Highly adaptive and efficient Q-learning based navigation system for large e-commerce warehouse ground robots

Md. Sazidur Rahman
dept. Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
md.sazidur.rahman@g.bracu.ac.bd

Md. Jisan Bin Islam
dept. Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
md.jisan.bin.islam@g.bracu.ac.bd

***Abstract*—With increasing number of major online retailers and its influence on the consumers, efficient and fast delivery systems in e-commerce warehouses has become indispensable. While demanding cost-effectiveness and swift maneuvering in the warehouse, these systems are required to be highly productive that can maintain the growing fast supply chain demand. In this paper, we consider a large warehouse where picking operations are performed by ground robots. We propose a highly efficient yet dynamic navigation system where robots learn to strategically navigate between the packaging area and all other locations where they are allowed to travel. Our model is highly adaptive to other industrial environments by changing demands and uncertainties while not limiting to only warehouses. To achieve this, we follow the reinforcement learning approach for solving the shortest path learning problem, known as Q-learning. In the recent years, reinforcement learning in path planning systems has been proven to be highly viable. We demonstrate our Q-learning based shortest path algorithm through a path optimization simulation of a warehouse robot.**

***Index Terms*—Q-Learning, E-commerce, Warehouse, Shortest path**

I. Introduction

Reinforcement learning is the area of machine learning concerned with methods that work well on a set of unsupervised problems. it involves learning what to do, how to map situations to actions and to maximize a numerical reward signal. In an essential way they are closed-loop problems because the learning system's actions influence its later inputs. The three main characteristics of reinforcement learning problems are, 1) -being closed-loop in an essential way, 2) not having direct instructions as to what actions to take, 3) and the consequences of actions, including reward signals, play out over extended time periods.

An agent must be able to sense the state of the environment to some extent and take actions that affect the state. The agent also must have a goal or goals relating to the state of the environment. The formulation is intended to include three aspects—sensation, action, and goal—in their simplest possible forms without trivializing any of them. Any method that is well suited to solving this kind of problem we consider to be a reinforcement learning method. In other words, Reinforcement learning is relatively a different paradigm in machine learning which can guide an agent how to act in the world. The interface to a reinforcement learning agent is much broader than just data, it can be the entire environment. That environment can be the real world, or it can be a simulated world such as a video game.[1]

The history behind the first industrial robot is when George Devol filed the first robotics patent in 1954 (granted in 1961), and his company, Unimation, produced the first industrial robot in 1956. That first robot was capable of moving material about a dozen feet or so. General Motors installed the first robot in a plant in New Jersey in 1962. For a long time, robots were only suited for work in industrial manufacturing because they were not safe for people to be around while they were in use.

The technology have improved over these years and the industries have learned to use these robots efficiently. Companies like Amazon, Alibaba, Jingdong etc. are using autonomous warehouse robots to run their business which cuts off 70 percentages of labour costs.

Path finding robots are used by all the renowned companies like Alibaba, JD, Amazon among others. The algorithms that are used in these robots are mostly static and rely heavily on the prior knowledge of the environment. There are many strategies or algorithms that has been followed by the researchers to provide such autonomous robots such as using Voting Q-Learning (VoQL),[2] Deep Q-learning combined with Convolution Neural Network(CNN) algorithm,[3] DRL algorithm with the traditional global path planning algorithm Probabilistic Roadmap (PRM) [4]etc.

In this paper, We consider a large warehouse where the e-commerce company would like warehouse robots to perform all the picking operations and delivery from the storage to the packaging area in the warehouse. Warehouse robots are autonomous ground vehicles that are designed to automatically handle many common warehouse related tasks. In the context of e-commerce warehousing, "picking" is known to be the task of collecting individual items from different locations in the warehouse storage to satisfy customer orders. First, the robots pick up the items from the inventory shelves. Then, they are required to bring the items to a specific location within the warehouse which is known to be the “packaging area”, where they can be packaged for shipping. To perform this task in a large environment, the robots will need to learn the most efficient and shortest path between the picking shelves, the packaging area and all other defined locations within the warehouse where the robots are allowed to move around. We demonstrate an improved adaptive model which uses Q-learning algorithm that can train itself to find the shortest path planing to reach its destination. The major contributions of this project are outlined as follows-

- It proposes a modified navigation planning system to find the shortest path specially designed for large warehouses.
- It demonstrates the highly adaptiveness of our system by altering uncertainties and supply chain demands regardless of the industry.
- Our proposed model greatly improves the accuracy of the shortest path algorithm in comparison with the previous path finding methods.

