Pick and Place Robot using Q-Learning Algorithm

Report submitted in fulfillment of the requirements
for the Exploratory Project of

Second Year B.Tech.

by

Jaideep Sharma
Siddharth Verma
Sridharan N

under the guidance of

Prof. Lakshmanan Kailasam

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
Varanasi 221005, India
May 2022

Dedicated to

Our parents and teachers

Declaration

We certify that

1. The work contained in this report is original and has been done by ourselves and under the general supervision of our supervisor.

2. The work has not been submitted for any project.

3. Whenever we have used materials (data, theoretical analysis, results) from other sources, we have given due credit to them by citing them in the text of the thesis and giving their details in the references.

4. Whenever we have quoted written materials from other sources, we have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: May 9, 2022

Jaideep Sharma, IDD
Siddharth Verma, B.Tech.
Sridharan N, B.Tech.
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Certificate

*This is to certify that the work contained in this report entitled "**Pick and Place Robot using Q-Learning Algorithm**" being submitted by **Jaideep Sharma (Roll No. 20074017)**, **Siddharth Verma (Roll No. 20075086)**, and **Sridharan N (Roll No. 20075109)** carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: May 9, 2022

Prof. Lakshmanan Kailasam
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Acknowledgments

We would like to express our sincere gratitude to our respected professor Dr. Lakshmanan Kailasam, who gave us the golden opportunity to do this wonderful exploratory project in Reinforcement Learning and provided support and guidance. The project enabled us to learn many concepts related to major algorithms involved in Reinforcement Learning and simulate a self-learning robot using PyBullet.

Place: IIT (BHU) Varanasi
Date: May 9, 2022

Jaideep Sharma, IDD
Siddharth Verma, B.Tech.
Sridharan N, B.Tech.
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Abstract

In this report, we have discussed and implemented the Q-Learning algorithm under Reinforcement Learning, which is intended to train a robotic arm to pick ‘m’ objects that are misplaced in the n^2 squares of an n*n grid and place them in their corresponding correct positions, using the least possible time while monitoring its battery level.

We have made a few assumptions involved in the task described above:

1. The initial and final positions of any two objects do not overlap. Therefore, there will be 2*m cells required in the grid for ‘m’ objects.
2. Accordingly, the inequality $2*m \leq n^2$ is followed.
3. The robot already knows the initial and final positions of each object.

Contents

List of Figures

List of Tables

List of Symbols

Symbol	Description
S_t	The state at time t
$\mathscr{S}$	The set of all feasible states at time t
A_t	The action at time t
$\mathscr{A}(S_t)$	The set of all possible actions in state S_t
R_t	The reward obtained at time t, dependent on S_{t-1} and A_{t-1}
$\Re$	The set of all possible rewards
π_t	The robot's policy at time t
$\mathbb{R}$	The set of real numbers
α	The learning rate
γ	The discount rate
G_t	N-step return, i.e., the sum of discounted rewards at time t

Chapter 1

Introduction

1.1 Overview

Industries of modern times generally use 'pick and place' robotic arms in manufacturing. The automated process of arranging objects without workforce requirements via robotic technology speeds up the production rates, improving the nation's economy.

1.2 Motivation of the Research Work

Reinforcement Learning is a rapidly evolving field of Machine Learning which uses a continuous process of rewards and punishments to train complex robotic systems to take suitable actions to achieve one or more goals in an unprecedented environment. This project on “Pick and Place Robot using Q-learning Algorithm” is an epitome of a Reinforcement Learning task having many useful, practical applications in the modern tech-savvy world.

1.3 Organisation of the Report

In this report, we first highlight the agent-environment interface of the pick and place robotic arm and verify if the state of this interface satisfies the Markov property, and if it does, we formulate a Markov Decision Process (MDP) consisting of the state of the interface, action, reward function, and transition function of the robotic arm in the environment defined. We then train the robotic arm using the Q-learning algorithm and analyze the graphical outcomes making the required computations. Finally, we simulate the whole process via the PyBullet module in Python and accentuate the mechanisms involved in picking and placing.

Chapter 2

The Agent-Environment Interface

2.1 The Agent and the Environment

A reinforcement learning problem involves learning from interaction to achieve a goal. The decision-maker and learner is called the agent, and the components that the agent interacts with, encompassing everything outside the agent, constitute the environment. Anything that cannot be altered arbitrarily by the agent also forms a part of the environment. **[1]**

Considering the above definitions, we decide on the agent and environment defined in our project as follows -

- The agent in our project is the pick-and-place robotic arm which must organize the misplaced objects in the least possible time and maintain a safe battery level during the pick-and-place process for each object, such that a picked-up object does not fall during the process. On moving an object from a square to its neighboring square, the robot's battery level is reduced by a discharge rate decided before the task.
- The environment comprises the n*n grid, the objects to be arranged, and the robot's battery. The robot interacts with the objects and the grid in which they are present to organize them. The robot cannot randomly change the size of the grid, and its battery level cannot increase unless the robot decides to recharge.

The robot formulates a policy to accomplish the given task, as part of which it selects actions and the environment responds to them, furnishing new situations to the robot and ensues rewards based on the quality of the actions chosen. The rewards are evaluated numerical values that the robot endeavors to enhance over time. **[1]**

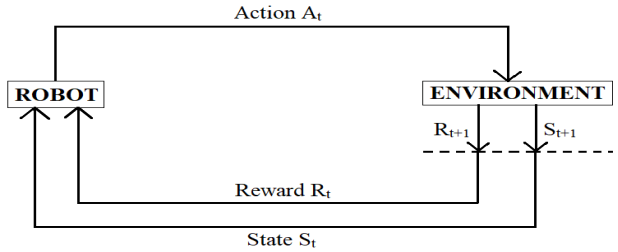

Figure 2.1 *The interaction between the robotic arm and the environment*

We note the following points from figure 2.1 -

- The robot and the environment interact at discrete time steps,

i.e., at t = 0, 1, 2, 3, …. The robot receives a configuration of the environment's state $S_t \in \mathcal{S}$ at a time t, where $\mathcal{S}$ represents the set of all feasible environment states.

- An action A_t is selected by the robot based on the states, such that $A_t \in \mathcal{A}(S_t)$ where $\mathcal{A}(S_t)$ is the set of all actions available in the state S_t. As the robot receives a reward $R_{t+1} \in \Re$ at time t+1, it enters into a new state S_{t+1}. At time t+1, the robot enters a new state S_{t+1}, receiving a reward $R_{t+1} \in \Re$, where $\Re$ is the set of all possible rewards.
- The robot implements its policy at every step, which is a mapping from the perceived states of the environment to the probabilities of choosing an action in $\mathcal{A}(S_t)$. The policy is denoted by π_t, and $\pi_t(a|s)$ indicates the probability that $A_t = a$ when $S_t = s$.

2.2 State of the Interface

At any given time t, the state S_t of the agent-environment interface is depicted by the grid configuration consisting of 'm-1' objects located at random positions, with one of the 'm' objects being picked up and carried by the robotic arm, and the robot's battery level.

At time t = 0, all objects lie in the incorrect positions, and on completion of the task, they lie in their correct positions in the 'm' squares of the grid.

The initial and final states of the grid for a task performed, as an example, are depicted in *figure 2.2*. The number 0 represents an empty square in the grid, and the objects are represented by the numbers 1 to 10.

