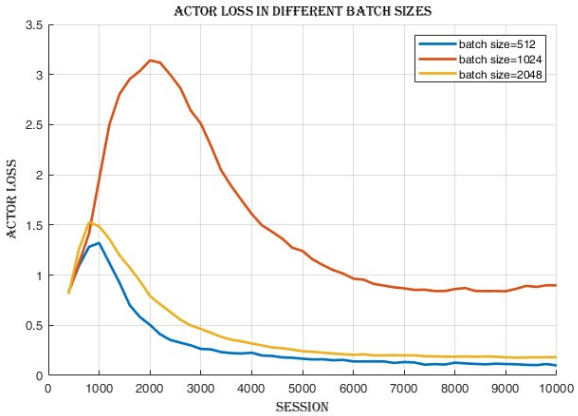

Figure 5. Actor Loss In Different Batch Sizes.

 doi:10.20944/preprints202501.1917.v1

Figure 6. Frames per Second In Different Batch Sizes

4.2.2. Learning Rate Comparison

After the training process, observations, Figure 8 revealed that there was a minimal disparity in the cumulative reward and success rate achieved between the two learning rates as shown in Figure 7. However, the learning rate of 2e-4 displayed slightly superior performance compared to 1e-3 in terms of success rate. Conversely, when using the learning rate of 1e-3, a notable decrease in actor loss, Figure 9 and critic loss, Figure 10 was observed, indicating improved policy and value estimation by the agent. Despite comparable cumulative reward and success rates, the reduced losses at the learning rate of 1e-3 signify enhanced convergence and a potentially more efficient learning process, suggesting the agent may have acquired higher-quality learned policies.

