DYNAMIC PATH PLANNING OF UNKNOWN ENVIRONMENT USING DOUBLE Q-LEARNING ALGORITHM

A dissertation report Submitted in partial fulfilment of the requirement for the award of Dual Degree (B. Tech + M. Tech)

Specialization in

(Artificial Intelligence and Robotics)

Submitted by:

Gautam khattri Enrolment No: 15/ics/019

UNDER THE SUPERVISION OF

Prof. Sanjay Sharma
(Professor)

SCHOOL OF INFORMATION & COMMUNICATION TECHNOLOGY, GAUTAM BUDDHA UNIVERSITY, GREATER NOIDA

DEC, 2019

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

GAUTAM BUDDHA UNIVERSITY, GREATER NOIDA, 201312, U. P., (INDIA)

Candidate's Declaration

I, hereby, certify that the work embodied in this dissertation entitled **"DYNAMIC PATH PLANNING USING DOUBLE Q-LEARNING ALGORITHM"** by **Mr. Gautam Khattri** having Enrolment No. **15/ics/019** in partial fulfilment of the requirements for the award of the dual degree of B. Tech (Computer Science Engineering) + M. Tech with Specialization in Artificial Intelligence and Robotics Submitted to the School of Information and Communication Technology, Gautam Buddha University, Greater Noida is an authentic record of my own work carried out under the supervision of **Prof. Sanjay Sharma (Professor)** School of ICT. The matter presented in this dissertation has not been submitted in any other University / Institute for the award of any other degree or diploma. Responsibility for any plagiarism related issue stands solely with me.

Student Name: Gautam Khattri **Signature of Student:** …….

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief. However, responsibility for any plagiarism related issue solely stands with the student.

Signature of the Supervisor:

Name with Designation**: Prof. Sanjay Sharma**
(Professor)

Date:

Place: Greater Noida

ACKNOWLEDGEMENT

It gives me great pleasure to acknowledge my indebtedness and deep sense of gratitude to **Prof. Sanjay Sharma (Professor)** for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project. I would also like to thank my friends for their invaluable and useful comments and voluntary help from time to time.

I would also like to express my gratitude towards my family for their persistent encouragement, patience and untiring help during the period of my study.

I would also like to convey my thanks and appreciations to **Dr. Pradeep Kumar Yadav (DEAN of ICT)** and **Dr. Pradeep Tomar (HOD of ICT)** in developing the project and sharing their experience in making this project worthy of acclamation.

And last but not the least i am very thankful to the almighty God without whose blessings nothing would be possible.

ABSTRACT

Dynamic path planning of a network present in an unknown environment has been a difficult task for certain robots. In this paper, I applied a Double Q network algorithm which comes under deep reinforcement learning proposed by Deepmind in the year 2016. Reinforcement learning has been a major subject of interest for research and its application in various real-world domains such as Robotics, Games, Industrial Automation and Control, System Optimization, Quality control and Maintenance. The reward and punishment function and the training method are designed for the instability of the training stage and the sparsity of the environment state space. It is a type of Machine learning Paradigms in which a learning algorithm is trained not on the preset data given but rather based on a feedback system. These machine learning algorithm such as DDQN has be shown as the future of Machine Learning as these eliminate the cost of collecting and cleaning the data. But there are some important areas where Reinforcement learning could be a vital need. This paper's aim is to turn the attention to the major problem of Dynamic Optimal path in case of emergencies. In this paper, I'm going to demonstrate how to implement a simple Deep Reinforcement Learning algorithm which is called the Double Q-Learning technique. With the updating of neural network and the increase of greedy rule probability, the local space searched by agent is expanded. Pygame module in PYTHON is used to establish dynamic environments. Considering lidar signal and local target position as the inputs, convolutional neural networks (CNNs) are used to generalize the environmental state. Q-learning algorithm enhances the ability of the dynamic obstacle avoidance and local planning of the agents in environment. The results show that, after training in different dynamic environments and testing in a new environment, the agent is able to reach the local target position successfully in unknown dynamic environment.

Keywords: Double DQN technique, ML model, Deep Q-Networks, Motion Planning, Reinforcement Learning.

LIST OF ABBREVIATIONS

DDQN	Dueling Deep Q-Network
ReUL	Rectified Linear Unit
CNN	Convolutional neural network
ML	Machine Learning
CNN	Convolutional Neural Network
RGB	Red Green Blue
RL	Reinforcement Learning
MORL	Multi Objective Reinforcement Learning
DQN	Deep Q Network
CQL	Classical Q learning
EQL	Extended Q learning

LIST OF FIGURES

LIST OF TABLES

CONTENTS

CHAPTER-1. INTRODUCTION

This chapter gives the brief introduction about the proposed algorithm and the dissertation. Here i will discuss about current algorithm that is used in finding the optimal path and i will compare the current algorithm with the proposed Q learning algorithm.