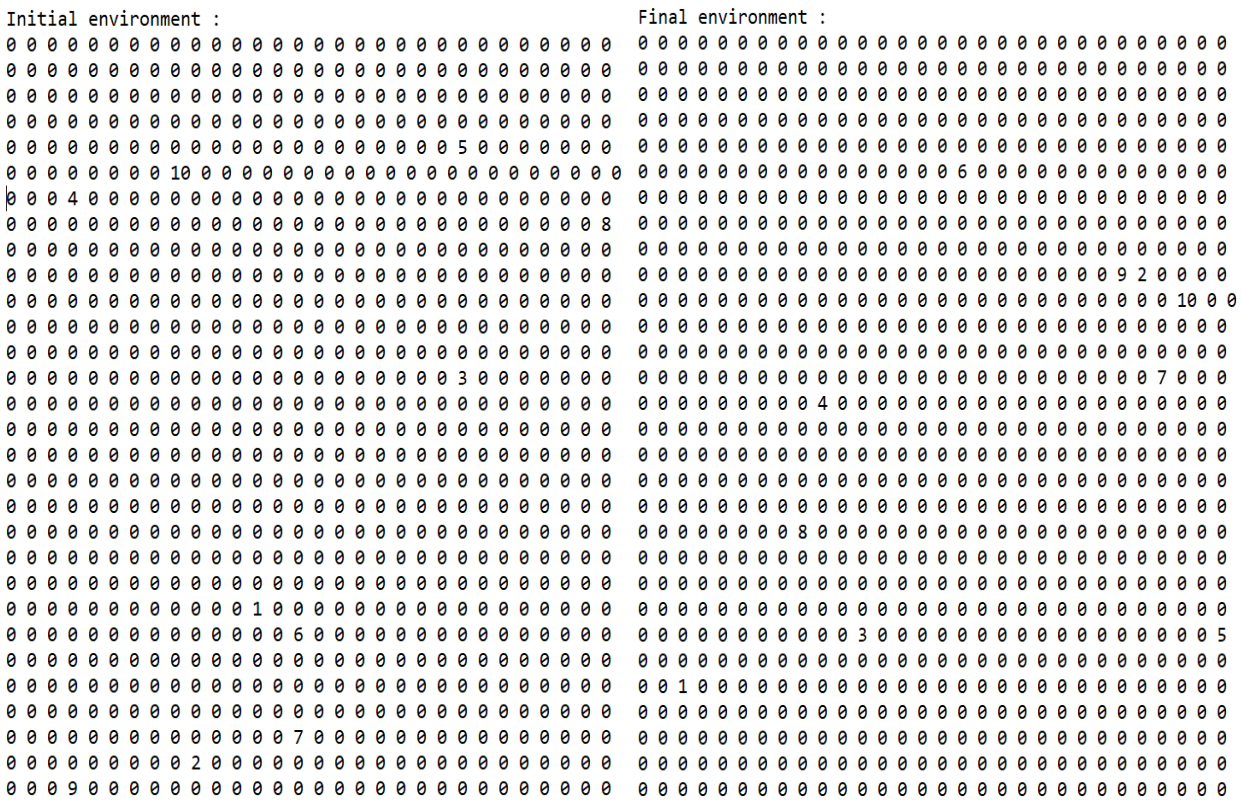

Figure 2.2 *The initial and final states of the grid in a sample task.*

2.3 Summary

In this chapter, we have first decided on the agent and the environment and then described their interaction. Following this, we have theoretically demonstrated the state of the agent-environment interface. We will illustrate the Markov property and formulate the Markov Decision Process for the task of pick-and-place robot in the next chapter.

Chapter 3

Markov Property and Markov Decision Process

3.1 Markov Property

Markov Property is a memory-less property of a stochastic process. The state signal that we have defined compactly summarizes all the past events and successfully retains all the relevant information but does not give any other helpful information to the robot for future decision-making. Therefore, the state we defined previously is said to satisfy the Markov property. **[1]**

In the most general case, the environment's response at a time t+1 to the action taken at time t depends on everything that has happened earlier. Hence, we can define the environment's dynamics only by specifying the complete probability distribution $p_{general}$:

$p_{general}(s', r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s' |S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$ $\forall$ reward $r \in \mathbb{R}$, next state s', and all possible values of the past events: $S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t$. **[1]**

If the state signal satisfies the Markov property, then the environment's response at a time t+1 is dependent only on the state and action representations at time t. Thus, we can define the environment's dynamics by specifying only p_{markov}:

$p_{markov}(s', r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s' |S_t, A_t\}$ $\forall$ reward $r \in \mathbb{R}$, next state s', $A_t \in \mathcal{A}(S_t)$, $R_{t+1} \in \Re$. **[1]**

For a Markov state, it follows that

$$p_{general} = p_{markov}$$

In this case, the environment and task also have the Markov property. **[1]**

3.2 Markov Decision Process (MDP)

Any task in reinforcement learning fulfilling the Markov property is called a Markov Decision Process (MDP), which is a tuple {S, A, R, T}, where S, A, R, and T represent the state space, action space of the agent, the reward function, and the transition function, respectively **[1][3]**. We have already defined the state of the robot and its environment. In the coming sections, we will determine the executable actions of the robot according to a policy, the reward, and transition functions.

3.2.1 Diagrammatic Representation of the State

Consider the figure representing a part of the n*n grid below.

0	**1**	**2**		
3	**OBJECT X**	**4**		
5	**6**	**7**		
				FINAL

Figure 3.1 *A part of the n*n grid.*

Let 'X' denote the number from 1 to 'm' among objects, and 'FINAL' denote the correct position of 'X' in the grid. The next positions where the robotic arm takes the object are labeled from 0 to 7 and stored in the position[] array. The Euclidean distances between each labeled cell starting from 0 and the 'FINAL' cell are computed and stored in the distance[] array. The Euclidean distance between two points with Cartesian coordinates (a, b) and (c, d) is equal to $\sqrt{(a-c)^2 + (b-d)^2}$. Therefore, the distance[] and position[] arrays for object 'X' in the current state are-

$distance[] = \{5.000,\ 4.242,\ 3.605,\ 4.472,\ 2.828,\ 4.123,\ 3.162,\ 2.236\}$

$position[] = \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7\}$

3.2.2 Action and Policy

We obtained the distance[] and position[] arrays for object 'X' in the state shown in the previous section. The distance array of the state at a time t, corresponding to the different positions of object 'X' in the next state at a time t+1, is sorted in ascending order, with the positions swapped accordingly. So, the modified arrays are-

$$distance[] = \{2.236,\ 2.828,\ 3.162,\ 3.605,\ 4.123,\ 4.242,\ 4.472,\ 5.000\}$$

$$position[] = \{7,\ 4,\ 6,\ 2,\ 5,\ 1,\ 3,\ 0\}$$

Let L be the length of the distance[] array. L denotes the number of possible subsequent states for an object. If the object 'X' is present at the corner of the grid, then L = 3; if it is present at an edge of the grid excluding the corners, then L = 5; if it is not present at any of the edges, then L = 8. The **action** performed by the robotic arm is to select a position so that the reward R_{t+1} obtained at time t+1 is maximized. The better the quality of the action, the greater the reward will be. The method implemented for choosing actions in subsequent states is the robot's **policy**.

Let us define three arrays storing the index of the selected position in the position[] array as-

$$index8[] = \{8\},\ used\ if\ L = 8$$

$$index5[] = \{5\},\ used\ if\ L = 5$$

$$index3[] = \{3\},\ used\ if\ L = 3$$

Initially, since the robot is unaware of the betterness of one action over the other, it selects and appends a random index from 0 to L - 1 until there are at least 'x' unique elements present in the corresponding arrays of the selected indices - *index8[]*, *index5[]*, and *index3[]* initialized with 8, 5 and 3 respectively. The robot stores the selected index in the suitable array according to the value of L. The learning rate 'α' (part of the Q-learning algorithm) of the robot influences the value of 'x'. With increasing values of 'α', the value of 'x' decreases, so fewer iterations are required for learning to choose better indices to obtain a greater reward. We have defined the value of 'x' as a function of 'α' as -

$$x = 1/(10 * \alpha) + 1,\ if\ L = 8$$

$$x = 1/(20 * \alpha) + 1,\ if\ L = 5$$

$$x = 1/(30 * \alpha) + 1,\ if\ L = 3$$

Once there are at least ‘x’ unique elements present in the array of selected indices, the robot has learned which index leads to a better state. The arrays for storing indices contain the previously chosen indices at discrete time steps from 0 to t-1. Let ‘min’ denote the minimum element in the array corresponding to the value of L in state S_t. The robot then selects a random index from 0 to min - 1 in a state of ‘L’ following possible states.

Initially, the robot is unaware of how to monitor its battery levels to pick and place an object in its position safely. Let ‘d’ be the discharge rate of the battery and ‘C’ be the chosen index of the modified position[] array for moving an object ‘X’ to the next position. After each action, i.e., moving a picked-up object by one cell, the battery level reduces by ‘d’. The robot needs a negligible battery to reach an object to pick it up. The battery level required to safely place an object from the current position to its final position is ⌈distance[C]⌉ + d, where ⌈.⌉ denotes the ceil function and distance[] array is sorted. We use the ceil function because the number of steps required to take the object from the next position to its final position will be at least ⌈distance[C]⌉, and at each step, the battery goes down by ‘d’. The discharge rate ‘d’ is added to the required battery level because the robot needs ‘d’ amount of battery to take the object from the current position to the next position.

If the battery levels are lower than required for the safe positioning of the object, we give the robot warnings. If the warnings given are lesser than ‘y’, the reward, set to zero on entering each state, decreases by 1. Likewise, if the battery gets fully discharged to 0 percent, the robot drops a picked object, damaging it, and therefore the reward decreases by 4 with another warning being given. The robot must be manually recharged to resume the task in the latter case. Here, the value of ‘y’ depends on the learning rate ‘α’ as -

$$y = 1/(10 * \alpha)$$

If the battery is lower than required for safely placing an object and the number of warnings is at least ‘y’, then the robot recharges itself up to the necessary level instead of complete recharging, thereby saving the time by resuming the task as fast as possible, and the reward increases by 1. Therefore, we observe that the higher the learning rate, the lower the number of warnings required by the robot to maintain sufficient battery levels.