II. Related Work

There are vast amount of researches that have been put through in this path finding sector. Researcher used different method and algorithms to create an agent that can find the shortest path. For instance, in [2], the author suggests an algorithm which uses VotingQ-Learning (VoQL), a model-free, on-policy learning algorithm. He also explored different voting methods to be used with the VoQL algorithm such as Approval voting, Borda Countmethod, Copeland voting, Negative voting, and Plurality voting. The later two voting methods, Negative and Plurality voting were evaluated for the first time in the context of VoQL algorithm for action-selection in a reinforcement learning problem.
In[3], the authors proposes a multi-robot path planning algorithm using Deep q learning combined with CNN (Convolution Neural Network) algorithm. They experiment on different environments to demonstrate the flexible and efficient movement of the robots comparing with conventional methods. But they came across some unnecessary movement where the generated path is simple or without obstacles. To enhance their proposed algorithm, research on the potential field algorithm is still undergoing.
Similarly, in[4] the authors propose to combine the DRL algorithm Twin Delayed Deep Deterministic policy gradients (TD3) with the traditional global path planning algorithm Probabilistic Roadmap (PRM) as a novel path planner (PRM+TD3). They acknolegde the fact that even if the incremental training mode can improve the development efficiency, convergence and generalization of the DRL enabled robot, the PRM is a type of sampling algorithm in nature, so the results of each planning are different and might not be optimal.
Furthermore, the authors in[5], suggest a novel optimal path routing algorithm by using reinforcement learning techniques from AI. Reinforcement learning considers the given topology of nodes as the environment and leverages the given latency or distance between the nodes to determine the shortest path. They state that the implemented algorithm which is a modified Q-learning from reinforcement learning adaptively finds the optimal path for a given network topology. In[6] the authors efficiently use the knowledge of q-learning to update the entries in the Q-table once only by utilizing four derived properties of the Q-learning, instead of repeatedly updating them like the classical Q-learning. Their proposed algorithm stores the Q-value for the best possible action at a state, and thus saves significant storage.
The authors of [7] proposes the shortest path algorithm with Q-learning method where a stochastic state-transition model is used to store a previous observed state, a previous action and a current state. Whenever a robot reaches a goal, a Stochastic Shortest Path(SSP) will be found from the stochastic state-transition model. State-action pairs on the SSP will be counted as more significant in the action selection.
Our paper suggests an optimal method of using Q-learning that is less time consuming than previously mentioned methods that are used in an agent. have come across.

III. Methodology

A. System Representation

1) Environment: We have taken a 26X26 grid as the environment which is similar to a warehouse. The environment consists of 26 rows and 26 columns. There are some cells which are occupied with different kinds of goods colored in ’black’, the free paths which are white in color and lastly the packaging area or the goal state which is colored in green.[fig 1]

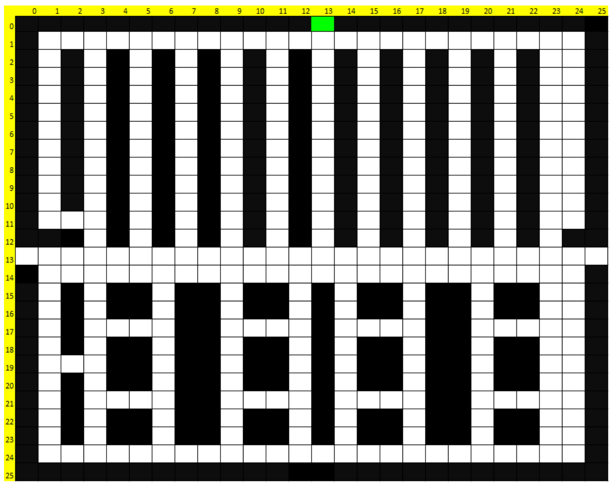
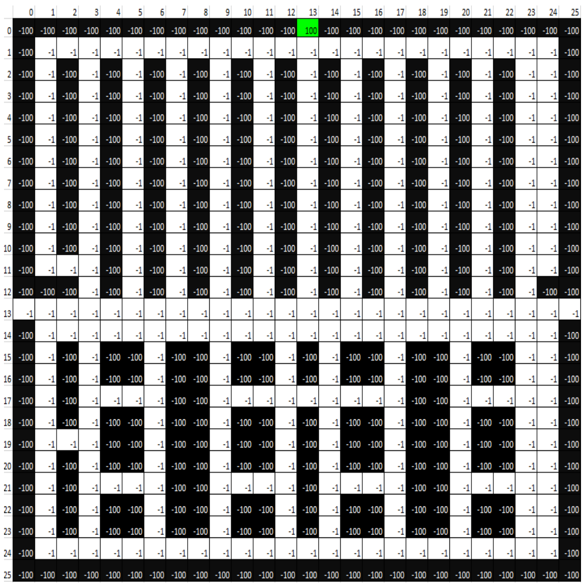


