Robotic Arm Pick-and-Place with Deep Reinforcement Learning

by

Mengyang Liu

A thesis submitted to the School of Computing
in conformity with the requirements for the course
CISC 500 - Undergraduate Thesis

Queen's University
Kingston, Ontario, Canada
April 2024

Abstract

Robotic arms have the advantage of executing repetitive and precise tasks. However, when it comes to situations that are complicated, stochastic or requires adaptability, traditional methods often find their limitations. Conversely, Deep Reinforcement Learning (DRL), a subset of machine learning, is a powerful technique in solving complex tasks that demands flexibility and learning from interaction. Adopting DRL on robotic arms is a method to enhancing robotic capabilities beyond constraints of traditional methods. This research specifically targets the enhancement of robotic arm control for the fundamental pick-and-place task, aiming to develop a DRL-based program that enables robotic arms to autonomously adapt and perform in dynamic environments, bridging traditional automation with intelligent, adaptive control.

In this thesis project, a Reinforcement Learning (RL) environment is defined for the pick-and-place task, with safety conditions considered. It is also implemented in Gazebo simulation environment and encapsulated into a ROS package. In addition, multiple agent training experiments were conducted. Throughout the experiments, the TD3 algorithm and Curriculum Learning (CL) are found to be critical in training, while the over-estimation bias is found to be a severe issue. For the performance of the best trained agent, has a success rate of 95% in pick-and-place testing trials, over a $57\,\text{cm} \times 40\,\text{cm}$ rectangular area on a table. Finally, the best agent is deployed

on a real robotic arm and can succeed in pick-and-place under a different setting, indicating the approach described in the thesis is reasonable.

Contents

List of Tables

List of Figures

List of Acronyms

CL Curriculum Learning.

DDPG Deep Deterministic Policy Gradient.

DL Deep Learning.

DOF Degrees of Freedom.

DQN Deep Q-network.

DRL Deep Reinforcement Learning.

IK Inverse Kinematics.

MDP Markov Decision Process.

RL Reinforcement Learning.

ROS Robot Operating System.

TD3 Twin Delayed Deep Deterministic Policy Gradient.

Chapter 1

Introduction

This section gives an overview on robotic arms and Deep Reinforcement Learning (DRL), as well as the motivation of the project. The contribution and the organization of the thesis are also presented.

1.1 Overview

Robotic arms are mechanical devices that mimic human arms in structure and movement, which plays a vital role in various scenarios. They offer several advantages over human beings when it comes to task execution, for examples, they can perform repetitive and physically demanding tasks with great precision and consistency, reducing the risk of human errors and injuries. However, robotic arms face challenges in learning tasks that require adaptability and flexibility. Unlike humans, robots may struggle to adapt quickly to unforeseen changes in their environment or to perform tasks that were not explicitly programmed into their control systems.

On the other hand, Deep Reinforcement Learning (DRL) is a powerful machine learning approach that combines Reinforcement Learning (RL) and Deep Learning (DL). It designs policies for agents to follow, maximizing cumulative rewards from the

environment, and utilizes artificial neural networks to approximate complex functions with high-dimensional input. DRL has demonstrated its effectiveness in various fields, such as game plays, natural language processing and human-machine interactions. In the context of robotic arms, where tasks often demand adaptability and flexibility, DRL can offer a viable solution. By leveraging its ability to learn from interactions with the environment and adapt to unforeseen changes, DRL empowers robotic arms to overcome limitations in traditional programming and efficiently handle tasks that may not have been explicitly pre-programmed, bridging the gap between human-like adaptability and robotic automation.

In robotic arm control, one critical and fundamental problem is called pick-and-place. Robotic arm pick-and-place refers to the task that first asks the arm to pick up an object and then place it to a designated location. The project goal is to come up with a program that can guide the robotic arm to complete pick-and-place, using DRL.

1.2 Motivation

A DRL agent that can perform pick-and-place has a wide range of applications. In daily life, such an agent could be integrated into household robots, assisting with chores such as organizing items or preparing meals, as well as providing support to individuals with physical limitations. In manufacturing, they could be employed in assembly lines to pick parts and place them correctly, increasing efficiency and reducing human error. In addition, DRL agents are flexible and can be easily adapted to more complex tasks than pick-and-place, such as assembling electronics and furniture.

1.3 Contribution

The contribution of the thesis includes a developed Robot Operating System (ROS) package for the simulation environment, which is uploaded on the Github[1]. This package also contains the best performed pick-and-place policy from training. The thesis also shown that training the Sawyer robotic arm in Gazebo simulation environment, and then switch to the real robotic arm is a feasible approach.

1.4 Organization of Thesis

The remaining part of the thesis will be organized as follows: Chapter 2 goes through some background knowledge, including traditional methods for robotic arm control, DRL algorithms and knowledge transfer techniques. Chapter 3 formulates the project goal in detail and briefly introduces the software used in this project. Chapter 4 discusses how the RL environment is defined and details about training. Chapter 5 presents and discusses the experimental results. Chapter 6 concludes the thesis with a summary and future work.

[1]The link for the package is: `https://github.com/Mengyang-Liu6666/gym_sawyer`

Chapter 2

Background

In this section, we will first briefly review traditional methods for robotic arm control and outline its benefits and drawbacks. Then, we will describe the framework of Deep Reinforcement Learning (DRL) in a few words. Finally, we will take a closer look at some DRL algorithms, as well as knowledge transfer techniques that are related to this project.

2.1 Traditional Methods for Robotic Arm Control

Traditional methods for robotic arm control here refers to approaches that are in the absence of machine learning. Here, we will focus on Inverse kinematics and path planning as they are essential problems and are ubiquitous in robotic arm tasks.

2.1.1 Inverse Kinematics

Inverse Kinematics (IK) in the context of a robotic arm refers to the problem of finding the pose for a robotic arm so that the arm can reach a specific coordinate in the working environment. Poses, in the context of robotic arms, refer to a vector of angles of each joint. If the end-effector is added into consideration, poses also an unit

quaternion, an extension of complex numbers to 4D with the form $a + bi + cj + dk$, to represent the orientation of the end-effector. Forward kinematics, on the other hand, refers to the problem of finding the coordinate that the robotic arm reaches when the pose is given.

Forward kinematics problems always have a unique solution, which can be found by substituting the pose to an equation. However, such guarantee is not presented in IK. In some cases, there can be multiple solutions due to the high Degrees of Freedom (DOF) of the robotic arm, while in others, there is no solution for IK, such as when the coordinate is out of reach or blocked by obstacles [5].

Traditionally, there are two approaches to IK: analytical and numerical.

In the analytical approach, IK is solved by finding the closed-form solution. However, for different robotic arms and under various environments, the closed-form solution should be redetermined. Additionally, when we have more degrees of freedom than the dimension of the environment, this approach may not work effectively [5]. Therefore, the analytical solution suffers from limited adaptability in different scenarios.

In the numerical approach, the solution process starts with an arbitrary pose. Then, we use forward kinematics to find the error, which is the distance between the coordinate resulting from this pose and the target coordinate. After that, we repeatedly apply numerical optimization methods, such as gradient descent, to improve the pose until the error is small enough to be considered satisfactory. However, the iterative nature of this approach makes it computationally expensive. Additionally, it can get stuck at a local minimum and may not reach a satisfactory error, even if a solution for IK exists [8].

2.1.2 Path Planning

In addition to IK, a robotic arm requires planning a sequence of actions to reach a pose. The actions at each time step can represent the angle of each joint to turn. Traditionally, path planning is treated as finding the shortest path in a weighted graph and is solved using algorithms such as A*. However, this approach becomes computationally expensive when the search space is large or continuous.

Later, sampling-based algorithms were proposed to address the path planning problem, such as rapidly-exploring random trees (RRT) and RRT* [6]. The basic idea behind these algorithms is to keep track of a set of collision-free poses and a graph that connects reachable poses with a sequence of actions. The algorithm iteratively samples poses from all collision-free poses and checks their reachability. It terminates when the goal pose is found to be reachable. However, due to their sampling nature, these algorithms are time-consuming and may not be suitable for real-time planning [1].

2.2 Deep Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning that involves an agent to learn make decisions through trial-and-error. Formally, it is used to solve Markov Decision Process (MDP) which includes states s, containing the information about the environment, and actions a, the decisions that can make under a state. After an action is made, a reward will be given by the reward function $r(s_t, a_t)$ and the state will be updated by a transition kernel $p(s_{t+1}|s_t, a_t)$. The goal of RL is to find a policy $\pi(a|s)$, a function that takes in state and output an action, that maximizes the total reward. A very important concept in RL is the action-value function $Q_\pi(s, a)$,

also known as Q-values. It is the expected cumulative reward the agent can get after executing action a under state s, if the policy it uses is π. One RL algorithm is called tabular Q-learning, which starts with an arbitrary policy and iteratively estimate its action-value function, store it into a table and improve the policy. Regehr and Ayoub [13] give a proof that shows tabular Q-learning is guaranteed to converge to optimal policy.