3.2.3 Reward Function

The aim of the robot's described policy by which it performs actions under a given state is to maximize the reward obtained from that state. The reward obtained by an action performed in a state S_t depends on the index of a position selected from the position[] array and battery level. We discussed in the previous section the influence of battery levels on reward, which affects the robot's policy. Let us find how the index chosen from the position[] array alters the reward.

Let 'R_B' be the reward obtained due to battery level 'B' in state S_t. The reward is calculated by the function-

$$reward = (L - C)/L + R_B, \; where \; R_B \in \{-5, -1, 0, 1\}$$

Therefore, the reward will be greater for lesser values of the chosen index and more suitable battery levels.

At time t, consider the state of object 'X' defined in section 3.1.1. If, for example, C = 2, d = 3, and B = 6, then the object will be taken to position labeled '6', because position[C] = 6 in the modified array. Also, L = 8, distance[C] = 3.162 and the required battery level to place the object to its final position is ⌈distance[C]⌉ + d = 4 + 3, i.e., 7. Thus, B < 7, and if the warnings given for lower battery are less than 'y', then R_B = -1. So, we calculate the total reward as -

$$reward = (8 - 2)/8 + (-1)$$

$$\Rightarrow reward = 3/4 - 1 \Rightarrow reward = -0.25$$

In another case, the warnings could have reached the value 'y'; hence the robot will recharge by one level, making B = 7 and R_B = 1. So, we calculate the total reward as -

$reward = (8 - 2)/8 + 1$

$\Rightarrow reward = 3/4 + 1 \Rightarrow reward = 1.75$

3.2.4 Transition Function

Transition is the movement of a picked-up object from one position in state S_t to the next position in state S_{t+1}. The function responsible for the environment's state change due to the agent's action is the transition function. Figure 3.2 can describe this transition from one state into another.

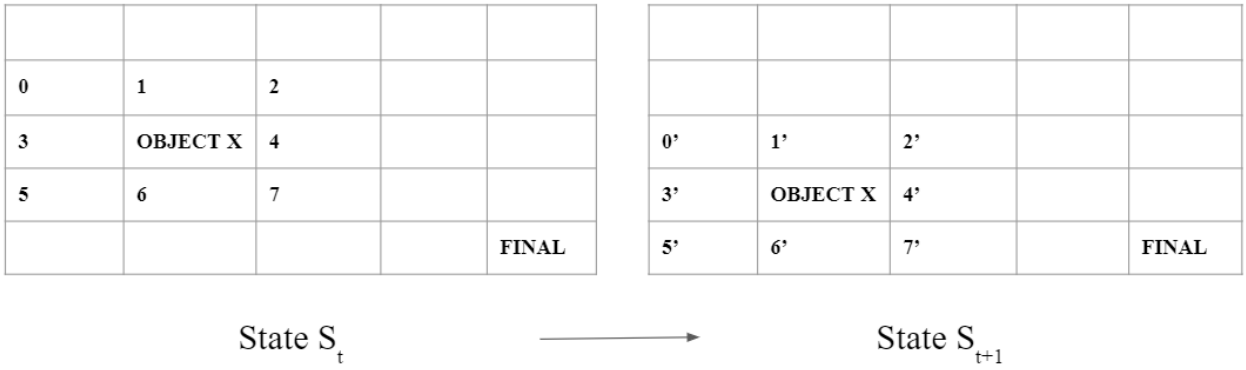

Figure 3.2 *The state transition from S_t to S_{t+1}.*

The labels of the next possible positions in state S_{t+1} are 0', 1', 2', …, 7'.

3.3 Summary

In this chapter, we first discussed the Markov property, and on verification, we found that the state of the agent-environment interface satisfies this property. So, we declared that the reinforcement learning task of the pick-and-place robot has the Markov property and hence formulated the Markov Decision Process (MDP) for the task. We then threw light on the components of the MDP of our task - state, action, the reward, and transition functions. We will elucidate the Q-learning algorithm in the next chapter.

Chapter 4

The Q-Learning Algorithm

4.1 Definition

The Q-learning algorithm is an exemplary reinforcement learning algorithm that aims to determine the best action to take, given the current state of the problem. In other words, the Q-learning algorithm intends to learn a policy that maximizes the total reward received **[1]**. The sequence of steps followed in the Q-learning algorithm includes

- initializing the Q-table,
- choosing an action,
- performing the action,
- measuring the reward,
- updating the Q-table, and
- selecting the next action. **[1]**

Eventually, after sufficient exploring, the agent learns the optimal q-values.

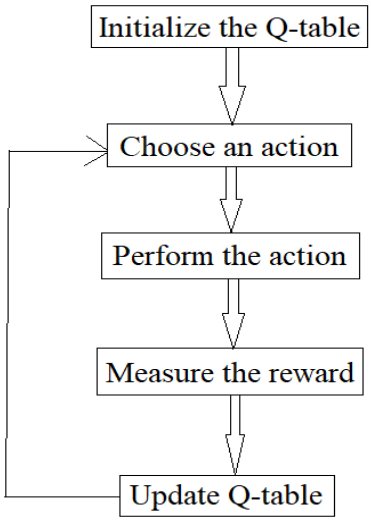

Figure 4.1 *The sequence of steps in the Q-learning algorithm.*

4.2 Computation of Q-Values

The Q-function involves the Bellman equation and takes in two inputs - the state, denoted by 's', and the action, indicated by 'a'. The Q-value of the state-action pair denoted by Q(s, a) is calculated by the formula -

$$Q(s,a) = (1 - \alpha) * Q(s,a) + \alpha * [R(s, a) + \gamma * maxQ(s', a')]$$

where $0 \leq \alpha \leq 1$, and $0 \leq \gamma \leq 1$. **[1][3]**

Here α, R(s, a), Q(s, a), Q(s', a'), and γ refer to the learning rate, immediate reward, current Q-value, maximum Q-value in the next state, and the discount-rate parameter, respectively. We initialize the Q-values for all objects to zero and observe from the equation that the maximum Q-value in the next state is given as-

$$max\ Q(s', a') = [Q(s,a) - R(s,a)] / \gamma$$

Hence, to maximize max Q(s', a'), R(s, a) must be minimized, which will occur on the maximum possible value of index 'C', according to the reward function. The worst possible reward due to the battery level will occur at the poorest index chosen.

If there are at least 'x' unique elements in the corresponding array of selected indices according to the value of L, the maximum possible value of C is given as-

$$C = min - 1$$

We redefine 'min' as the minimum element in the array of selected indices corresponding to the value of L in state S_t.

If there are less than 'x' unique elements in the corresponding array of selected indices according to the value of L, the maximum possible value of C is given as-

$$C = L - 1$$

Chapter 5

The Experiment

An experiment of the pick-and-place task of the robotic arm was performed keeping m=10, n=30, d=3, α = 0.03, and γ = 0.5. Thus, the number of objects in the 30*30 grid is 10, the battery's discharge rate is 3%, the learning rate is 0.03, and the discount-rate parameter is 0.5.
The initial and final states of the grid were observed as -

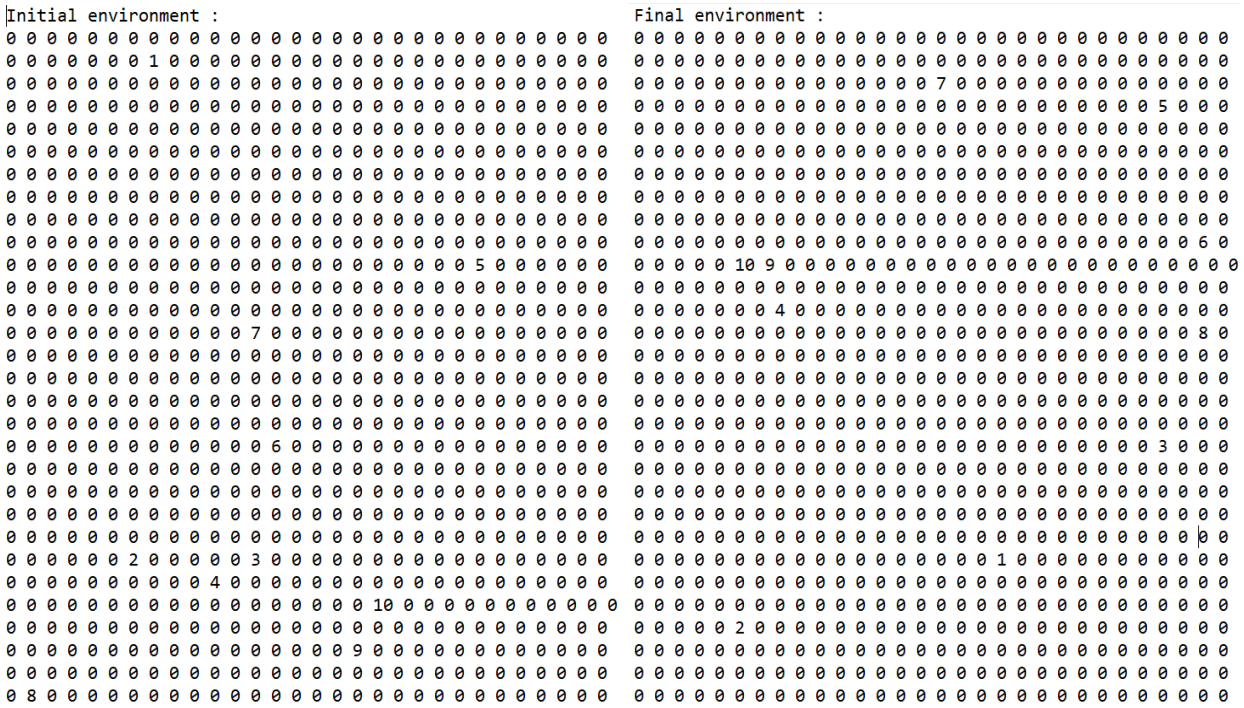

Figure 5.1 *The initial and final states of the grid in the experiment.*

5.1 The Graphical Outcomes

The following graphs were recorded for placing the ten objects in their respective places in successive episodes.