Fig. 1. Environment(Warehouse)

2) State: In the grid each of cell is represented here as a state. To illustrate, (13,12) cell is a state. There are total (26*26) = 676 states in total in the warehouse. We have also declared terminal states which are black in color. The terminal states mean that they are invalid and robot cannot go through it as it contains packages.

3) Action: We defined the action as the movement of the robot. Here the action of the robot is a tuple of (up,right,down,left). In our code, 0 will represent the robot will choose the upward state form its current position, if the action is 1, the robot will go to the right state. if the action is 2 the robot will move to the down state and if it gets 3 it will move to the left state from its current state.

4) Reward: The robot is given rewards and punishments based on its actions. Here we have set the reward of the terminal states -100, the rewards of the free paths -1 and the reward of the goal state is 100.[fig 1]



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100
1	-100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-100
2	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
3	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
4	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
5	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
6	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
7	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
8	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
9	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
10	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
11	-100	-1	-1	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-1	-100
12	-100	-100	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-1	-100	-100
13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
14	-100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-100
15	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-1	-100
16	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-1	-100
17	-100	-1	-100	-1	-1	-1	-1	-100	-100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-100	-100	-1	-1	-1	-1	-1	-100
18	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-1	-100
19	-100	-1	-1	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-1	-100
20	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-1	-100
21	-100	-1	-100	-1	-1	-1	-1	-100	-100	-1	-1	-1	-1	-100	-1	-1	-1	-1	-100	-100	-1	-1	-1	-1	-1	-100
22	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-1	-100
23	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-100	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-1	-100
24	-100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-100
25	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100

Fig. 2. Final Rewards(Warehouse)

B. Task Description

C. Q-Learning Method

To understand Q-learning method, firstly we need to know about some concepts. The concepts are given below,

1) MDP- Markov Decision Process: MDP helps us to find a way to formalize sequential decision making. This formalization is the basis for structuring problems that are solved with Reinforcement learning.
In MDP, we have a decision maker called agent, that interacts with the environment(in our case the agent will be the robot). These interactions occurs sequentially over time. At each time step, the agent will get some representation of the environment's state. Given this representation, the agent selects an action to take. The environment then moves into a new state and the agent is given a reward as a consequence of the previous action.
There are several components of MDP,

- Agent
- State
- Action
- Reward

This process of selecting an action from a given state, moving to a new state and receiving a reward happens sequentially over and over again, which creates something called a trajectory that shows the sequence of states, actions and rewards. Throughout this process, it is the agent's goal to maximize the total amount of rewards that it receives from taking actions in given states. This means that the agent wants to maximize not just the immediate reward, but the cumulative rewards it receives over time.

2) MDP Notation: In an MDP, we have a set of states S , a set of actions A and a set of rewards R. We will assume that each of these sets has a finite number of elements. At each time step $t = 0, 1, 2, ...$, the agent receives some representation of the environment's state $S_t \in S$. Based on this state, the agent selects an action $A_t \in A$. This gives us the state-action pair (S_t, A_t).
The time is then incremented to the next time step $t + 1$ and the environment is moved to a new state $S_{t+1} \in S$. At this time, the agent receives a numerical reward $R_{t+1} \in R$ for the action A_t taken from state S_t.
We can think of the process of receiving a reward as an arbitrary function f that maps state-action pairs to rewards. At each time t, we have

$$f(S_t, A_t) = R_{t+1}.$$

The trajectory representing the sequential process of selecting an action from a state, transitioning to a new state and receiving a reward can be represented as,