Deep Learning (DL) is a subset of machine learning that uses neural networks to model and understand complex patterns in the provided data. Neural networks are computational models consisting of layers of interconnected nodes. Each node receives input, processes it according to its own set of parameters and passes the output to the next layer. Parameters of neural networks are typically updated by an algorithm called gradient descent. The formula is given by: $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$, where θ is the parameters of neural network, $L(\theta)$ is the loss function we want to minimize and α is the learning rate.

Deep Reinforcement Learning (DRL) uses neural networks to approximate the policy or its Q-value function. Representing policies or Q-value function by tables may not be practically possible, as the state or action space can be too large or uncountable. Approximating them by neural networks can significantly reduce the number of parameters used than tables. In practice, neural networks are preferred over other statistical learning methods as they can vary in complexity and being able to be updated by gradient descent. It is true that in many cases, DRL algorithms cannot converge to the optimal policy, as is it possible that optimal policy cannot be perfectly approximated by a neural network, many obtained sub-optimal policies by DRL are sufficiently good in practice.

2.3 Deep Reinforcement Learning Algorithms

In this section, we will take a look at two DRL algorithms that were introduced in sequence after Deep Q-network (DQN) [9], as well as hindsight experience replay, which is often used together with many DRL algorithms. DQN uses neural networks to approximate the Q-values and addresses the moving target problem, as well as sample correlation problem, that are arise from using function approximations. An limitation of DQN is that it can only be applied when the state space is discrete. Both of the DRL algorithms mentioned in this section handle continuous state spaces.

2.3.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [7] is an off-policy actor-critic algorithm that is used for environments with continuous action spaces. The critic network approximates the Q-values and the actor network approximates the action that maximizes the critic network. The algorithm is called "deterministic" since the actor only outputs the action, instead of a distribution over the action space. Inspired by DQN, DDPG also uses a replay buffer to avoid the critic from over-fitting to similar states and target networks for both critic and actor to stabilize their updates.

Below is the pseudo-code for DDPG. Similar to DQN, the online critic is updated to minimize the squared TD-error, where the bootstrapping uses the target networks. However, unlike the discrete case that DQN is dealing with, where we can take $\gamma \max_{a'} Q_{\theta'}(s_{i'}, a')$ for bootstrapping, the action space is continuous so we use the output of the target actor network instead. Similar to actor-critic algorithms, the actor is updated to maximize the Q-value using gradient descent. To find the gradient of Q-values over parameters of the online actor, we can apply the chain-rule

in calculus. For the exploratory policy to collect episodes, the original work, Lillicrap et al. [7], samples a noise and adds it to the output of the online actor.

Algorithm 1: Deep Deterministic Policy Gradient

initialization

Online critic networks: $Q_\theta(s, a)$

Online actor network: $\pi_\phi(s)$

Target network parameters: θ', ϕ'

Learning rates: α, β.

for $m \leftarrow 1$ **to** M **do**

Collect $(s_i, a_i, r_i, s_{i'})$ using some policy with exploration noise

Add them into the replay buffer

for K *times* **do**

Uniformly sample a batch of $(s_i, a_i, r_i, s_{i'})$ from replay buffer

$y_i = r_i + \gamma Q_{\theta'}(s_{i'}, \pi_{\phi'}(s_{i'}))$; `// use targets`

$\theta \leftarrow \theta - \alpha \sum_i [(y_i - Q_\theta(s_i, a_i)) \nabla_\theta Q_\theta(s_i, a_i)]$

$\phi \leftarrow \phi + \beta \sum_i [\nabla_a Q_\theta(s_i, \pi_\phi(s_i)) \nabla_\phi \pi_\phi(s_i)]$

end

; `// Below can be replaced with Polyak update`

if $m \mod N$ **then**

Target update for critic: $\theta' \leftarrow \theta$

Target update for actor: $\phi' \leftarrow \phi$

end

end

2.3.2 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed Deep Deterministic Policy Gradient (TD3) [3], is a modified version of DDPG. TD3 reduces the variance of the policy update by updating the policy network less frequently than the critic network. It also introduces a clipped version of double Q-learning to further overcome the overestimation bias in the critic network. In addition, it also proposed a technique called target policy smoothing, which forces the critics to have similar Q-values for similar actions.

In their work [3], Fujimoto et al. mentioned that in actor-critic algorithms, the actor and critic network can exacerbate the poor performances of each other. That is, the critic network suffer from the overestimation bias when the actor is poor. Furthermore, since TD-learning is used, the overestimation from is accumulated from bootstrapping. Since the policy update requires the estimation of the state-value from the critic, the overestimating critic can the policy update inefficient and potentially harmful. Consequently, the policy can become worse after such updates. To mitigate this issue, they proposed 3 techniques and combined them together in TD3.

To improve the qualities of updates in actor, they proposed an idea of updating the actor when the critic is stable. In practice, they update the actor less frequently than the critic. Consequently, during the update of the actor, the critic is more stable and reliable, so that the actor is less likely to have harmful updates.

To mitigate the overestimation bias in the critic network, they use a clipped version of double Q-learning. In double Q-learning for discrete action space, Q_1 is updated with target $r + \gamma Q_2(s', \arg\max_a Q_1(s', a))$. In practice, if the policy is changing slowly, Q_1 and Q_2 will give very similar estimations to the Q-action values, which also means their errors in estimations are very similar. As a result, we may have

$Q_2(s', \arg\max_a Q_1(s', a)) > Q_1(s', \arg\max_a Q_1(s', a))$ and the overestimation bias is greater. Therefore, they propose to take minimum of 2 estimates, which is $y = r+\gamma \min_{j=1,2} Q_j(s', \pi_{\theta'}(s'))$. To reduce the computational cost, the two critic networks are updating with the same target y.

Since the actor is deterministic, $\min_{j=1,2} Q_j(s', \pi_{\theta'}(s'))$ is very susceptible to the predicted action, leading to a large variance in y. This effect is more obvious when the critic is not smooth, that is, the critic function has narrow peaks. To reduce the variance, they proposed target policy smoothing. This technique sets up the update target as $y = r + \gamma \min_{j=1,2} Q_j(s', \tilde{a})$, where $\tilde{a} = \pi_{\theta'}(s') + \varepsilon$ is an action sampled from the target actor $\pi_{\theta'}$, with noise ε added. The noise can come from sampling a normal distribution and clip it from a pre-defined bound. The use of sampled noise can be seen as a simulation of an on-policy update, since the data-collecting policy and policy used to update critic both contain noises. Target policy smoothing also requires the critics to have similar Q-values for actions that are close to the one predicted by the actor. As a result, the critic has less narrow peaks and the variance in the update target is reduced. Below is the pseudo-code for TD3.

Algorithm 2: Twin Delayed Deep Deterministic Policy Gradient

initialization

2 online critic networks: $Q_{\theta_1}(s,a)$, $Q_{\theta_2}(s,a)$

1 online actor network: $\pi_\phi(s)$

Target network parameters: $\theta_1', \theta_2', \phi'$

Learning rates: α, β, τ

Variance and bound of noise for target actor: σ^2, c

for $m \leftarrow 1$ **to** M **do**

Collect $(s_i, a_i, r_i, s_{i'})$ using some policy with exploration noise

Add them into the replay buffer

for K *times* **do**

Uniformly sample a batch of $(s_i, a_i, r_i, s_{i'})$ from replay buffer

Target policy smoothing:

$$\tilde{a} \leftarrow \pi_{\theta'}(s) + \varepsilon,\ \varepsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2), -c, c)$$

Update online critics with clipped double Q:

$$y_i = r_i + \gamma \min_{j=1,2} Q_{\theta_j'}(s_{i'}, \tilde{a})$$; `// use target critics`

$$\theta_1 \leftarrow \theta_1 - \alpha \sum_i [(y_i - Q_{\theta_1}(s_i, a_i)) \nabla_{\theta_1} Q_{\theta_1}(s_i, a_i)]$$

$$\theta_2 \leftarrow \theta_2 - \alpha \sum_i [(y_i - Q_{\theta_2}(s_i, a_i)) \nabla_{\theta_2} Q_{\theta_2}(s_i, a_i)]$$

; `// critics share` y_i

if $m \mod M$ **then**

Delayed update for policy:

$$\phi \leftarrow \phi + \beta \sum_i [\nabla_a Q_{\theta_1}(s_i, \pi_\phi(s_i)) \nabla_\phi \pi_\phi(s_i)]$$

Update target networks:

$$\theta_1' \leftarrow \tau\theta_1 + (1-\tau)\theta_1',\ \theta_2' \leftarrow \tau\theta_2 + (1-\tau)\theta_2'$$

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

end

end

end

2.4 Knowledge Transfer in Deep Reinforcement Learning

In this section, we will briefly discuss some transfer learning techniques. These are problem-specific techniques that allow us to reduce the time or resources used in training RL agents, by using existing domain knowledge. Knowledge can be in many forms, such as human expert knowledge, demonstrations, polices for similar tasks and a division of the target task into several simpler ones. Transfer learning is a technique of transferring knowledge from one task to boost the performances of an agent for another similar, but different task. In RL, training an agent from scratch can require a large amount of interaction with the environment, which sometimes is not even practically possible. For example, when training agents for a new robotic arm in reality to do the reaching task, the training time can be very long. However, if we have an existing robotic arm that has trained policies for reaching, we can reuse the trained policy to some extent, so that the training time can be shortened. Thus, transfer learning is a motivated topic in reinforcement learning [17].