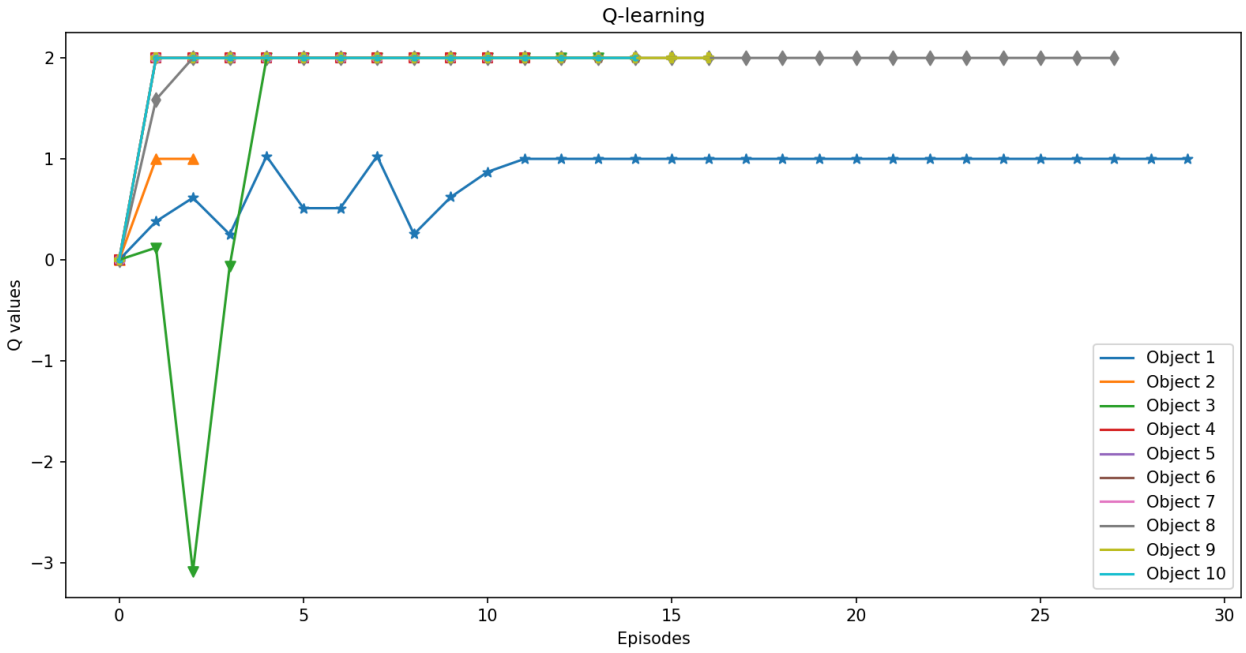

Figure 5.2 *The Q-values computed using the Q-learning algorithm.*

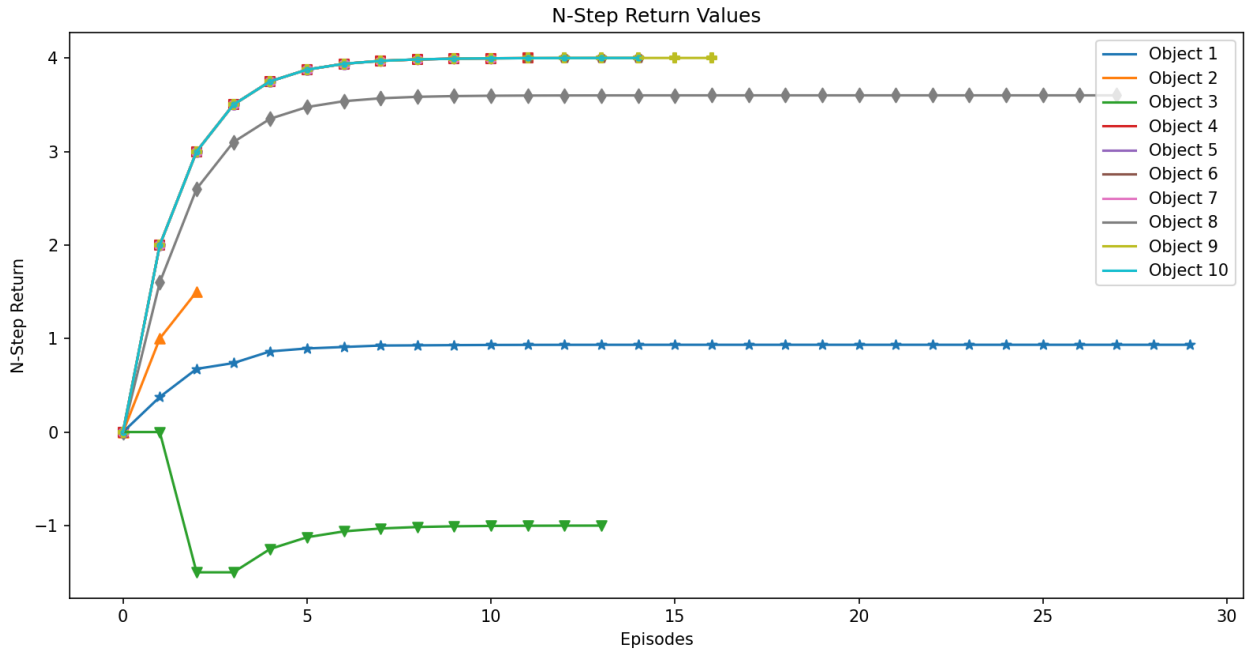

Figure 5.3 *The N-step return values.*

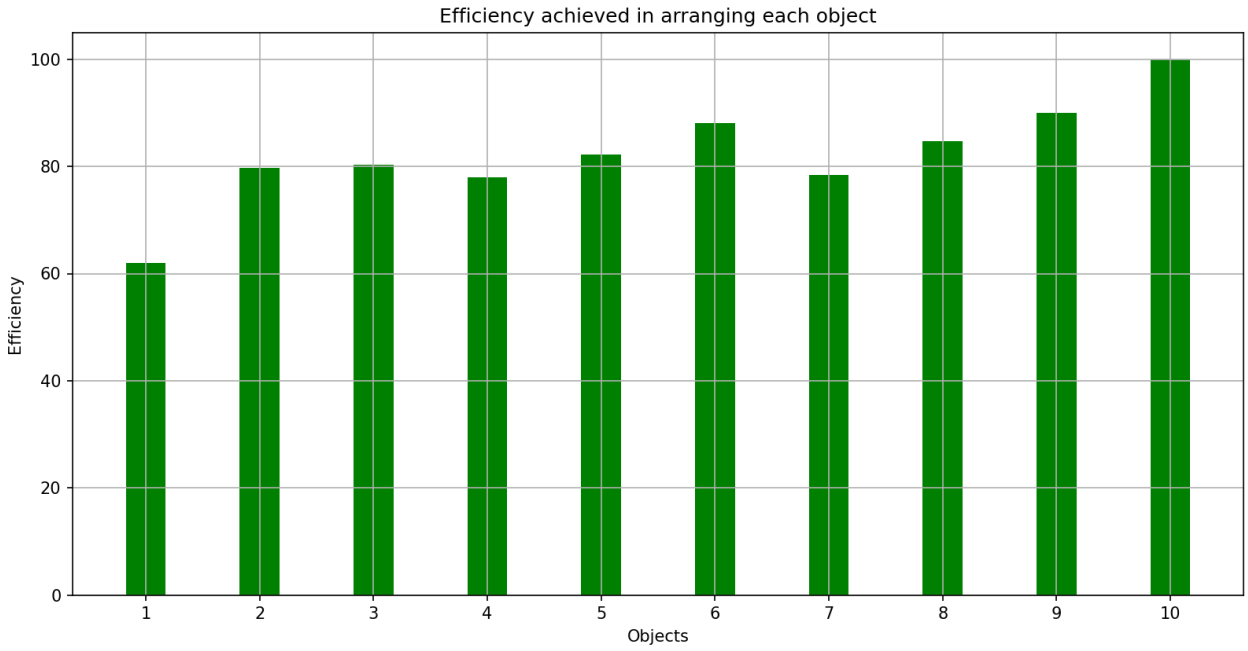

Figure 5.4 *The efficiency in percentage for placing each object.*

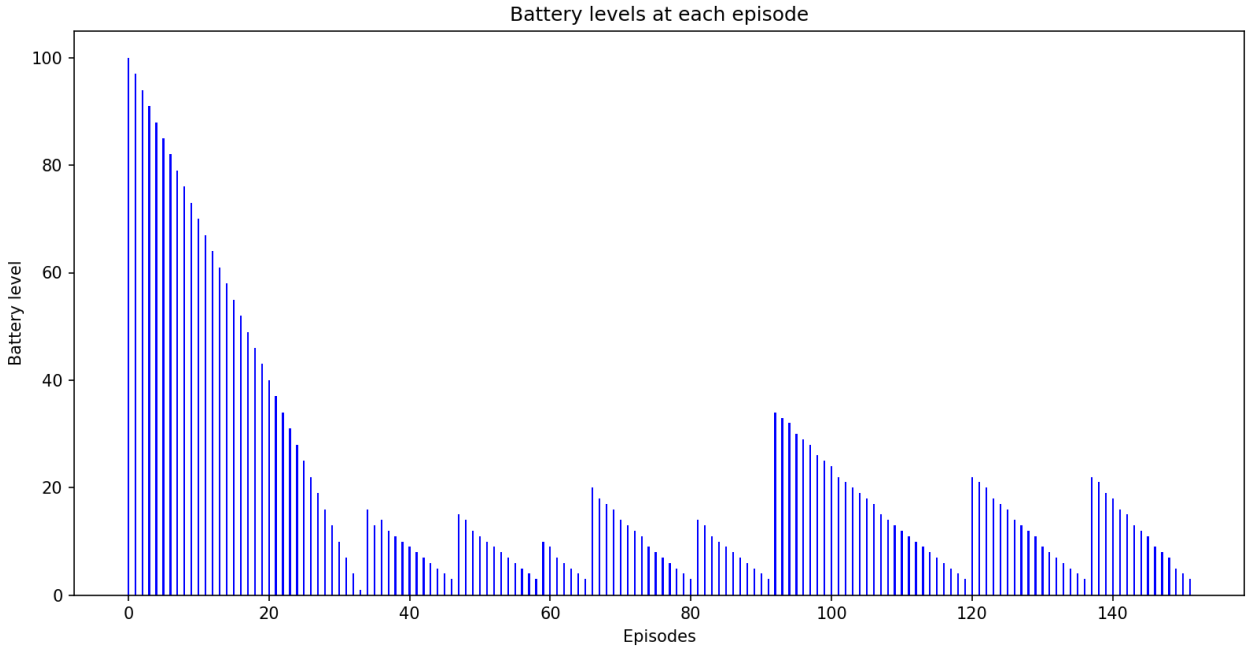

Figure 5.5 *The battery levels of the robot during the whole task.*

5.2 Definitions and Computations

- The N-step return G_t is the sum of the discounted rewards at time step t, calculated as-

$$G_t = \sum_{k=0}^{k=N-1} \gamma^k R_{t+k}. \textbf{ [3]}$$

- The percentage efficiency is the distance between the initial and final positions of an object divided by the total number of steps taken by the robot to arrange it. Thus, the greater the distance and the lesser the steps taken, the higher the efficiency. We accord a percentile of 100 to the object arranged with the highest efficiency, with relative evaluation for the other arrangements.
- Initially, we set the battery level at 100%. The battery decreases by the discharge rate d=3 for each episode of moving an object by one cell.

5.3 Inferences from the Graphs

We arrive at the following inferences from the graphs in the previous section-

- The Q-values for the successive episodes are initially non-uniform while arranging the first three objects and eventually get maximized, representing a similar curve for the remaining objects.
- The N-step returns for consecutive episodes are lesser for the first three objects - negative for object 3, but they become maximum for the latter objects. This scenario resembles that of the Q-values' graph.
- The percentage efficiency is the least for arranging the object 1 (~61.96%), with a substantial increase observed for the subsequent objects. We recorded the average efficiency of placing the objects as 82.31%.
- The robot learns to recharge itself several times by the warnings given during the task to place the objects safely. In episode 33, its battery level is the lowest during the task. After several warnings, it recharges itself before reaching such levels.

These observations suggest that the robot initially learns to perform optimal actions and ultimately determines the policy that produces the highest reward.

5.4 Comparison of Graphical Outcomes with Different Learning Rates

In the previous section, we kept the learning rate α at 0.03. Now, let us observe the behavior of our pick-and-place robotic arm with a learning rate $\alpha' = \frac{1}{10} * \alpha$, i.e., $\alpha' = 0.003$.