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$$

The diagram below shows the entire idea nicely,

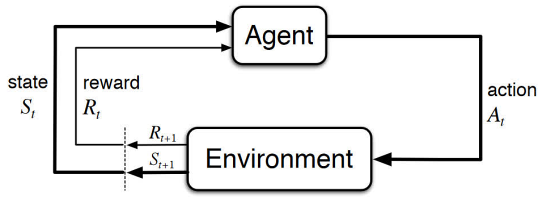


Fig. 3. MDP Diagram

If we break down this diagram, we get

- At time t, the environment is in state S_t.
- The agent observes the current state and selects action A_t.
- The environment transitions to state S_{t+1} and grants the agent reward R_{t+1}.
- This process then starts over for the next time step, $t + 1$.

3) Transition Probabilities: Since the sets S and R are finite, the random variables R_t and S_t have well defined probability distributions. In other words, all the possible values that can be assigned to R_t and S_t have some associated probability. These distributions depend on the preceding state and action that occurred in the previous time

step $t-1$.
For all $s' \in S, s \in S, r \in R$ and $a \in A(s)$, we define the probability of the transition to state s' with reward r from taking action a in state s as:

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

4) Expected Return: We can think of the return simply as the sum of the future rewards. Mathematically, we define the return G at time t as

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$$

where T is the final time step. This concept of the expected return is important because it's agent's objective to maximize the expected return. The expected return is what's driving the agent to make the decisions it make.

5) Episodic Vs. Continuing Tasks: In expected return, we have seen T, the final time step. When the notion of having a final time step makes sense, the agent-environment interaction naturally breaks up into sub-sequences, called episodes.
There exits other types of tasks though where the agent-environment interactions do not break up naturally into episodes, but instead continue without limit. These types of tasks are called continuing tasks.
Continuing tasks make our definition of the return at each time t problematic because our final time step would be $T = \infty$, and therefore the return itself could be infinite since we have

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$$

Because of this, we need to refine the way we are working with the return.

6) Discounted Return: To define the discounted return, we first define the discount rate γ, to be a number between 0 and 1. The discount rate will be the rate for which we discount future rewards and will determine the present value of future rewards. With this, we define the discounted return as,

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

This definition of the discounted return makes it to where our agent will care more about the immediate reward over future rewards since future rewards will be more heavily discounted. So, while the agent does consider the rewards it expects to receive in the future, the more immediate rewards have more influence when it comes to the agent making a decision about taking a particular action.

7) Policies: A policy is a function that maps a given state to probabilities of selecting each possible action from that state. We will use the symbol π to denote a policy.
When speaking about policies, formally we say that an agent "follows a policy". For instance, if an agent follows policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ is $S_t = s$. This means that, at time t, under policy π, the probability of taking action a in state s is $\pi(a|s)$.

8) State-Value Function: The state-value function for policy π, denoted as v_π, tells us how good any given state is for an agent following policy π. In other words, it gives us the value of a state under π.
Formally, the value of state s under policy π is the expected return from starting state s at time t and following policy π thereafter. Mathematically we define $v_\pi(s)$ as,

$$v_\pi = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

9) Action-Value Function: Similarly, the action-value function for policy π, denoted as q_π, tells us how good it is for the agent to take any given action from a given state while following policy π. In other words, it gives us the value of an action under π.
Formally, the value of action a in state s under policy π is the expected return from starting from state s at time t, taking action a, and following policy π thereafter, Mathematically, we define $q_\pi(s, a)$ as

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

10) Optimal Policy: In terms of return, a policy π is considered to be better than or the same as policy π' if the expected return of π is greater than or equal to the expected return of π' for all states. In other words,

$\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$

11) Optimal State-value function: We denote the optimal state-value function as v_* and define as

$$v_*(s) = \underset{\pi}{max}\ v_\pi(s)$$

for all $s \in S$. In other words, v_* gives the largest expected return achievable by any policy π for each state.

12) Optimal Action Value Function: Similarly, the optimal policy has an optimal action-value function or optimal Q-function, which we denote as q_* and define as

$$q_*(s, a) = \underset{\pi}{max}\ q_\pi(s, a)$$

for all $s \in S$ and $a \in A$. In other words q_* gives the largest expected return achievable by any policy π for each possible state-action pair.