1.1 Introduction

Path Finding is a vastly studied subject in the field of Computer Science. The problem of path-finding is defined as the discovery and plotting of an optimal route between two points on a plane. The existing algorithms that solve this problem are mostly static and rely heavily on the prior knowledge of the environment. They also require the environment to be deterministic. However, in real-world applications of the path-finding problem, often the environment is priorly unknown and stochastic, and with several conflicting objectives. In such cases, the aforementioned algorithms fail to produce effective results. In this dissertation, I use a reinforcement learning approach for solving the many-objective path-finding problem. In this project, a set of optimal policies is determined with the help of the DDNQ algorithm. This algorithm uses various probability methods borrowed from the field of social choice theory for action-selection. In addition to working with the existing methods for DDNQ, the performance of additional voting methods is studied and evaluated for the first time. Machine learning is often used in mobile robots to make the robot aware about its world map. In early research on mobility management of robots, supervised learning was generally employed to train a robot to determine its next position in a given map from the sensory readings about the environment. Supervised learning is a good choice for mobility management of robots in fixed maps. Reinforcement learning is an alternative learning policy, which rests on the principle reward and punishment. No prior training instances are presumed in reinforcement learning. A learning agent here does an action on the environment, and receives a feedback from the environment based on its action. The feedback provides an immediate reward for the agent. The learning agent here usually adapts its parameter based on the current and cumulative (futuristic) rewards. Since the exact value of the futuristic reward is not known, it is guessed from the knowledge about the robot's world map. The primary advantage of reinforcement learning lies in its inherent power of automatic learning even in presence of small changes in the world map.

1.1.1 Current Work

Since deep reinforcement learning was first proposed in 2013 formally, tremendous progress has been made in the field of artificial intelligence. Deep Q-network agent was demonstrated to be able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester in Atari 2600 games. AlphaGo zero has defeated all previous AlphaGo versions by self-play without using any human chess spectrum. An agent can be trained to play FPS game receiving only pixels and game score as inputs. The aforementioned examples fully demonstrate the great potential in autonomous decision-making field after the reinforcement learning of neural network solving the problem of the curse of dimensionality. In, deep reinforcement learning has been applied to autonomous navigation based on the inputs of visual information, which has achieved remarkable success. In [11], The authors analyze the agent behaviour in static mazes feature complex geometry, random start position and orientation, and dynamic goal locations. Their results show that their approach enable the agent navigate within large and visually rich environments that include frequently changing start and goal locations, but the maze layout itself is static. The authors did not test the algorithm in environments with moving obstacles. If there are moving obstacles in environments, this means the images collected by cameras may be unknown for each episode. In [12], Yuke Zhu et al. try to find the minimum length sequence of actions that move an agent from its current location to a target that is specified by an RGB image. To solve the problem of a lack of generalization, i.e., the network should be retrained for new targets, they specify the task objective (i.e., navigation destination) as inputs to the model and addresses problems by introducing shared Siamese layers to the network. But as it is mentioned in the paper, in the network architecture of deep Siamese actor-critic model, the ResNet-50 layers are pretrained on ImageNet and fixed during training. This means that they have to collect a large number of different target and scene images with a constrained background to pretrain the ImageNet before training the navigation model, indicating that the generalization ability is still conditioned to the information of maps in advance. If i consider the environment to be a graph, then, at its very core, all the path finding problems address the question of how to reach a destination node from a starting node in a graph. This can be done by implementing a graph search algorithm for the given problem, which

searches the graph starting at an arbitrary node and exploring the adjacent vertices of the visited nodes until it reaches the destination node. The problem of finding any path between two nodes in a graph is just one of the two primary questions that path finding tries to answer. The other question that can be answered by a path finding problem is that of determining the most optimal route between the start and the destination node. Most frequently, we are concerned with the later problem, where we have to find an optimal path that avoids obstructions and minimizes the deviations due to those obstructions from the optimal path as much as possible before reaching the destination. Since, this is an optimization problem it is more complex and difficult to solve as compared to finding any path between the two nodes. Different strategies are applied to solve the optimal path finding problem such as using greedy technique in Dijkstra's algorithm, dynamic programming technique in Bellman-Ford algorithm, use of heuristics in A* search algorithm, etc. In most cases, algorithms for path finding problems have an inherent assumption that there is only one objective that the software program is trying to achieve such as, minimizing the overall distance travelled or reducing the time taken to travel from one point to the other. It is also considered that the environment will be deterministic and fully known in advance. However, in real-world applications we frequently work with initially unknown and stochastic environments. Also, it is more likely that instead of a single objective, the problem at hand needs to consider multiple conflicting objectives. The validity of the solution provided by above mentioned algorithms becomes void when the environment changes. Reinforcement learning (RL) is a good technique that deals with such stochastic environments and the multiple objectives can be evaluated by establishing a Pareto dominance relation among themselves. Thus, Multi-Objective Reinforcement Learning (MORL) method can be thought of as a combination of these two techniques. In reinforcement learning, Deep Q-learning is a technique that is used to find an optimal action-selection policy. To evaluate our algorithm, I used Python to build the DDQN training frameworks for simulation and demonstrate the approach in real world. In simulations, the agent is trained in a lower-level and intermediate dynamic environment. The starting point and target point are randomly generated to ensure diversity and complexity of local environment, and the test environment is a high-level dynamic map. We show details of the agent's performance in an unseen dynamic map in the real world.