The initial and final states of the grid were observed as-

```
Initial environment :
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 9 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
Final environment :
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0
0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5.6 *The initial and final states of the grid.*

We recorded the following graphs with the lesser learning rate α'.

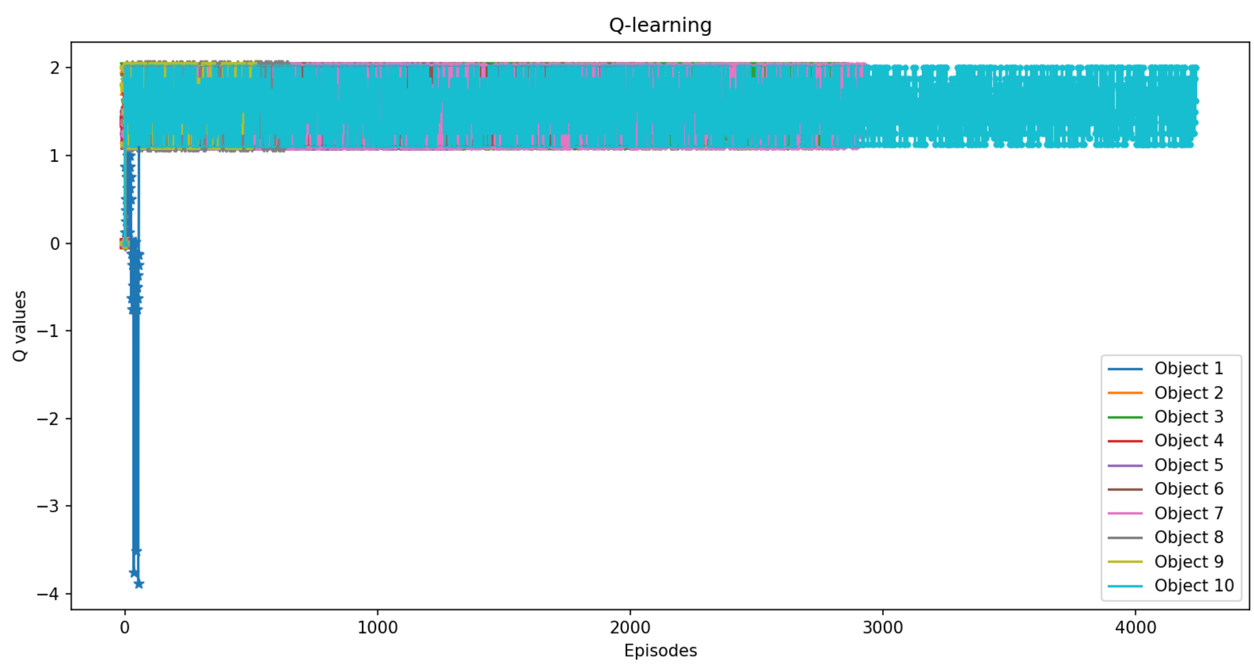

Figure 5.7 *The Q-values.*

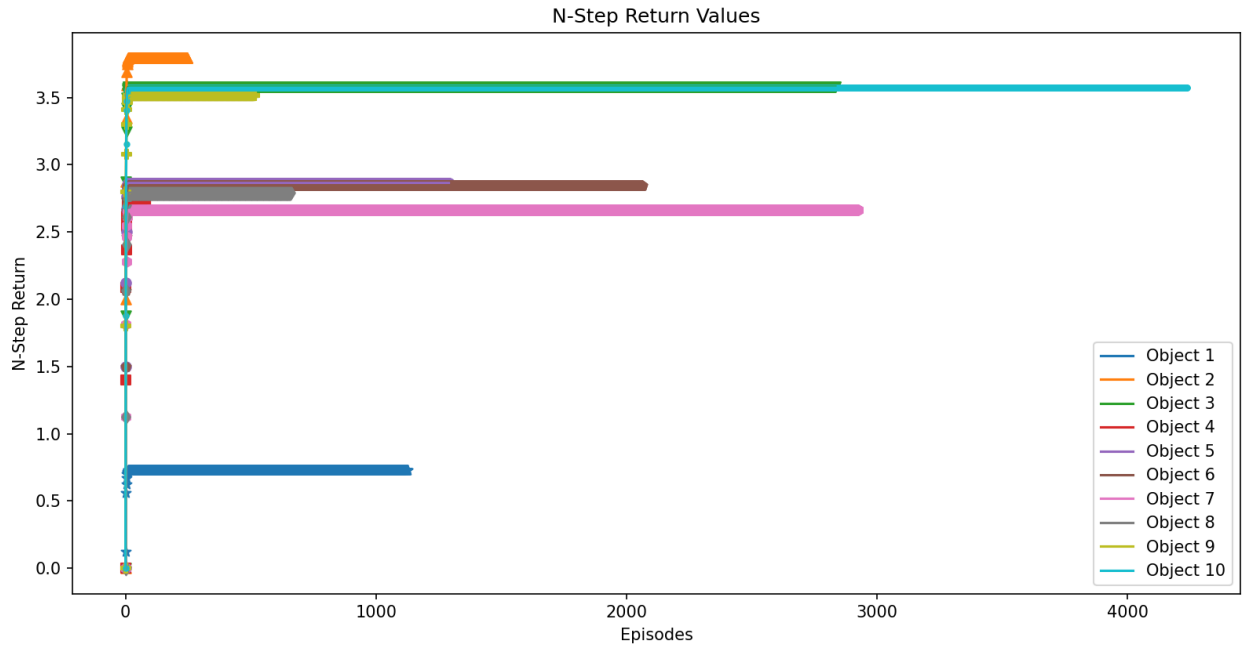

Figure 5.8 *The N-step return values.*

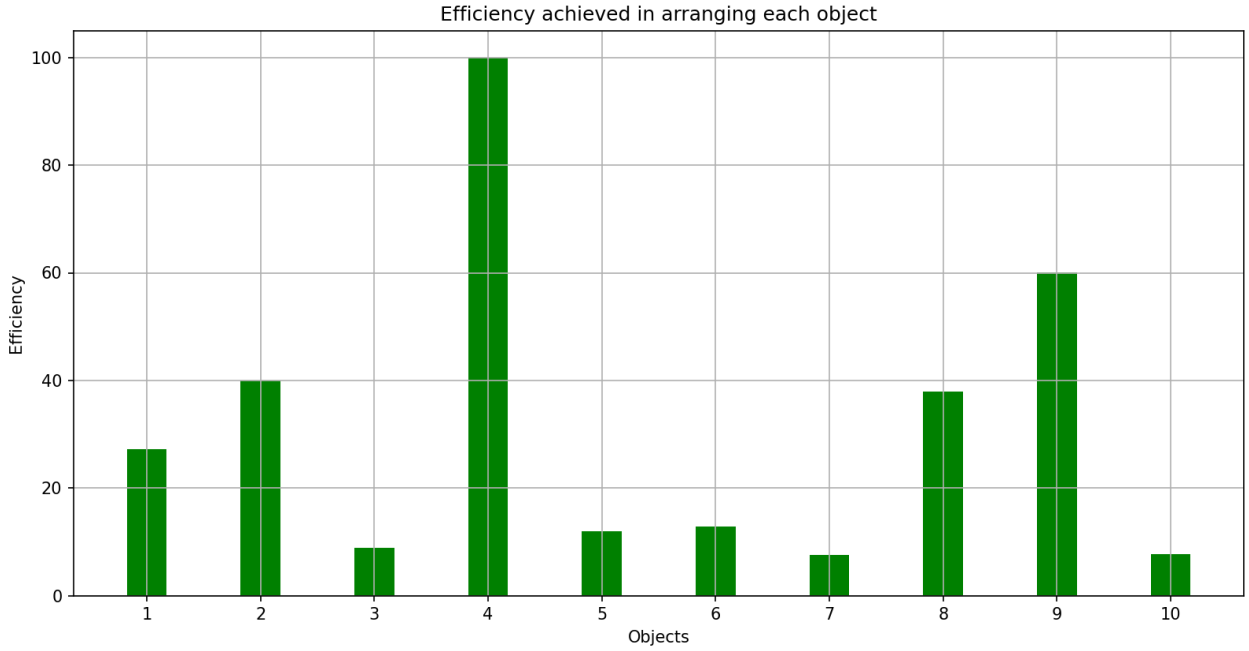

Figure 5.9 *The efficiency in percentage for placing each object.*

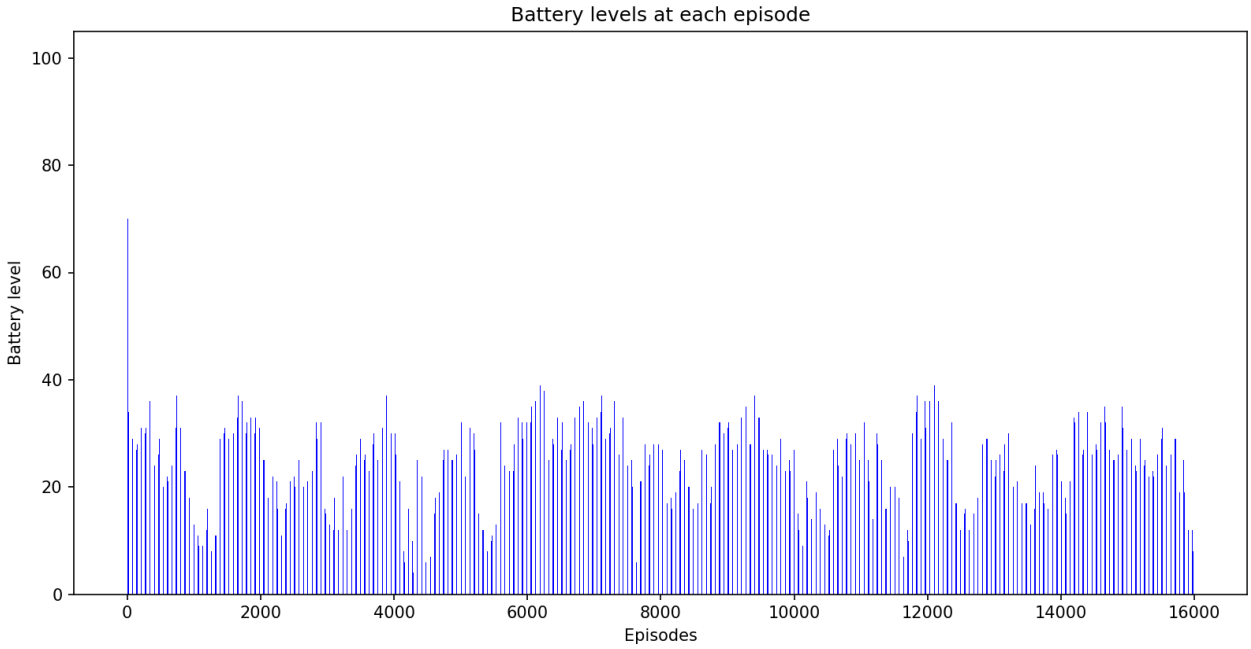

Figure 5.10 *The battery levels of the robot during the whole task.*

Let the 'first task' denote the experiment done with the learning rate α=0.03, and the 'second task' denote the experiment done with the learning rate α'=0.003.

Although each object's initial and final positions given to the robot in both tasks are quite different, the overall task to be performed does not differ significantly. But we notice that the differences observed between the performance of the robot with higher and lower learning rates are wide-ranging, such as-

- The number of episodes needed to complete the second task is relatively high compared to the first task - there are around 16000 episodes needed for arranging all objects in the second task, compared to only about 150 episodes in the first task.
- The Q-values achieved in the second task are highly irregular for all the objects, while in the first task, they were irregular for the first three objects but became uniform and maximal for the rest of the objects. Also, the Q-value for the first object is quite negative, reaching up to -4. These observations imply that the robot has not learned an optimal policy in the second task even after many episodes because of the lower learning rate.
- The N-step returns obtained show a similar pattern to the Q-values. They remain entirely random for all objects and do not maximize for arranging most of the objects, unlike observed in the first task.

- The percentage efficiency achieved for arranging each object is also completely random, showing an increasing and decreasing pattern. The efficiency for object 4, which is 100% relative to that of other objects, is also not attained following the highest reward-yielding actions. We find the average efficiency in the second task to be quite abysmal, ~31.44%, compared to ~82.31% in the first task. These observations again imply that the robot does not learn which action is better than another throughout the second task to secure a greater reward.
- The robot's battery levels fluctuate significantly during the second task than in the first task. When they fall below the required level, the robot receives warnings, and its reward decreases, but the minimum number of the warnings needed to make the robot learn to avoid lower battery levels is 34 (since y=(1/10*α')), which is 8.5 times greater compared to only four such warnings required in the first task. So, the robot has to be manually recharged quite often to complete the remaining task.

Some other experiments were also performed with different learning rates, keeping the other parameters the same, and the approximate number of episodes required to organize all objects and the average efficiency are noted in the following table-

Learning rate α	Number of episodes to finish the task	Average efficiency (in percentage)
0.9	105	83.87
0.5	119	81.81
0.1	139	81.84
0.05	159	85.84
0.01	14932	23.13
0.005	25749	23.64
0.001	28229	21.79
0.0001	37200	16.17

Table 5.1 *The number of episodes and average efficiency for tasks performed with different learning rates.*

We conclude from the above observations that the learning rate significantly affects the robot's performance concerning both time and efficiency. The higher the learning rate, the fewer are the episodes required to complete the task, and the better is the average efficiency.

5.5 Expected Value of Efficiency

In the previous sections, we executed the experiments of a single task of arranging ten objects in their respective places and calculated the average efficiency for each task. The average efficiency for a particular learning rate varies slightly on executing the same task with different initial and final configurations of the grid. Therefore, we compute the average efficiency more accurately for a task with a defined learning rate using the Q-learning algorithm.

The Law of Large Numbers states that the average results obtained from extensive tests converge to the expected value and become nearer to the expected value as more experimentations are performed. It clarifies the consequence of conducting the same experiment many times **[4]**. So, we deduce that by completing the task of our experiment a large number of times, we can get the expected value of efficiency.

We define the values required in the experiment as-

m=10, n=30, d=3, $\alpha = 0.03$ and $\gamma = 0.5$.

On executing the task 100,000 times, the average efficiency converges to its expected value at nearly 81.10%. The expected efficiency value for various learning rates, with other parameters the same as above, is recorded after executing the tasks 100000 times in the table below.

Learning rate α	Expected efficiency (in percentage)
1	82.93
0.5	82.91
0.1	82.90
0.05	82.46
0.04	81.66
0.02	80.19

Table 5.2 *The expected values of efficiency for different learning rates.*

We note from the table that the expected efficiency is directly proportional to the learning rate. The maximum achievable expected efficiency occurs at the learning rates greater than or equal to 0.1, i.e., nearly 83%.

5.6 Summary

In this chapter, we first experimented with the pick-and-place robot and assessed the outcomes with the help of several graphs, incorporating the Q-values, N-step returns, percentage efficiency, and battery levels.