13) Bellman Optimality Equation for Q_*: One fundamental property of q_* is that it must satisfy the following equation,

$$q_*(s, a) = E[R_{t+1} + \gamma\ \underset{a'}{max}\ q_*(s', a')]$$

This is called Bellman optimality equation. It states that, for any state-action pair (s, a) at time t, the expected return from starting in state s, selecting action a and following the optimal policy thereafter is going to be the expected reward we get from taking action a in state s, which is R_{t+1}, plus the maximum expected discounted return that can be achieved from any possible next state-action pair (s', a').
We are going to see how we can use the Bellman equation to find q_*. Once we have q_*, we can determine the optimal policy because, with q_* for any state s, a reinforcement learning algorithm can find the action a that maximizes $q_*(s, a)$.

14) Q-Learning Objective: The objective of Q-learning is to find a policy that is optimal in the sense that the expected value of the total reward overall successive steps is the maximum achievable. So, in other words, the goal of Q-learning is to find the optimal policy by learning the optimal Q-values for each state-action pair.

15) Storing Q-values in a Q-table: We'll be making use of a table, called a Q-table, to store the Q-values for each state-action pair. The horizontal axis of the table represents the actions, and the vertical axis represents the states. So, the dimensions of the table are the number of actions by the number of states.
As the Q-table becomes updated, in later moves and later episodes, the lizard can look in the Q-table and base its next action on the highest Q-value for the current state.

D. Algorithm

The proposed algorithm is presented in **Algorithm 1** section where each and every step of the code is discussed. For readers understandability purpose the terms are descried below, -

- Terminal states denoted in black (fig 1), item storage location.
- Free paths of the environment denoted in white where robots can travel.
- Goal state denoted in green, item packaging area.
- Reward of the terminal states is -100.
- Reward of the free path or white state is -1.
- Actions are expressed as up, right, down, and left.
- 10000 episodes for training and finally result is fetched from getShortestPath function.

Algorithm 1: Algorithm of the full system

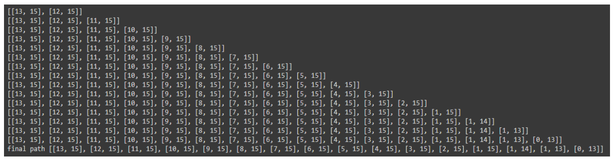
```
Define qvalues array;
Define action array;
Define the location of free paths in aisles array;
Set the reward of the free path(-1);
function getShortestPAth(startRowID,startColId){
    return immediately if this is a starting location;
if this is a legal starting position then
    continue moving along the path until we reach the
      goal;
else
end
}
for episode 1 to 10000{
    rowId,colId = getStartingLocation;
}
while it is not a terminal state do
    actionId = getNextAction;
    reward = rewards[rowId,colId];
    oldQValue = qValues[oldRowId, oldColId,
      actionId];
    temporalDifference = reward + (discountFactor *
      np.max(qValues[rowId, colId])) - oldQValue;
end
print the final shortest path from starting to goal state;
```

E. Implementation Details

Google Colaboratory was used to implement this system. This platform is found to be the best for this system as there are not enough resources in the local computer.
The training parameters are used as follows in this system.

- Learning rate as 0.9
- Discount factor as 0.9
- Epsilon as 0.9

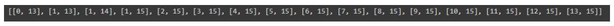
Here Epsilon is the percentage of time when we should take the best action.
In this system the AI agent will learn for 10000 episode and then gives the best result it can obtain.For example, if we give (13,15) as the starting cell it will give the following result after the full training is completed.[fig 4]



```
[[13, 15], [12, 15]]
[[13, 15], [12, 15], [11, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15], [4, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15], [4, 15], [3, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15], [4, 15], [3, 15], [2, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15], [4, 15], [3, 15], [2, 15], [1, 15]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15], [4, 15], [3, 15], [2, 15], [1, 15], [1, 14]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15], [4, 15], [3, 15], [2, 15], [1, 15], [1, 14], [1, 13]]
[[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15], [4, 15], [3, 15], [2, 15], [1, 15], [1, 14], [1, 13], [0, 13]]
final path [[13, 15], [12, 15], [11, 15], [10, 15], [9, 15], [8, 15], [7, 15], [6, 15], [5, 15], [4, 15], [3, 15], [2, 15], [1, 15], [1, 14], [1, 13], [0, 13]]
```