2.4.1 Reward Shaping

Reward shaping is one transfer learning technique for reinforcement learning. Instead of directly using the reward from the environment, reward shaping is a technique that applies a transformation to the reward, making it more informative. Such transformations require external domain knowledge of the environment [17]. Potential based Reward Shaping (PBRS) requires a potential function Φ to evaluate states, which come from expert knowledge. Then, "shaped" reward $F(s, a, s')$ for a state transition is evaluated as: $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$ [12]. Another way of doing reward shaping can be utilizing an expert policy [16]. For example, given an expert

policy π, the "shaped" reward can be $F(s, a, \cdot) = \pi(a|s)$.

2.4.2 Curriculum Learning

Curriculum Learning (CL) refers to the technique of sequentially training an agent from simple tasks to difficult ones, so that the training is accelerated or the final performance is improved. Narvekar et al. [11] gives an overview of curriculum learning. To perform curriculum learning, a curriculum, representing the dependencies between tasks, is required, which definition relies on a directed acyclic graph. A common simplified definition, called sequence curriculum, is that the learning proceeds by learning through a sequence of tasks. In sequence curriculum, tasks are ordered from easy to hard in a list and the sequence of learning is from the first task to the last one. The last task is the one we want our agent to succeed.

One curriculum learning technique is called task generation. Task generation refers to the process of creating easier tasks for the agent to learn. Narvekar et al. [10] gives several methods for it. Task simplification refers to the method of reducing the complexity of the environment in the task directly. For example, generating easier tasks for chess may be to use a smaller chess board and/or to use less pieces. Promising initialization adds states that can lead to high reward states in few steps in initialization. For example, in chess, we initialize the board to those that have few steps to victory. Narvekar et al. [10] also suggests methods that rely on mistake detection. It is true that in many cases, we do not have the optimal policy so we cannot tell whether an action is a mistake or not. Instead, the definition of "mistake" is modified a bit and it includes 3 cases: the action that leads to an undesirable terminal state, the action that does not make a change in state and the action that

leads to a large, negative reward. Action simplification refers to the strategy that generates a task by pruning the actions that are most frequently making mistakes. Mistake learning adds states that the agent is about to make mistakes in initialization. Then, the agent is more likely to enter these states and learn to correct the mistakes.

Chapter 3

Problem Description

This section describes the goal of this project and problem specific details, such as information about the robotic arm, the simulation environment and the Robot Operating System (ROS).

3.1 Project Goal

The goal of this project is to come up with a policy for the robotic arm so that it can first pick up a block and then place it to a designated location. The location of the block and the location to place it are known, however, the locations cannot be outside a specified 57 cm × 40 cm rectangular region. In the simulation environment, the block is a cube with each side to be 4.5 cm. An existing IK is provided, but it is treated as a black box and it can return no solution for some conditions.

Additionally, safety requirements are also considered in this project. To make sure interacting it with human, a restriction on the velocity is used. On each of the 3 dimensions, the velocity cannot exceed 10 cm/s. To make sure other parts of the robotic arm are not moving to fast, a restriction on the angular velocity is applied to each joint of the arm. The angular velocity of each joint cannot exceed 90 °/s, which

takes 4 seconds to make one rotation.

3.2 Sawyer Robotic Arm

Sawyer robotic arm is a 7-DOF robotic arm, developed by the Rethink Robotics. It is also equipped with 2 cameras and a screen for displaying [14]. Below are pictures that shows the available angles for each joint of the arm can turn. For the first 4 joints (J0 to J3), it can turn up to 350°. For the next 2 joints (J4 to J5), it can turn up to 341° and the last joint can reach the whole 360°.

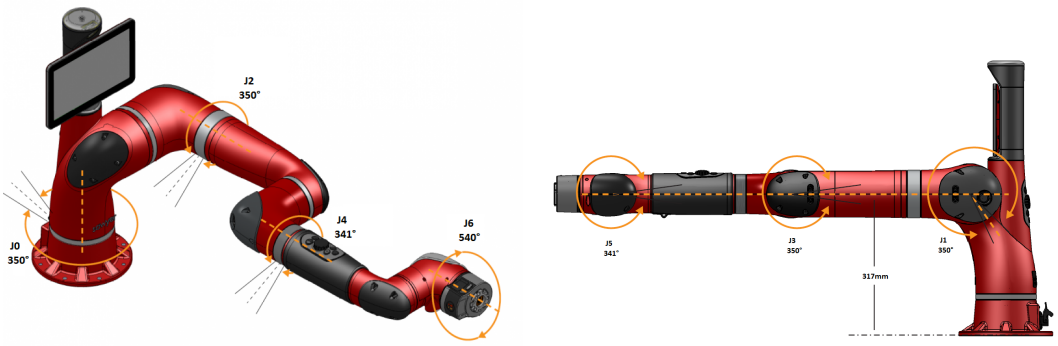

Figure 3.1: Joint limits of the Sawyer robotic arm.

Depending on the task, the robotic arm can be installed with different grippers. In the simulation environment, the gripper has 2 fingers and the max distance between them is 5.8 cm.

3.3 Gazebo Simulation Environment

Gazebo is a 3D simulation platform that provides a dynamic environment for testing robotics algorithms and designing robots, particularly when integrated with the Robot Operating System (ROS). It is good at simulating realistic physics laws.

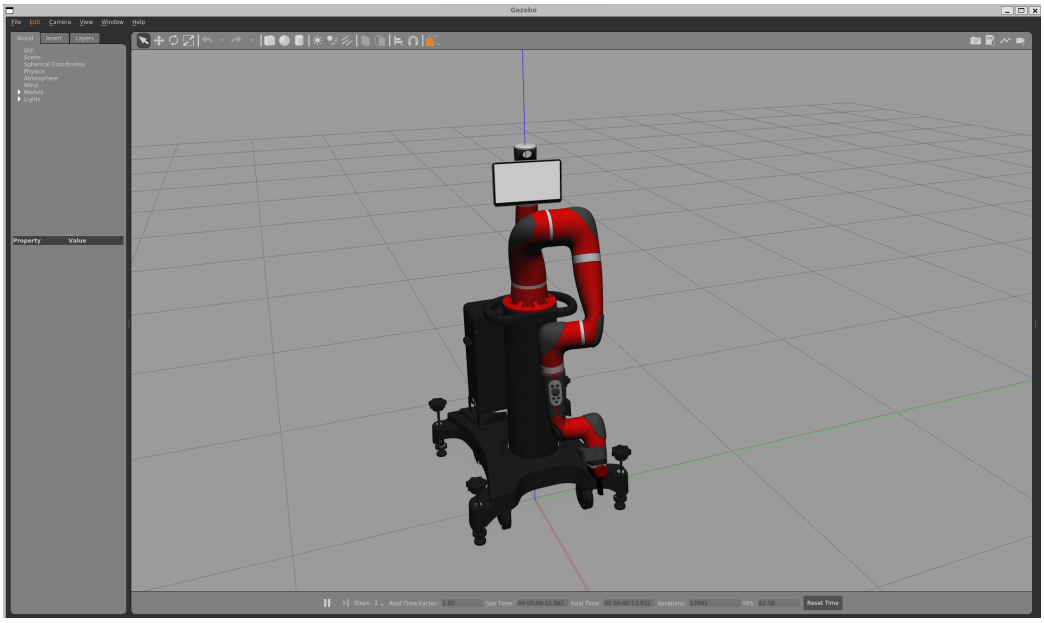

Figure 3.2: Sawyer robotic arm in Gazebo simulation environment

In Gazebo, robots and other objects are typically described using Universal Robot Description Format (URDF) models, which define the physical components of an object, including textures, shapes, masses and center of masses. Gazebo has a close integration with ROS, allowing for seamless transitions between simulated testing and real-world application.