We then formally defined these terms and the method of calculating them. Following this, we deduced the inferences obtained from the graphs regarding the robot's policy of taking actions and its effect on the rewards, Q-values, efficiency, and battery levels attained.

We then performed experiments with different learning rates in the Q-learning algorithm and analyzed their impact on the robot's performance concerning time and efficiency.

Finally, we used the law of large numbers to calculate the expected efficiency value and observed its relation with the learning rate.

Chapter 6

Simulation using PyBullet

We have trained the robotic arm according to our objective in the previous chapters. The motive ahead is to simulate the task of the pick-and-place robot to add a realistic attribute to our project, for which we have used the PyBullet module in the Python programming language.

6.1 PyBullet

PyBullet is a quick and convenient Python module facilitating simulation in robotics and machine learning, which can graphically mimic the physics involved in the real world. It provides forward and inverse kinematics, inverse dynamics computation, forward dynamics simulation, and collision detection. **[2]**

6.2 The Physics Server

To carry out the simulation, we need to connect to the physics servers, namely DIRECT and GUI, in PyBullet, built around a client-server-driven Application Programming Interface (API) where a client sends commands, and a physics server returns the status. In our simulation, we connect to the GUI server and set the gravity to -10 in the vertical direction. **[2]**

6.3 The Universal Robot Description File (URDF)

The Universal Robot Description File (URDF) is a file containing all the details regarding the structure of a robot or any particular shape, which includes the orientation and location of the links and the joints connecting any two links, the geometry, color, and collision, visual and inertial properties of the links **[2]**. Among any two links attached by a joint, one link is called the parent link, and another is the child link. We can load a robot or a shape into the simulation space by sending commands to the physics server, which uses the format specified in the URDF file concerned **[2]**. The URDF files of some robots or shapes are included in the PyBullet module by

default, whereas others have to be constructed as per requirement. In this simulation, we have used four different URDFs for these objects -

1. The Pick and Place Robot
2. Ten objects denoted by different colored small cuboids.
3. Black-colored large cubes marking the boundary of the 30*30 grid.
4. The plane on which all objects, including the robot, are placed.

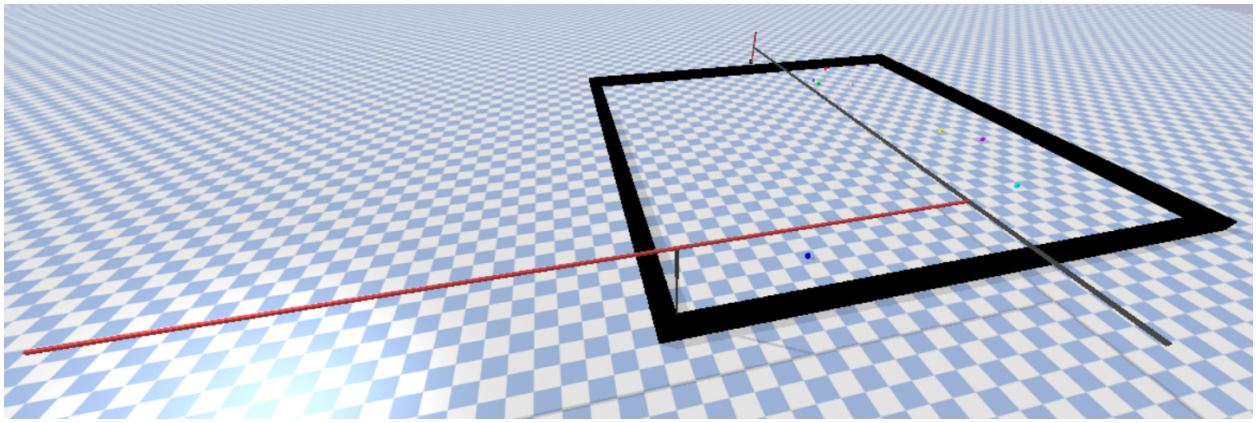

Figure 6.1 *The Pick and Place Robot.*

6.3.1 The URDF of Pick and Place Robot

We construct the URDF for the robot as elucidated in the following points -

- Six links denote the robot's parts connected by five prismatic joints. A prismatic joint facilitates a sliding movement between two links, and the moving link has one degree of freedom, i.e., it can move only along one axis specified.
- The base or the first link remains fixed and perpendicular to the plane. It is connected to the second link.
- The second link is perpendicular to the base link and can move horizontally to the plane, along one pair of opposite sides of the grid, covering its entire length (or breadth). It is connected to the third link.
- The third link is perpendicular to the second link and can move horizontally to the plane, along another pair of opposite sides of the grid, covering its entire breadth (or length). It is connected to the fourth link.
- The fourth link is perpendicular to the third link and can move vertically to the plane. It is connected to the fifth link.

- The fifth link is perpendicular to the fourth link and can move horizontally to the plane. It is connected to the sixth link.
- The sixth link is perpendicular to the fifth link and can move vertically to the plane. It functions as a lock to rigidly hold the picked-up objects.

6.3.2 The URDFs of Objects in the Grid

The objects are differentiated from each other by their color. We construct the URDFs of the ten objects to be arranged by the robot as described in the following points -

- Four cuboidal links constitute an object connected by three fixed joints. A fixed joint does not permit any movement between two links, i.e., each link has zero degrees of freedom.
- The base link lies horizontal to the plane. It is connected to the left and right links.
- The left and right links lie horizontally on the base link and are at some distance apart, thus leaving a tunnel-shaped hole across the object, passing through its center of mass. The right link is connected to the top link, which lies horizontally to the plane, on the left and right links.
- The size of the objects is less than that of the square so that they can be lucidly identifiable when placed at the center of a square.

The URDFs of large cubes and the plane are already present in the PyBullet module, but for simplicity, we change the color of the cubes to black and reduce the size of the plane's squares and the cubes by changing their geometric scale.

6.4 Reaching the Given Coordinates

The robot must first reach the coordinates of an object in the grid to pick or place it. The robot uses the hole present across each object to pick it. Since this tunnel-shaped hole lies horizontal to the plane, the robot maintains a small distance from the object's center of mass upon reaching the square containing it.

Reaching the coordinates is effectively carried out by the functions defined in PyBullet to control the target positions, maximum velocity, force, and damping oscillations of various joints between the two links they connect.

The second and third links attempt to reach the object's 'x' and 'y' coordinates, respectively. During this journey, the sixth link, i.e., the lock, is constantly applied an upward force to prevent it from sliding down due to gravity. This force is applied to the lock throughout the simulation except when the object is being locked, carried, or released.

6.5 Picking Mechanism