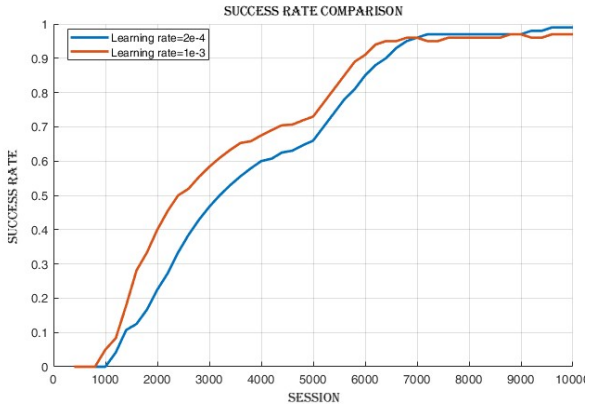

Figure 7. Success Rate In Different Learning Rate

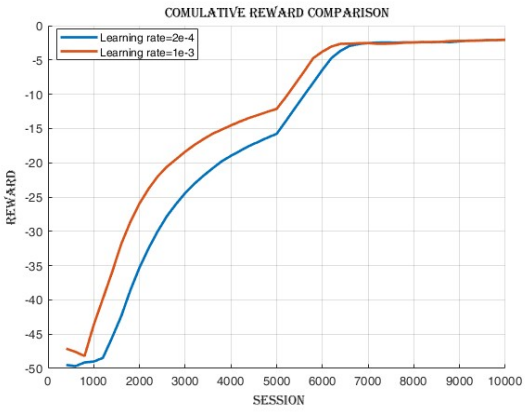

Figure 8. Cumulative Mean Reward In Different Learning Rate

 doi:10.20944/preprints202501.1917.v1

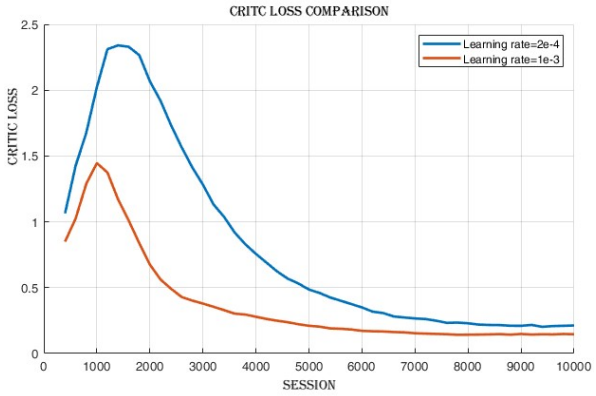

Figure 9. Actor Loss In Different Learning Rates

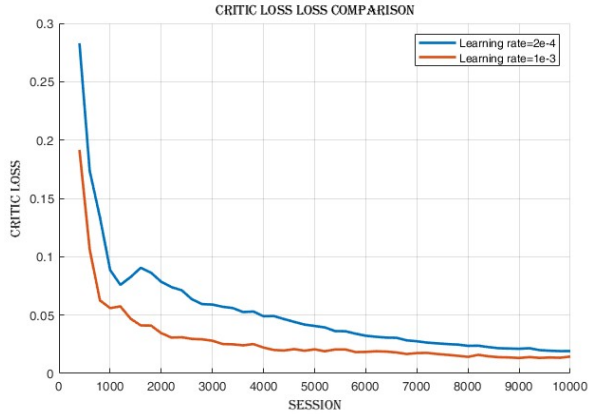

Figure 10. Critic Loss In Different Learning Rates.

4.3. Selection of Optimal Hyperparameters and Extended Training of DDPG Agent

After comparing the hyperparameters, as depicted in the preceding figures, and conducting a thorough analysis of the associated results, the selection of hyperparameters with promising performance was undertaken. Following this, an extended training phase was initiated, encompassing 100,000 time steps. This extended training phase serves as the fundamental training stage, which will be elaborated upon in the subsequent section.

Table 3. Parameters and Selected Hyper parameters.

Parameter	Value
Policy	MultiInputPolicy
Replay buffer class	HerReplayBuffer
Verbose	1
Gamma	0.95
Tau (τ)	0.005
Batch size	2048
Buffer size	100000
Replay buffer kwargs	rb kwargs
Learning rate	1e-3
Action noise	Normal action noise
Policy kwargs	Policy kwargs
Tensorboard log	Log path

 doi:10.20944/preprints202501.1917.v1

Table 4. Training Metrics at 200 time step and at 100,000 time step.

Category	Value	Category	Value
rollout/		**rollout/**	
Episode length	50	Episode length	50
Episode mean reward	-49.2	Episode mean reward	-1.8
Success rate	0	Success rate	1
time/		**time/**	
Episodes	4	Episodes	2000
FPS	18	FPS	5
Time elapsed	10	Time elapsed	19505
Total timesteps	200	Total time steps	100000
train/		**train/**	
Actor loss	0.625	Actor loss	0.0846
Critic loss	0.401	Critic loss	0.00486
Learning rate	0.001	Learning rate	0.001
Number of updates	50	Number of updates	99850

4.3.1. Improvement in Cumulative Reward and Success Rate

The mean episode reward, Figure 11 improves from -49.2 in the first loop to -1.8 in the last loop. The success rate, Figure 12 increases from 0 in the first loop to 1 in the last loop.

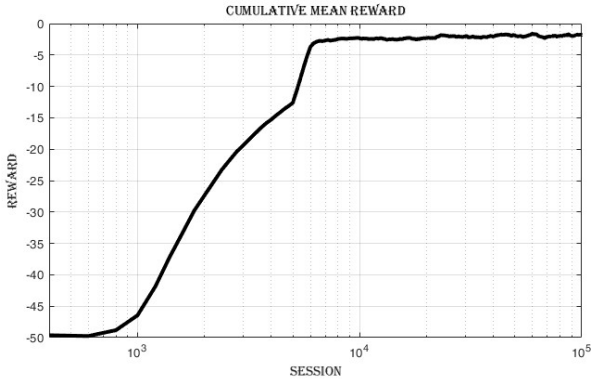

Figure 11. Improved Cumulative Mean Reward

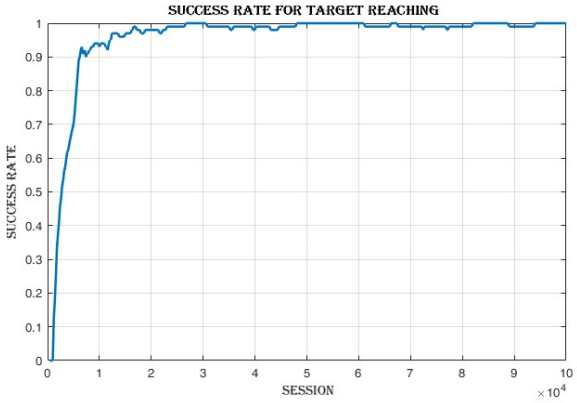

Figure 12. Improved Success Rate

4.3.2. Frames per Second (FPS)

The training speed, Figure 13 decreases from 18 frames per second (FPS) in the first loop to 5 FPS in the last loop.