3.4 Robotic Operating System

The Robot Operating System (ROS) is an open-source framework for robot software development. It provides an interface that allows communications and control over hardware, simulation environments, custom scripts, etc. In ROS, processes are represented by node, which are modular entities that perform computation. Nodes communicate with one another via messaging over topics, services, and actions, typically coordinating through a central node known as the master node.

Table 3.1: Table of ROS nodes and their description, during the initialization. Nodes for robotic arm IO are omitted for simplicity.

Node description	Node name
Master node (hidden)	'master'
Log collector	'rosout'
Gazebo simulation environment	'gazebo'
Gazebo simulation GUI	'gazebo_gui'
Block info publisher	'block_odom_publish_node', 'block_tf_publish_node'
UDRF model spawner	'urdf_spawner'

The process of launching the project demonstration can be seen an example of how ROS nodes interacts with each other. First, the master node is launched, followed by a node for Gazebo simulation environment and another node for its Graphical User Interface (GUI). Then, a URDF spawner node is launched, and terminates after spawning the robotic arm into the simulation environment. After that, 7 nodes in total are launched to handle the input/output (IO) of the robotic arm. Later, we launch another script to set up the interactable objects for pick and place. A block and a coffee table are again spawned by the URDF spawner. In addition, 2 nodes are launched to publish the information of the block. Finally, we run the demonstration script, which is also a node. Conceptually, the GUI, robotic arm IO, block information and the demo nodes are call contacted to the simulation node. However, in ROS, the master node is the intermediate of all communications.

Chapter 4

Methods

4.1 Reinforcement Learning Environment

This section explains how the RL environment is defined to solve the pick and place task. Also, the significance of such definition is discussed.

4.1.1 Procedure

The procedure to implement pick and place is broken into several steps. The first step is to initialize the robotic arm to the pose that has 0° at each joint and gripper opened. Then, we let the agent to control the arm, so that the gripper is 10 cm above the block. After that, we execute a programmed control to land down 10 cm, close the gripper and lift the gripper by 15 cm. Later, we invoke the agent again and let it move the gripper that is 10 cm above the location to place the block. Finally, we execute the same control, open the gripper and lift the gripper by 15 cm. One round of pick and place is finished and we can begin another round without the pose initialization step. Below is a diagram that shows the described procedure.

This procedure not only makes training easier, but also makes it possible to adapt

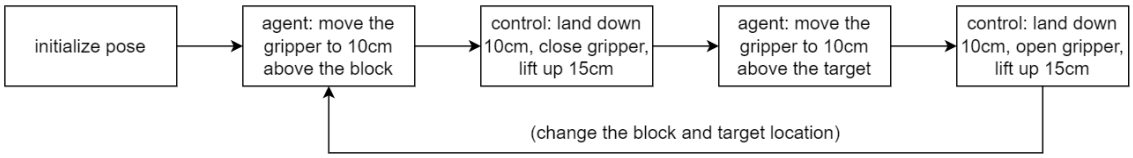

Figure 4.1: Sequence diagram for the pick-and-place procedure.

for different scenarios. Since the agent is only responsible to locate the target from above, there is no need to consider the collision of the block and the gripper. This saves the agent from learning to avoid the gripper pushes the block away. In addition, since the landing behavior is controlled by a programmed script, it is explainable and can be modified to satisfy different problem settings. However, if the locating and landing are both done by the agent, the procedure becomes inflexible as a change in problem setting may need to a retrain on the model.

4.1.2 State

The state is composed of three parts, the joint angles for each of the 7-DOF of the robotic arm, the location of the gripper and the location that is 10 cm above the target. All locations are with respect to the robotic arm.

The introduced 10 cm above the target brings a good property to the environment. When the gripper location equals to the location that is 10 cm above the target, the programmed control will be executed and this episode is guaranteed to success. This equality allows the environment to easily adapt into a goal environment, which supports algorithms such as Hindsight Experience Replay. Most neural networks are invariant to linear changes in individual features, since the linear change can be compensated by another weight and bias. Therefore, the introduced 10 cm will not have a large impact on training results at convergence.

The angles of each joint is important in addition to the gripper location. Since the robotic arm has a high DOF, there are potentially multiple poses that can reach the same location. Below is a diagram example to illustrate this. For the 2 poses, even though their end-effector locations are the same, their feasible locations to move their end-effectors next, which are shown by the green areas in the diagram, are different.

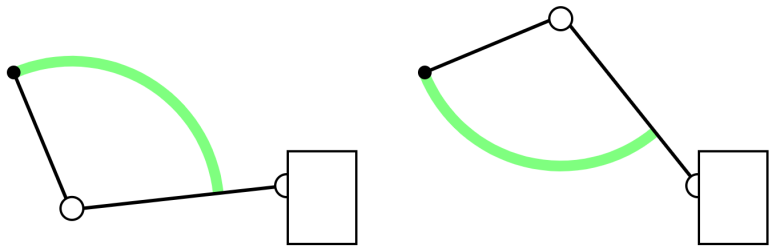

Figure 4.2: A diagram of a 2-DOF robotic arm, with same end-effector location but different joint angles.

Therefore, without providing the joint angles, there is no function to find the range of locations that can be reached in the next step. Consequently, this problem becomes a partially observed Markov Decision Process (MDP). To avoid this problem becoming partially observed, we need to provide the past trajectories of the end-effector and assume the initial location is fixed, which makes this problem non-Markovian. Thus, including the joint angles keeps this problem an MDP.

4.1.3 Action

The action is set to be the change in the gripper location. On each of the 3 dimension, the absolute value of action is limited to 2.5 cm. The actual movement on the robotic arm is handled by an internal IK, which is treated as a black box. The response rate is set to 4 Hz. Combining the action space and the response rate, we

satisfies the proposed safety constraint, which limits the velocity of the gripper to 10 cm/s.

Response rate refers to how frequently the robot can react to the environment. It varies a lot depending on the task. For pick and place, around 10 Hz will be preferred. However, considering the safety constraint, the action space will be smaller if 10 Hz is used for response rate, which can leads to numerical instability in IK.

4.1.4 Termination criterion and reward function

The termination criteria can be categorized into 4 types: "successfully located", "landed early", "invalid move" and "out of workspace". Before go to the description of each case, define landing_reward as the following:

$$\text{landing\_reward} = 150 - 160[\min\{d, 1\}]^2, d = \frac{||g_T - g_{\text{target}}||_2}{||g_0 - g_{\text{target}}||_2}$$

where g_{target} refers to the target location. g_0 and g_T are the gripper location at the initial state and final state in an episode respectively. d can be seen as the normalized final distance, between the gripper to the block. Furthermore, if $g_T = g_{\text{target}}$, that is, the gripper reached the block perfectly, we have landing_reward = 150. If the gripper did not move any closer to the target than the initial location, that is, $||g_T - g_{\text{target}}||_2 > ||g_0 - g_{\text{target}}||_2$, we have landing_reward = -10. We will always have landing_reward between 150 and -10.

"Successfully located" refers to the case where the gripper is 10 cm above the block and the gripper is close to the block in the horizontal plane. In this case, a reward of 400 + landing_reward is assigned. In addition, the control script will be invoked to land and pick up or place the block. During the picking, if the block is successfully

picked up, additional 200 reward is given. Otherwise, no further reward is given and the episode is terminated. During the placing, if the block placed close to the target location, an additional 200 reward is given. No matter the placing is accurate or not, the episode will be terminated.

"Landed early" refers to the case where the gripper is 10 cm above the block but the gripper is not close enough to the block in the horizontal plane. In this case, the episode will be terminated and a reward of landing_reward.

"Invalid move" includes 2 cases, no IK solution is found or one of the joints are attempting to move too fast. In either case, we have to terminate the episode. A reward of landing_reward $-$ 20 will be assigned.

"Out of workspace" means the gripper is outside a defined rectangular region. The gripper should not go outside of this region, not only when running pick-and-place tasks but also during learning. A reward of -100 will be assigned in this case.

Implicitly, an episode can be terminated when the max steps is reached. We can seen it as a reward of 0 at the last step. By running experiments, setting the max steps to 50 is large enough.