After reaching the square containing the object, the robot picks it up by a sequence of movements of links by controlling the associated joints as described -

- The fourth link descends up to a certain distance such that the fifth link comes in level with the hole of the object.
- The fifth link then moves horizontally and enters the hole. The sixth link moves upwards and locks the object to prevent it from sliding while carrying at another position.
- The fourth link ascends back to its position, and the robot successfully picks up the object.

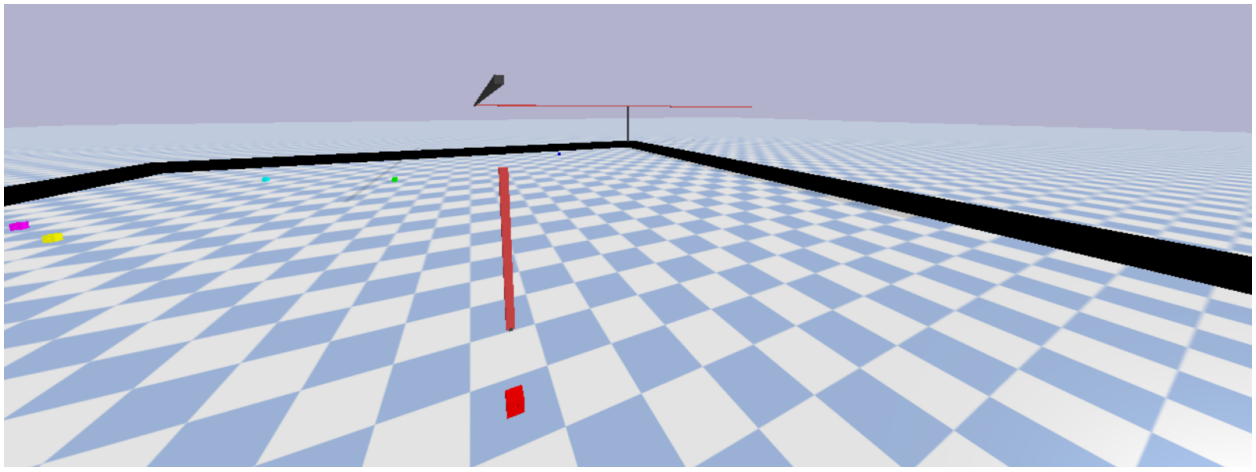

Figure 6.2 *The robot picking an object according to the defined mechanism.*

6.6 Placing Mechanism

After picking up the object, the robot must reach its correct position to place it. This reaching is implemented by the Q-learning algorithm used to train the robot in the Markov Decision Process, described in chapters 3 and 4. The sequence of movements of links by controlling the associated joints is -

- The fourth link descends up to a certain distance such that the object's base comes in contact with the plane.
- The sixth link moves downwards to unlock the object.
- The fifth link moves horizontally out of the hole, back to its position.
- The object is now successfully placed in its final position.
- The fourth link ascends back to its position, and the robot executes the picking and placing processes for the remaining objects.

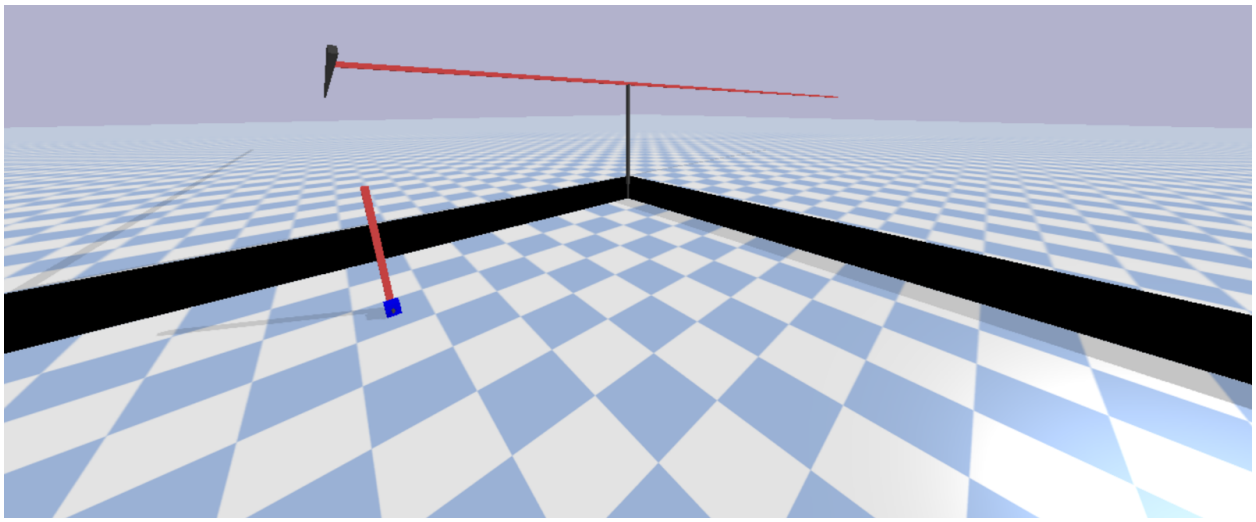

Figure 6.3 *The robot placing an object according to the defined mechanism.*

6.7 Project Link

The code and experiment for our project are available on the link below:

Google Drive Link

Chapter 7

Conclusions and Discussion

In this report, we focused on training the Pick and Place robot using the Q-learning algorithm, a classic reinforcement learning algorithm to solve a problem of the Markov Decision Process. The robot learned an optimal policy based on rewards from the environment to arrange all the misplaced objects in the least possible time and with sufficient battery level while carrying each object, without any external supervision.

The two major parameters in applying the Q-learning algorithm were the learning rate α and discount rate γ. The learning rate affected the Q-values, the rewards obtained in each episode, the efficiency achieved for arranging each object, and the average efficiency of the whole task. The discount rate affected the Q-values and the N-step return at each episode.

The robot achieved the maximum expected efficiency of nearly 83% at learning rates greater than or equal to 0.1.

7.1 Contributions

The pick and place robot in this report can have many valuable contributions to the industries, hospitals, and homes, such as -

- The robot integrated with computer vision technology can organize and select objects based on their size, shape, color, or barcode and place them in their desired location or can modify their orientation. It can also pick and remove defective products before they reach the subsequent production phase.
- The robot can be employed to clean a room by collecting useless objects lying on the floor and placing them in the bin. It can also be instructed to sort valuable things accidentally dropped in a bin and put them in their desired locations.
- The robot can find use in assembling incoming objects on a conveyor belt in industries, where it will pick an object and join or insert it into another thing.
- The robot can be used in the packaging process, where it shall grab the material and pack it into a container or place it on a conveyor belt for transportation purposes.

- The robot can be of significant use in transporting medicines and first-aid kits to the patients' rooms in the hospital by pick and place mechanisms.
- The robot can be operated in households to arrange misplaced things in a room or transport the stuff from one room to another.

The prominent advantages in all the above tasks using the pick and place robot are -

- The robot learns to execute the job in the least possible time without prior knowledge of optimal actions.
- It tracks its battery level and executes the task only with a safe level defined to avoid damaging the picked-up objects due to a sudden drop in case of a fully drained battery.
- It relieves human beings from highly mundane and tiresome jobs, enabling them to concentrate on more complicated work and alleviating the risk of any strain injury while doing a recurring task.

7.2 Future Directions

There are many other reinforcement learning algorithms applicable to train this robot, namely, State-Action-Reward-State-Action (SARSA), Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG), which can be possibly applied to train the robot in this report and can be the prospects of this project. The robot can also be equipped with a vision-guided system to find the objects on its own without specifying their initial positions. For example, in our experiment, the robot enabled with computer vision can take the image of the grid containing the objects and then analyze it to predict the coordinates of various things to execute the pick and place task. The reward can be decreased for inaccurately computing an object's coordinates while keeping the rest of the algorithm the same.

Bibliography

[1] Sutton R and Barto A (2014-15). *Reinforcement Learning: An Introduction*. London, England, The MIT Press, Cambridge, Massachusetts: 53-67, 157-158.

[2] Coumans E, Bai Y (2017-18). *PyBullet Quickstart Guide*. [online] usermanual.wiki. Available at:https://usermanual.wiki/Document/pybullet20quickstart20guide.479068914/view [Accessed 30th Apr. 2022].

[3] Liu Y, Hu Y, Gao Y, Chen Y and Fan C (2019) 'Value Function Transfer for Deep Multi-Agent Reinforcement Learning Based on N-Step Returns': *The Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*. [online] Nanjing, China, National Key Laboratory for Novel Software Technology, 458-459. Available at: https://www.ijcai.org/proceedings/2019/0065.pdf [Accessed 14th Mar. 2022].

[4] 'Law of large numbers' (2022). *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Law_of_large_numbers (Accessed: 20 April 2022).