 doi:10.20944/preprints202501.1917.v1

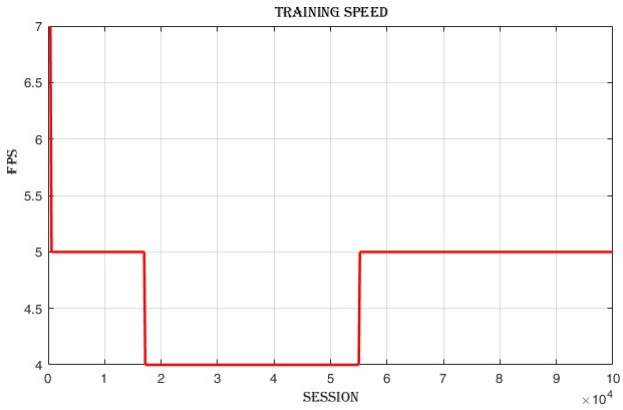

Figure 13. Frames Per Second (FPS) or Training Speed

4.3.3. Improvement in Actor and Critic Losses

The actor loss, Figure 14 decreases from 0.625 in the first loop to 0.0846 in the last loop. The critic loss, Figure 15 decreases from 0.401 in the first loop to 0.00486 in the last loop.

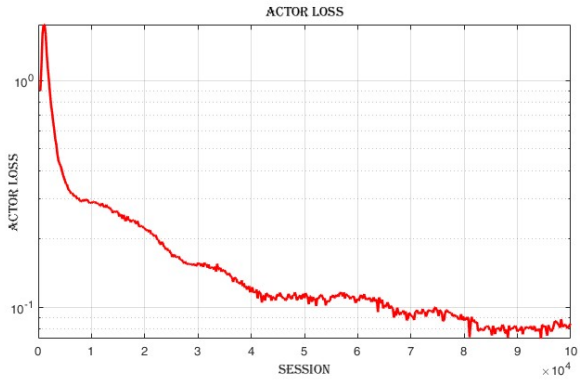

Figure 14. Improved Actor Loss

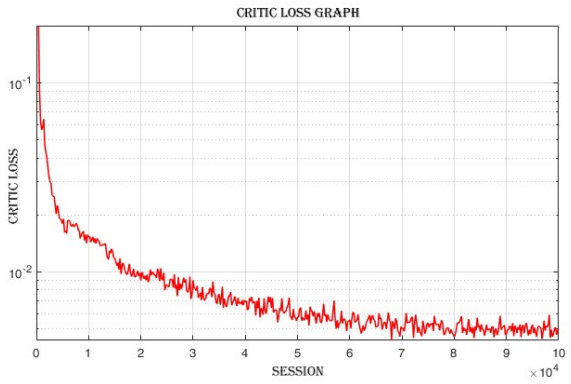

Figure 15. Improved Critic Loss

4.4. Comparing DDPG and PPO: Off-Policy vs. On-Policy Reinforcement Learning Algorithms

Proximal policy optimization (PPO), an on-policy reinforcement learning algorithm, was trained to compare its performance with Deep deterministic policy gradient (DDPG), an off-policy algorithm, as shown in Figure 16 and Figure 17. The cumulative reward achieved by DDPG was -1.8, whereas the cumulative reward obtained by PPO was -50 as shown in Figure16. The results of this comparison

 doi:10.20944/preprints202501.1917.v1

indicate that, in this particular scenario, DDPG exhibited superior performance over PPO in terms of cumulative reward.

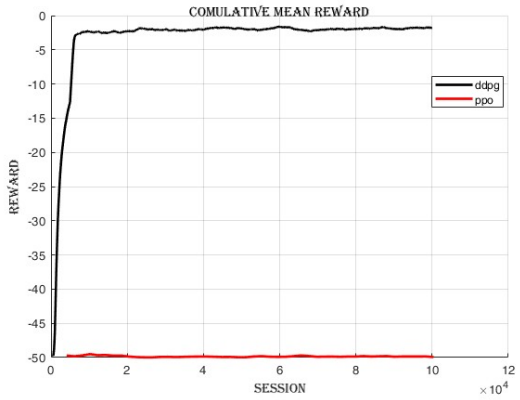

Figure 16. Cumulative Mean Reward.