Step rewards are also given to encourage the agent for going close to the block. Specifically, let the location at time t to be g_t, the action at time t, which is the change in location, to be a_t, and the target location to be g_{target}. The step reward is set to be the following:

$$\text{step\_reward}(s_t, a_t) = 100c \cdot \frac{||g_t - g_{\text{target}}||_2 - ||g_t + a_t - g_{\text{target}}||_2}{||g_0 - g_{\text{target}}||_2}, c = 1.6$$

Without considering the coefficient c, every trajectory that eventually moves to the

target will ideally collect a reward of 100 from step_reward. This condition is based on the assumption of IK movement is perfect, meaning $g_t + a_t = g_{t+1}$. However, the IK movement has some inaccuracies so a coefficient of $c = 1.6$ is introduced. Below is a table to summarize the RL environment.

Table 4.1: Summary of the RL environment

State	joint angles for 7-DOF of the arm, gripper location, target location,
Action	change in gripper location
Termination criteria	gripper located the target, gripper landed too early, one of the joints attempts to move too fast, the action has no IK solution, gripper is outside of workspace

4.1.5 Reward shaping

The reward function above is defined to roughly satisfy a ranking on termination conditions. The best scenario is the gripper located the block successfully. A bit worse than the success is the gripper landed early, but still close to the target location. Following that can be the gripper is quite close to the target location, but terminated by an invalid move. In such trajectories, it is fair to say the issue is the final step, while other steps are quite good.

The remaining termination conditions are less favored. Landing early, but too far away from the target is worse than all cases mentioned above. Then we have the gripper is too far away from the target and terminated by an invalid move. Finally, we have the gripper is outside of the workspace. Below is an axis diagram to display

the range of rewards of different termination conditions for picking. The reward assignment during placing is almost identical to picking.

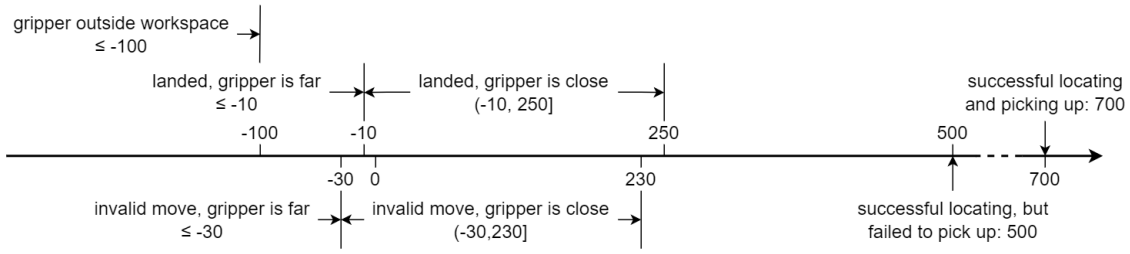

Figure 4.3: An axis diagram summarizing the rewards associated to different termination conditions.

4.2 Training Set Up

In this section, more information regarding how the training is proceeded. An overview of algorithm choices, hyper-parameter selections and the different ranges of locations to summon the block in different experiments.

4.2.1 Algorithm and hyper-parameter selection

Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3) are used since the action space is continuous. For algorithms that can only handle discrete action spaces, such as Deep Q-network (DQN) action discretization is required. For pick-and-place, action discretization is not favored since if the discretization is too precise, the action space will be too large. On the other hand, if the discretization is not precise enough, there might have no solution for picking or placing. Either case, the assumption that similar actions leads to similar transition and similar reward, is not utilized in most discretization techniques. Consequently, such techniques may be less sample efficient.

Below is a table of hyper-parameter settings. The unmentioned ones, such as target network updating setups, are using the default setting in Stable Baselines 3. Considering the simplicity and the time to run experiments, only one working set up is used in experiments.

Table 4.2: Summary of the hyper-parameters

Learning rate	0.001
Discount factor	0.99
Exploration noise	$\mathcal{N}(0_{3\times1}, 0.3^2 I_{3\times3})$
Neural network structure	densely connected $[50, 800, 600, 300]$
Replay buffer size	$1,000,000$
Collected transitions before learning	100
Policy update delay (TD3 only)	2
Target regularization noise (TD3 only)	$\mathcal{N}(0_{3\times1}, 0.2^2 I_{3\times3})$

4.2.2 Different ranges of the block location

To test if the environment, algorithms and hyper-parameters are feasible or not, an easier task is tried, comparing to the project goal. The easier keeps every set up the same, except a smaller range of blocks is adopted. This range is called "easy range", which is close to the initial gripper location in the xy-plane.

In addition, to "easy range", we denote the "testing range" to the range of block specified in the project goal, which has dimension $57\,\text{cm} \times 40\,\text{cm}$. Also, "training range" is introduced, which extends the "testing range" by $8\,\text{cm}$ in each of the 4 directions. A diagram of different ranges of the block is shown in the diagram below. From the diagram, we can see that "easy range" is entirely included in the "training

range" so Curriculum Learning (CL) can be applied by first training the agent on "easy range" and then "training range".

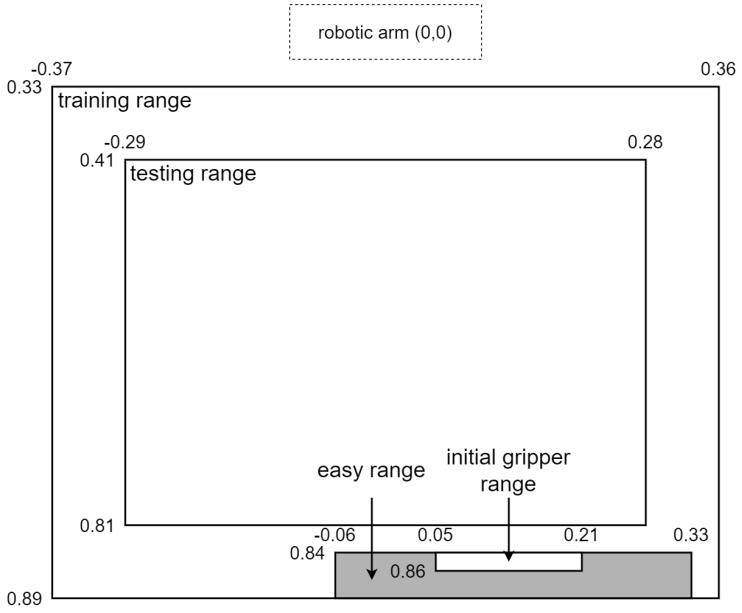

Figure 4.4: A diagram of block ranges in different phases of training. The block range of the project goal is the "testing range".

Chapter 5

Results and Discussions

This section presents and discusses the results of the three experiments ran in the project. The three experiments are conducted sequentially and the insights from previous ones are used in later ones. Eventually, the best policy is deployed on a real robotic arm and tested in a slightly different setting from the simulation environment.

5.1 Picking in the "easy range"

The first experiment is training the agent to pick up a block, while the block is in the "easy range", using DDPG and TD3. Recall that "easy range" refers to a small range that is close to the initial gripper location, which is illustrated in figure 4.4. Below is a diagram for the learning curve of the two algorithms and a table summarizing the results from testing trials.

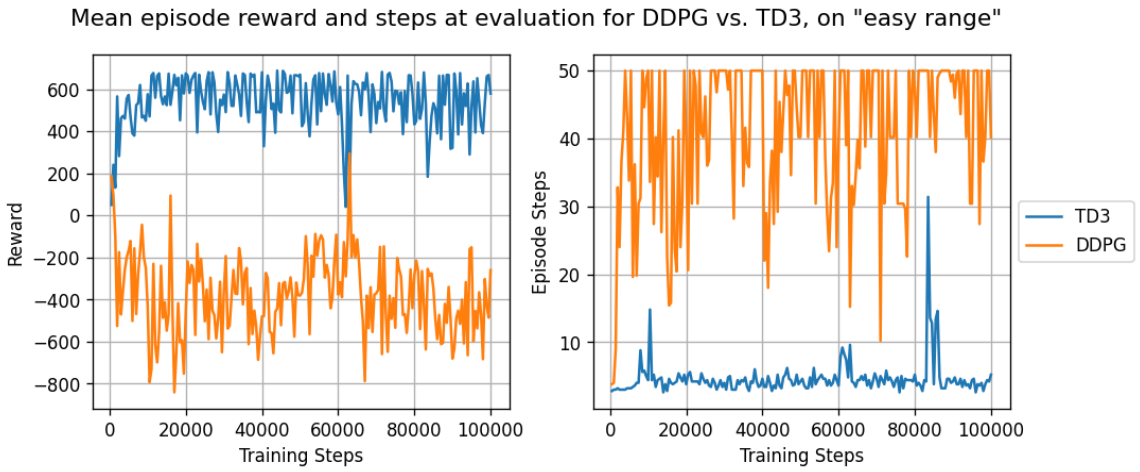

Figure 5.1: Diagrams for picking in "easy range" learning curves of DDPG and TD3.