Fig. 4. Final output

Again if we want to go to a specific position from the goal state we can go by reversing the shortest path. For instance, if we give command state (13,15). The output will be the shortest path from the packaging area or the goal state (0,13). [fig 5]



```
[[0, 13], [1, 13], [1, 14], [1, 15], [2, 15], [3, 15], [4, 15], [5, 15], [6, 15], [7, 15], [8, 15], [9, 15], [10, 15], [11, 15], [12, 15], [13, 15]]
```

Fig. 5. Output of the reversed path

IV. Simulation Results and Discussions

We have tested our system in google colaboratory and found our method to be one of the fastest. It gives the correct shortest distance within a very short span of time. The accuracy rate is also remarkable. As before, if we give the input of (13,15), the shortest path is as follows[fig 6]

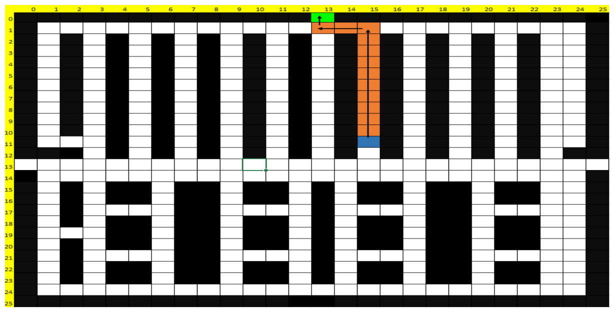


Fig. 6. Indicating the shortest path

It converges in 9997 episodes for this particular input and takes only 2 seconds to give the optimal path to the goal state from the starting state. Again, if we give (24,15) as the input it converges in 9999 episodes and takes only 2 seconds to show the shortest path. From these result we can say that our algorithm can find the optimal path very efficiently which will help the delivery companies to deliver products faster than before as the robots are taking products from product shelf to the packaging area a lot faster than ever. Besides, the robots can go to specific places from the packaging area after giving the package. So, this Q-learning method is very ease to use and also it is very user friendly and it cuts off the labour cost significantly and can work 24/7.

V. Conclusion & Future Work

Warehouse robots are such an invention that will change the e-commerce business sector completely. There are a lot of companies that are using warehouse robots that can perform specific actions in different ways. But our system not only takes less time to find the shortest path to reach its goal from the starting state but also can overcome any unusual situation in the environment. As the result shows that the Q-learning algorithm converges faster than other traditional algorithms like VoQL, CNN , DRL etc. Our system is not only limited to use in warehouse but also can be used in other places like restaurants, offices, households etc.

Due to lack of time and resources, we could not add automatic charging system of the agent. Our future goal of this project is to add this feature to make fully autonomous warehouse system that can help to grow the e-commerce businesses.

References

[1] Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.

[2] Thombre, Prashant, "Multi-objective Path Finding Using Reinforcement Learning" (2018). Master's Projects. 643.

[3] Bae H, Kim G, Kim J, Qian D, Lee S. Multi-Robot Path Planning Method Using Reinforcement Learning. Applied Sciences. 2019; 9(15):3057.

[4] Gao J, Ye W, Guo J, Li Z. Deep Reinforcement Learning for Indoor Mobile Robot Path Planning. Sensors. 2020; 20(19):5493.

[5] R. Nannapaneni, "OPTIMAL PATH ROUTING USING REINFORCEMENT LEARNING",2020

[6] A. Konar, I. Goswami Chakraborty, S. J. Singh, L. C. Jain and A. K. Nagar, "A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 43, no. 5, pp. 1141-1153, Sept. 2013, doi: 10.1109/TSMCA.2012.2227719. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] Wooyoung Kwon, Il Hong Suh, Sanghoon Lee and Young-Jo Cho, "Fast reinforcement learning using stochastic shortest paths for a mobile robot," 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007, pp. 82-87, doi: 10.1109/IROS.2007.4399040.

[8] S. Saadatmand et al., "Autonomous Control of a Line Follower Robot Using a Q-Learning Controller," Proceedings of the 10th Annual Computing and Communication Workshop and Conference (2020, Las Vegas, NV), Institute of Electrical and Electronics Engineers (IEEE), Jan 2020. The definitive version is available at https://doi.org/10.1109/CCWC47524.2020.9031160