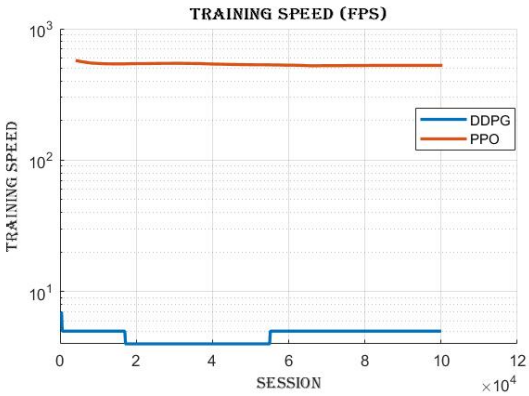

Figure 17. Training Speed.

5. Conclusion

In this study, the Deep Deterministic Policy Gradient (DDPG) algorithm is applied to train a robotic arm manipulator, specifically the Franka Panda robotic arm, for a target-reaching task. The objective of this task is to enable the robotic arm to accurately reach a designated target position. The DDPG algorithm is chosen because of its effectiveness in continuous control tasks and its ability to learn policies with high-dimensional action spaces. By leveraging a combination of deep neural networks and actor-critic architecture, DDPG approximates the optimal policy for the robotic arm. When comparing the performance of PPO and DDPG after training for $100,000$ time steps:

PPO achieved a mean episode reward of -50 indicating that the agent struggled to achieve positive rewards on average. Despite training at a relatively fast speed of $561FPS$, the results suggest that PPO faced challenges in finding successful strategies for the given task.

On the other hand, DDPG demonstrated superior performance with a mean episode reward of -1.8. It achieved a success rate of 1, indicating consistent success in reaching desired outcomes. Despite a slower training speed of $5FPS$. DDPG showcased its capability to effectively learn and improve its policy over time. Based on these results, DDPG outperformed PPO in terms of cumulative reward and success rate in the given scenario.

Author Contributions: The authors contributions in this manuscript are stated as follows: Conceptualization, L.H. and Y.A.; methodology, L.H.; software, L.H.; validation, A.T., Y.S.J. and S.J.; formal analysis, L.H.; investigation, A.T., Y.S.J.; resources, L.H.; data curation, L.H.; writing—original draft preparation, L.H.; writing—review and editing, Y.A., and A.T., and S.J.; visualization, A.T. and L.H.; supervision, Y.A. and S.J.; project administration, S.J.; funding acquisition, S.J. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the project for Smart Manufacturing Innovation R&D funded Korean Ministry of SMEs and Startups in 2024(Project No.RS-2024-00434311).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable'

Data Availability Statement: All the required data in this work are available with the authors and they can be provided upon request.

Acknowledgments: This work was supported by the project for Smart Manufacturing Innovation R&D funded Korean Ministry of SMEs and Startups in 2024(Project No.RS-2024-00434311).