Table 5.1: Success rates of DDPG and TD3 policies in picking task, over 3 different block ranges. Each set up is ran over 20 trials.

Algorithm	"easy range"	"testing range"	"training range"
DDPG("easy range")	0%	0%	0%
TD3("easy range")	70%	0%	5%

In the learning curve, DDPG has rewards fluctuating around -400. Regardless of the block location, the DDPG policy always moves the gripper to its left and then stops there without going outside of workspace. On the other hand, TD3 reaches episode reward 700 from time to time in the learning curve, which implies that it can succeed in most of the episodes. In the testing trials, the failed scenarios of TD3 include making "invalid moves" and landed early. However, DDPG never triggers any of the two.

5.1.1 Over-estimation bias

The reason why DDPG failed to learn a sub-optimal policy is probably due to the issue of over-estimation bias in critic, which is alleviated in TD3 by the clipped double Q-learning. In many scenarios, the over-estimation only makes the training less efficient. However, in this experiment, we can argue that DDPG converged to a poor policy, so the over-estimation bias here has a larger impact.

One potential explanation can be that the over-estimation in this environment are more likely to occur and the over-estimation itself is large, due to the moving constraints in the environment. Recall from table 4.1, there is a termination condition in this environment, which is called "invalid move". "Invalid move" includes actions that attempts to move the joints too fast or has no IK solution. Together, the 2 constraints can make the true $Q(s, \cdot)$ function at some states s full of ravines. Consequently, it is very likely for the critic to overestimate $Q(s, \cdot)$ and the loss for the over-estimation is large.

Another potential explanation is that, the actor-critic architecture of DDPG exacerbates the negative effect of over-estimation bias, leading it to converge to a poor policy. Unlike DQN, which can obtain the policy from the estimated Q-values easily by taking $\pi(a|s) = \text{argmax}_a\, Q(s, a)$, for DDPG to obtain the policy, we need both the actor and the critic network by $\pi(a|s) = Q_\theta(s, \pi_\phi(s))$, where Q_θ is the critic network and $\pi_\phi(s)$ is the actor network. Therefore, $\pi_\phi(s)$ can be seen as the estimated maximizer of $Q_\theta(s, \cdot)$. However, during the training, the critic network itself is updated frequently and is over-estimating. According to the DDPG updating rule, the actor is updated using gradient descent to the local maxima of the current target critic network.

Putting the above two explanations together, the actual training process could be described as follows: At first, the agent learned to move the gripper to the block and the learning is accelerated by the over-estimation bias and the frequent update of the actor. However, some episodes are terminated due to "invalid moves" and received a reward of -20, when the agent is trying to move the gripper to the block. Transitions from these episodes are stored in the replay buffer, but they are not immediately used for training since transitions used for update are sampled from the buffer. Due to the over-estimation bias, the actor is not learned to avoid making invalid moves at first. Therefore, more and more "invalid move" transitions are stored in the buffer and they are very likely to be sampled. Since at each step, the penalty of moving away from the block is less than the penalty from the "invalid move", the action that moves away from the block seems to be the best action because empirically it is the least likely action to make an "invalid move". Due to the frequent update of actor network, the actor learned to minimize the chance of making "invalid moves" at all cost. Therefore, the final DDPG policy is a simple policy that never makes "invalid moves".

On the other hand, TD3 uses clipped double Q-learning to reduce the over-estimation bias, so that the actor will learn to avoid making "invalid moves" earlier. Also, TD3 updates the actor network less frequently so it will be less likely to treat the action that minimizes the chance of making "invalid moves" as the best action. Instead, it is more likely to maximize the long-term rewards, while accepting the chance of making "invalid moves".

5.2 Picking in the "training range"

The second experiment includes 2 set ups. The first set up trains the agent to pick up a block, while the block is in the "training range". The second set up also trains the agent to pick up a block, except Curriculum Learning (CL) is used. In the first 30,000 steps, the block is in the "easy range" and it is in the "training range" for later steps. Both training algorithm is set to Twin Delayed Deep Deterministic Policy Gradient (TD3) as Deep Deterministic Policy Gradient (DDPG) failed in the last experiment. The "easy range" and "training range" are illustrated in figure 4.4. Below is a diagram for the learning curve and a table summarizing the testing trials. "TD3" stands for the set up without CL and "TD3+CL" stands for the set up that uses CL.

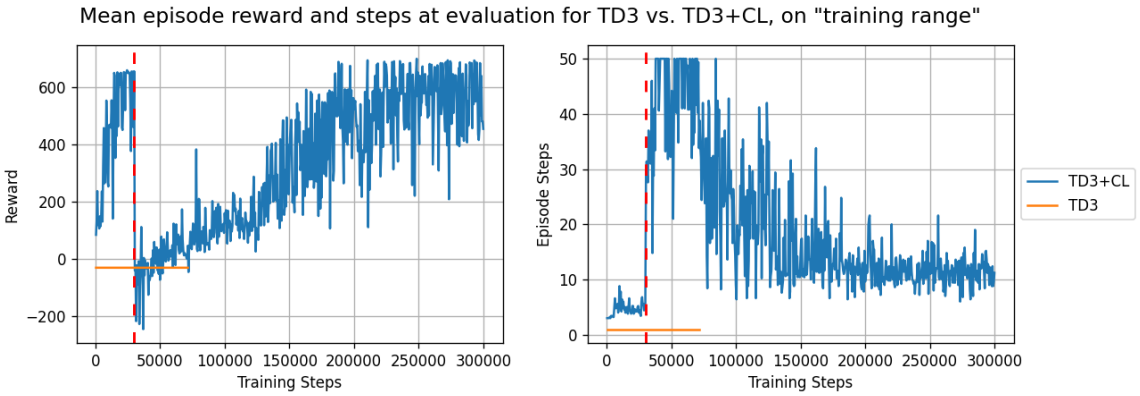

Figure 5.2: Diagrams for picking in "training range" learning curves of TD3 and TD3+CL. For TD3+CL, the two phases of CL is separated by the red dashed lines: phase 1 is picking in the "easy range" and phase 2 is picking in the "testing range".

Table 5.2: Success rates of TD3 and TD3+CL policies in picking task, over 3 different block ranges. Each set up is ran over 20 trials.

Algorithm	"easy range"	"testing range"	"training range"
TD3("training range")	0%	0%	0%
TD3+CL("training range")	80%	100%	95%

In the learning curve, the training without CL failed to learn almost anything useful. It consistently fails in the first step and obtain a constant -30 reward. The -30 reward is made up of a -20 from "invalid move" and the -10 comes from the landing reward. Since the initialization of each episode is quite time-consuming, while each episode is terminated in 1 step. In addition, from the learning curve, it is fair to argue that there is no hope for it to improve. Thus, the training without CL is stopped early to save time and resources. The training that uses CL reaches episode reward 700 quite frequently in the learning curve, which implies that it can succeed in most of the episodes. Also, the decrease in the episode steps show that the path planning is more and more efficient, as training proceeds.

5.2.1 Curriculum learning

In this experiment, Curriculum Learning (CL) definitely played a critical role. The reason why that the agent cannot learn anything on the "training range" without CL is probably because the task is too difficult, comparing to "easy range". In "easy range", the policy that reaches a high success rate can be simple: just move the gripper close to the block, without making "invalid moves". However, in "training ranges", the block is closer to the robotic arm than the initial gripper location most

of the time. To obtain a positive step reward and reach those locations, the agent needs to learn to "bend the arm". The trajectories to reach those locations are far from being linear: they are typically curves that concave down. Therefore, learning such behaviors from scratch is practically turned out to be impossible.

On the other hand, the agent can learn important knowledge from the easy tasks in CL. The most important knowledge in this task is probably the goal, which is moving the gripper to the target location. Since success in the "easy range" is easy, the agent can learn the goal in early stages of training. However, if the agent is directly trained on the "training range", successful episodes are very rare at the early stages of training. Therefore, the agent is unable to learn the goal in the early stage, while to learn the goal, a decent policy is required to collect the relevant transitions. Consequently, the agent ended up with nothing useful is learned.

5.3 Pick-and-place

The third experiment is training the agent to pick-and-place. From the previous two experiences, both Twin Delayed Deep Deterministic Policy Gradient (TD3) and Curriculum Learning (CL) are very important techniques to use. The set up for training the agent to pick-and-place goes like follows: in the 0 to $30,000$ steps, the agent is trained on picking in the "easy range". In the $30,000$ to $200,000$ steps, the agent is trained on picking in the "training range". In the $200,000$ to $400,000$ steps, the agent is trained on pick-and-place in the "training range". After that, the performance of the policy at the end of $200,000$ steps, which is not trained on pick-and-place, and the policy at the end of $400,000$ steps are compared. Below is a diagram for the learning curve and two tables summarizing the testing trials for each

evaluated policy.

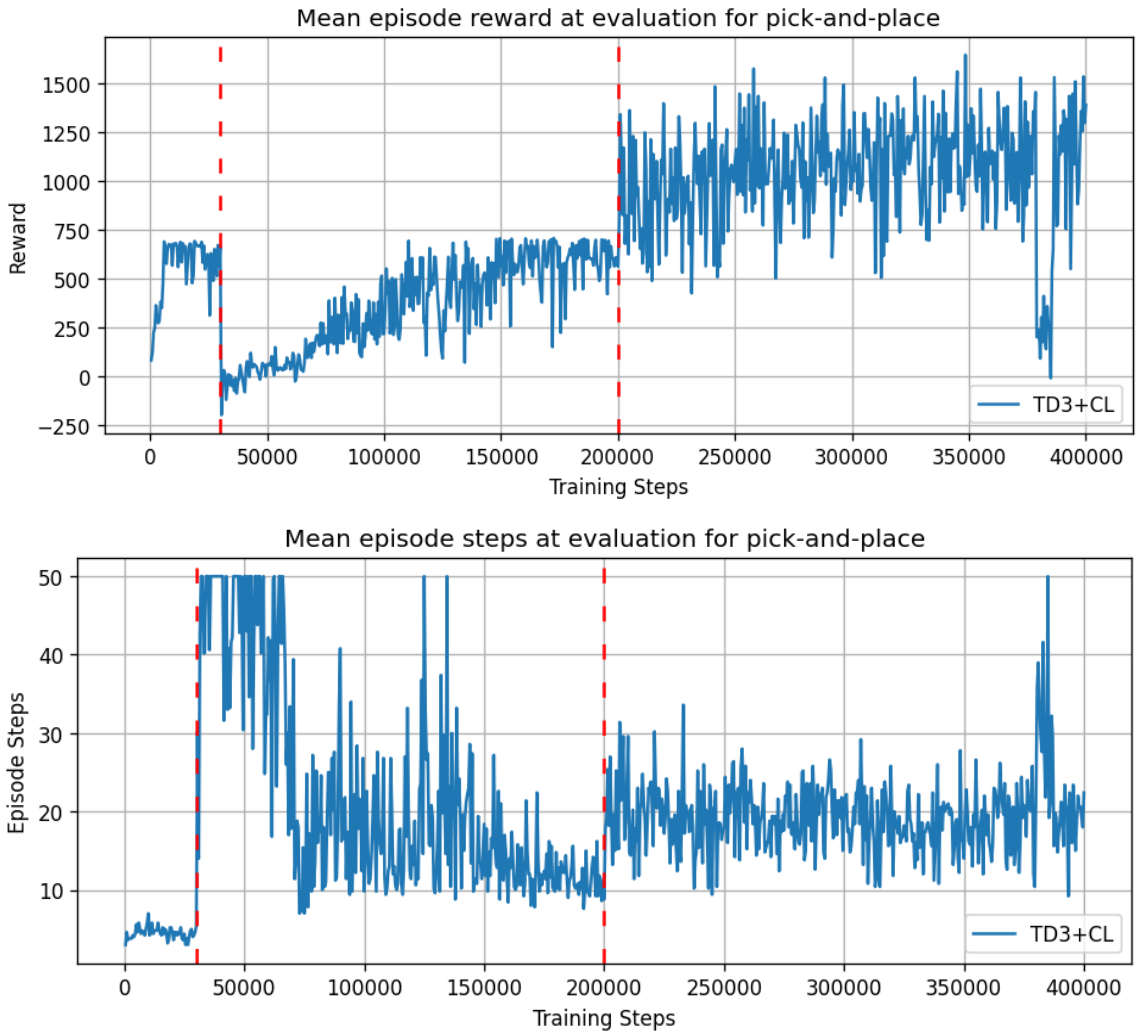

Figure 5.3: Diagrams for pick-and-place learning curves, with 3 phases, separated by the red dashed lines: phase 1 is picking in the "easy range", phase 2 is picking in the "testing range" and phase 3 is pick-and-place in the "testing range".

Table 5.3: Success rates of TD3+CL policies in pick-and-place task, over 2 different block ranges. The policy is not trained on pick-and-place. Each set up is ran over 20 trials.

Termination	"testing range"	"training range"
Successful picking and placing, placing is accurate	95%	80%
Successful picking and placing, placing is inaccurate	0%	10%
Successful picking, failed in placing	5%	10%
Failed in picking	0%	0%

Table 5.4: Success rates of TD3+CL policies in pick-and-place task, over 2 different block ranges. The policy is trained on pick-and-place. Each set up is ran over 20 trials.

Termination	"testing range"	"training range"
Successful picking and placing, placing is accurate	75%	35%
Successful picking and placing, placing is inaccurate	15%	10%
Successful picking, failed in placing	5%	50%
Failed in picking	5%	5%

5.3.1 Reward hacking

From the learning curve, we can see that a drop in the episode reward is occurred at around $380,000$ steps, with a rise in the episode steps. This is because the agent was attempting to hack the step reward. This is possible since the IK cannot move the gripper to the desired location perfectly and the step reward is using the actual gripper displacement.

Recall the step reward is proportional to the difference between the gripper location and the target, that is: $\text{step\_reward}(s_t, a_t) = C(||g_t - g_{\text{desired}}||_2 - ||g_t + a_t - g_{\text{desired}}||_2)$, where g_t is the gripper location at time t and g_{desired} is the target location. C only depends on g_{desired}, so it is fixed in each episode. Suppose the an action a is executed that moves the gripper far away to from the block. However, due to the imperfectness of IK, the gripper is actually moved further away from the block. Then, another action a' is executed that moves the gripper closer to the block than the location before executing a. Again, due to IK, the gripper is not moving as closer as a' wished. Consequently, the gripper is moved further away from the block, but the total reward from these two transitions gives a positive reward. Figure 5.4 below is a diagram illustration on the described example for reward hacking.

However, as the training proceeds, the policy will not converge to one that attempts to hack the reward all the time. Since the max time step is finite and it is not observed by the agent, the agent does not know when to stop the hacking behavior within an episode and continue reaching the goal. This phenomenon is also shown in the learning curve. Still, due to the possibility of reward hacking, the optimal policy for this task is not exactly the one we want. It is true that Twin Delayed Deep Deterministic Policy Gradient (TD3) uses deterministic policy, the optimal policy can

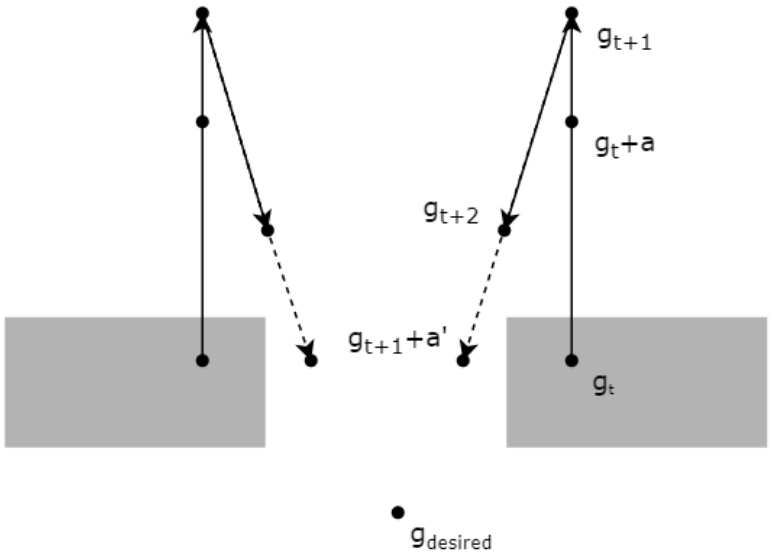

Figure 5.4: A diagram illustration for reward hacking and the policies that utilizes reward hacking. g_{t+2} has more collected reward than g_t, it is farther away to the goal than g_t. Policies that only hack reward in the grey region can still reach the goal within the max time steps.

choose some states and hack reward at those states, while in other states, the policy will just approach the block. The illustration is in figure 5.4 above. Noticed that if the policy only hacks reward at the grey area, it can still reach the goal within the max time step. Therefore the optimal policy will hack the reward in some states.