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Mohsen, S., Behrooz, A., Roza, D. Artificial intelligence, machine learning and deep learning in advanced robotics, a review. Cognitive Robotics, 2023;3;54-70.
2. Sridharan, M. and Stone, P. Color Learning on a Mobile Robot: Towards Full Autonomy under Changing Illumination. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2007;2212-2217.
3. Xinle, Y., Minghe Sh., Lingling Sh. Adaptive and intelligent control of a dual-arm space robot for target manipulation during the post-capture phase. Aerospace Science and Technology, 2023;142;108688.
4. Abayasiri, R. A. M., Jayasekara, A. G. B. P., Gopura, R. A. R. C., Kazuo K. Intelligent Object Manipulation for a Wheelchair-Mounted Robotic Arm. Journal of Robotics, 2024. CrossRef
5. Mohammed, M. A., Hui, L., Norbert, St., Kerstin Th. Intelligent arm manipulation system in life science labs using H20 mobile robot and Kinect sensor. 2016 IEEE 8th International Conference on Intelligent Systems (IS), Sofia, Bulgaria, 2016. CrossRef
6. Yoshiyuki Ohmura and Yasuo Kuniyoshi. Humanoid robot which can lift a 30kg box by whole body contact and tactile feedback. In 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, CA, USA, 2007;1136–1141. CrossRef
7. Li, Zh., Ming, J., Dewan, F., Hossain, M, A. Intelligent facial emotion recognition and semantic-based topic detection for a humanoid robot. Expert Systems with Applications, 2013;40(13);5160–5168.
8. Martin, J. G., Muros, F. J., Maestre, J. M., and Camacho, E. F. Multi-robot task allocation clustering based on game theory. Robotics and Autonomous Systems, 2023;161:104314.
9. Nguyen, M. N. T. and Ba, D. X. A neural flexible PID controller for task-space control of robotic manipulators. Frontiers in Robotics and AI, 2023;9:975850.
10. Laurenzi, A., Antonucci, D., Tsagarakis, N. G., and Muratore, L. The XBot2 real-time middleware for robotics. Robotics and Autonomous Systems, 2023;163:104379.
11. Zhang, L., Jiang, M., Farid, D., and Hossain, M. A. Intelligent Facial Emotion Recognition and Semantic-Based Topic Detection for a Humanoid Robot. Expert Systems with Applications, 2013;40(13);5160-5168.
12. Floreano, D. and Wood, R. J. Science, Technology, and the Future of Small Autonomous Drones. Nature, 2015;521(7553);460-466.
13. Chen, T. D., Kockelman, K. M., and Hanna, J. P. Operations of a Shared, Autonomous, Electric Vehicle Fleet: Implications of Vehicle & Charging Infrastructure Decisions. Transportation Research Part A: Policy and Practice, 2016;94;243-254.
14. Chen, Z., Jia, X., Riedel, A., and Zhang, M. A Bio-Inspired Swimming Robot. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014;2564-2564.
15. Ohmura, Y. and Kuniyoshi, Y. Humanoid Robot Which Can Lift a 30kg Box by Whole Body Contact and Tactile Feedback. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007;1136-1141.
16. Kappassov, Z., Corrales, J.-A., and Perdereau, V. Tactile Sensing in Dexterous Robot Hands. Robotics and Autonomous Systems, 2015;74;195-220.

17. Arisumi, H., Miossec, S., Chardonnet, J.-R., and Yokoi, K. Dynamic Lifting by Whole Body Motion of Humanoid Robots. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008;668-675.
18. Aryslan, M., Yevgeniy, L., Troy, H., Richard, P. A deep reinforcement-learning approach for inverse kinematics solution of a high degree of freedom robotic manipulator. Robotics, 2022;11(2);44.
19. Serhat O., Enver T., Erkan Z. Adaptive Cartesian space control of robotic manipulators: A concurrent learning based approach. Journal of the Franklin Institute, 2024;361(6);106701.
20. Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, 1996;4;237-285.
21. Fadi, Al., Katarina Gr. Reinforcement learning algorithms: An overview and classification. In 2021 IEEE Canadian Conference on Elec-trical and Computer Engineering (CCECE), 2021;1–7.
22. Thrun, S. and Littman, M. L. Reinforcement Learning: An Introduction. AI Magazine, 2000; 21(1);103-103.
23. Smruti Amarjyoti. Deep reinforcement learning for robotic manipulation-the state of the art. arXiv preprint arXiv:1701.08878, 2017. CrossRef
24. Jens, K., Bagnell, J A., Jan, P. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 2013;32(11);1238–1274.
25. Tianci, G. Optimizing robotic arm control using deep Q-learning and artificial neural networks through demonstration-based methodologies: A case study of dynamic and static conditions. Robotics and Autonomous Systems, 2024;(181);104771.
26. Andrea, Fr., Elisa, T., Nicola C., Stefano, Gh. Robotic Arm Control and Task Training through Deep Reinforcement Learning. arXiv:2005.02632v1 [cs.RO] 6 May 2020.
27. Jonaid, Sh., Michael, Sch., Karl, M. Optimizing Deep Reinforcement Learning for Adaptive Robotic Arm Control. arXiv:2407.02503v1 [cs.RO] 12 Jun 2024.
28. Roman, P., Jakub, K., Radomil, M., Martin, J. Deep-Reinforcement-Learning-Based Motion Planning for a Wide Range of Robotic Structures. Computation 2024, 12(6), 116.
29. Wanqing, X., Yuqian, L., Weiliang, X., Xun, X. Deep reinforcement learning based proactive dynamic obstacle avoidance for safe human-robot collaboration. Manufacturing Letters, 2024;(41); 1246-1256.
30. Franka Emika Documentation. Control Parameters Documentation, 2024. Available at: CrossRef.
31. Weng, L. Policy Gradient Algorithms. Lil'Log, 2018. Available at: https://lilianweng.github.io/posts/2018-04-08-policy-gradient. [Accessed: 2024]