One potential way to avoid hacking is to include a negative constant term in the step reward, so that the optimal policy plans the most efficient trajectory. However, the choice of the constant term might be challenging: if it is too small, it won't be effective enough. However, a too large constant can hinder the exploration.

5.3.2 Stability in the critic

From the test trials in 5.3 and 5.4, we can see that the policy that is trained on pick-and-place is not as good as the one without training on pick-and-place, especially in the "training range". By reviewing the RL environment set up, the potential reason is due to the missing specification of the current task in the states. That is, the agent needs to know whether the current task is picking or placing. It is true that the optimal policy might not differ much in picking and placing, as the policy that is not trained on pick-and-place can do well in placing, there might be a serious effect in the critic.

Without knowing the current task can make the critic unstable. Here is an example to show this: assuming at time t, the state is s and the action a is fixed to be going straight down. At time $t+1$, the state is s' and the gripper is closer to the block. Later, at time $t+2$, the block is reached and the script for landing is launched. After running the script, the gripper is back to state s. Then the Q-values satisfy $Q(s,a) \leq Q(s',a) \leq Q(s,a)$, which is a contradiction and can cause unstable updates. On the other hand, if we include a bit to denote the task, say 0 for picking and 1 for placing, we have $Q([s,0],a) \leq Q([s',0],a) \leq Q([s,1],a) \approx Q([s,0],a) + 700$, which solves the contradiction and stabilizes the updates.

5.4 Simulation to reality

From table 5.3, the best policy has a success rate of 95% on pick-and-place in the "testing range", which is reasonable to be deployed on a real robotic arm. In reality, the setting is adjusted for practical concerns. Specifically, the gripper on the real robotic arm has a smaller finger distance than the one in the simulation environment.

As a result, the real gripper cannot pick up a cube with each side to be 4.5 cm. To solve this, a cylinder, with height 4.318 cm and diameter 1.08 cm, is used instead of the block. The height of the table is also different from the one in the simulation environment, which does not affect path planning and can be handled by modifying the landing script. Finally, in the simulation environment, the gripper is always reset to an initial pose before an episode begin. In reality, the gripper is not reset after finishing one round of pick-and-place.

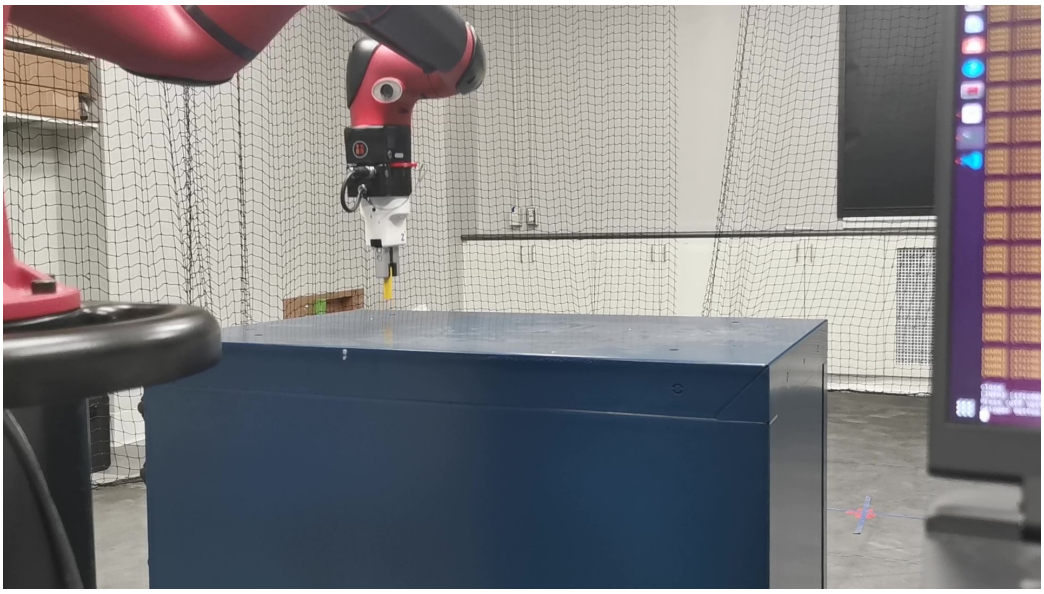

Figure 5.5: A picture of a physical Sawyer robotic arm, picking up a cylinder in Dr. Givigi's lab.

Through the trials in reality, when the gripper moving distance is above for approximately 25 cm, there is a risk of the gripper attempts to land early, resulting in an inefficient trajectory, which is not observed in simulation environments.

One potential solution is to include a penalty term in the step reward, when the gripper location has a low height. The term can be constant, or decreases as the height increases. However, this term should be tuned carefully. If it is too small,

it does not make a significant change. On the other hand, if it is too large, it can exacerbate the over-estimation bias. Also noticed that relaxing the landing early termination condition may not be able to solve this issue. That is, during training, we consider height lower than 15 cm to the target is landed early, but when we deploy it in reality, only 10 cm is considered landed early. The issue is that the policy never visit a state with height below 15 cm during training, so it may output unexpected actions.

Chapter 6

Conclusion and Future Work

This section summarizes the thesis by recapping the project's goal, summarizing the experiments conducted and their results, explaining the outcomes, and proposing the future works.

6.1 Conclusion

The project uses Deep Reinforcement Learning (DRL) to solve a robotic arm pick-and-place problem. Three experiments are conducted in a simulation environment. In the first experiment, the agent is trained to solve a picking problem in a small area. DDPG converged to a poorly-performed policy, while TD3 converged to a sub-optimal policy, indicating that vanilla algorithms can suffer from severe over-estimation issue in this environment. In the second experiment, the agent is trained to solve a picking problem in a larger areas. TD3 learned a trivial policy, however, when Curriculum Learning (CL) is used, TD3 learned a sub-optimal policy. This shows that some additional techniques, which can help the agent to learn the goal, are required to solve this problem. In the third experiment, the agent, which succeeded in the second experiment, is further trained on pick-and-place. The result shows that

the agent became worse when trained on pick-and-place. This indicates that training on a carelessly formulated state space and reward function can lead to reward hacking and instability in training, which decreases the agent's performance. For simulation to reality, the best policy can succeed in pick-and-place when the moving distance during is not too long. One way to improve the agent can be add penalty terms on the step reward when the gripper is at a low height.

6.2 Future Work

Different training strategies and DRL algorithms can be tried as an extension. Instead of training the agent first to pick and then pick-and-place, we can train the agent to learn pick-and-place directly. In addition, different algorithms, such as soft actor-critic (SAC) [4] and trusted region policy optimization (TRPO) [15], or incorporating hindsight experience replay (HER) [2] can be used. The current environment is already a goal environment, so adopting HER is simple. However, due to the long training time and the time limit, this project cannot afford further experiments in the sense of time.

Building upon this project, more challenging tasks, such as obstacle avoidance and using vision to predict the target locations can be tried. For obstacle avoidance, it is important to keep the environment fully-observed, which means there is a function from state to the actual obstacle. This can be very important as the current environment has a severe over-estimation bias problem and a partially observed state can exacerbate that. For vision-based methods it is almost trivial in simulation environment as the simulated image is far from being realistic. For vision-based methods in reality, Andrychowicz et al. [2] showed that it is important to include noises to the

provided target location in the simulation environment. The included noise can be seen as a mimic of prediction error from the vision models.

Bibliography

[1] M. Al-Gabalawy. Path planning of robotic arm based on deep reinforcement learning algorithm. *Advanced Control for Applications: Engineering and Industrial Systems*, 4(1):e79, 2022.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[3] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[5] K. Hauser. Chapter 6. inverse kinematics, 2023. URL `https://motion.cs.illinois.edu/RoboticSystems/InverseKinematics.html`. Accessed on July 1st, 2023.

[6] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[8] A. Malik, Y. Lischuk, T. Henderson, and R. Prazenica. A deep reinforcement-learning approach for inverse kinematics solution of a high degree of freedom robotic manipulator. *Robotics*, 11(2):44, 2022.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[10] S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*, pages 566–574, 2016.

[11] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1):7382–7431, 2020.

[12] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer, 1999.

[13] M. T. Regehr and A. Ayoub. An elementary proof that q-learning converges almost surely. *arXiv preprint arXiv:2108.02827*, 2021.

[14] R. Robotics. Sawyer hardware, 2022. URL `https://support.rethinkrobotics.com/support/solutions/articles/80000976455-sawyer-hardware`.

[15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[16] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[17] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.