1.1.2 Proposed Work

In this paper, I will be presenting a novel path planning algorithm and solve the generalization problem by means of local path planning with deep reinforcement learning DDQN based on lidar sensor information. In the aspect of the recent deep reinforcement learning models, the original training mode results in a large number of samples which are moving states in the free zone in the pool, and the lack of trial-and-error punishment samples and target reward samples ultimately leads to algorithm disconvergence. So, we constrain the starting position and target position by randomly setting target position in the area that is not occupied by the obstacles to expand the state space distribution of the pool of sample. To evaluate our algorithm, I used TensorFlow to build the DDQN training frameworks for simulation and demonstrate the approach in real world. In simulations, the agent is trained in a lower-level and intermediate dynamic environment. The starting point and target point are randomly generated to ensure diversity and complexity of local environment, and the test environment is a high-level dynamic map. We show details of the agent's performance in an unseen dynamic map in the real world.

1.2 Motivation

Optimal Path finding problems can be solved using algorithms like Dijkstra's algorithm, A* search algorithm, Simulated Annealing, etc. But for these algorithms, most of the times it is assumed that there is only one objective to be considered and that the problem environment is fully known in advance and is deterministic. For example, in a simple path finding problem, it is assumed that the agent only wants to reach the goal state in minimum number of steps regardless of any other factors. Also, most of the times the environment is a grid of a fixed size which is known in advance before the algorithm even starts looking for a solution. We need to have some form of a priori knowledge about the problem domain. This includes, but is not limited to, the help from domain experts before or during the algorithm execution to guide the learning agent or setting up a predetermined solution preference based on which the agent will model its decisions. Also, it is not always feasible to set up a predetermined solution when we do not have a complete idea about the environment. Thus, this method is also ineffective in situations where the end goal or the optimal solution is not known in advance. All the above points force us to consider an

exploratory algorithm such as Reinforcement Learning to find a solution to the path finding problem in an unknown environment. All the above points force us to consider an exploratory algorithm such as Deep Reinforcement Learning to find a solution to Dynamic path problem.

1.3 Objectives

The dissertation includes the following objective:

- Study the effectiveness of Deep Q learning algorithm for action selection in any unknown environment with many conflicting objectives.
- Implement and analyse each Path possible in a given network.
- Analyse the reward set to each node in the network and find it's probabilities of success.

1.4 Problem Definition

The problem of finding an optimal path between two points on a plane can be solved using several algorithms and/or techniques as mentioned earlier in the introduction. However, the study in this project is focused specifically on a reinforcement learning approach to path finding called as Multi-Objective Reinforcement Learning with Deep Q-Learning. The problem occurs in the calculation of dynamic path when the edges and node are connected in different manner and the optimal reward is not maximized. The learning algorithm is basically not able to work when the environment is not known to the algorithm so it is supposed to have a priori knowledge about the network. In a network if the number of nodes is more and the attribute are more with a large complexity then it can cause a problem from the network every state has got fixed action that has to be performed in order to reach the given goal. Naturally, if the environment is changed after learning then the agent has to learn the changed environment again. This problem is overcome in the algorithm proposed here. This research paper measure how well a reinforcement learning technique such as Deep Q learning can be applied to the network for best optimal path and in my research, I will be modifying the classical Q learning algorithm, hereafter called improved Deep Q-learning for increasing its performance one in the path planning problem

1.5 Organization of the Dissertation

The material presented in this dissertation is organized into 4 chapters. Each chapter is divided in multiple segments where it explains each segment in brief. It offers a set of guidelines intended to help the readers to know about the different chapters in a short description.

Chapter 1. Introduction: This chapter gives the introduction about the dissertation and also explains the work that is already done in past related to this dissertation, it also explains about the current system that is used to get the results that are expected from our Q learning algorithm which is described in the Current system.(pp 1-6)

Chapter 2. Literature Review: This chapter includes all the contribution of the studied research paper and a brief summary of the terminology used in the deep reinforcement learning algorithm it also describes the related literature work of the dissertation. It describes the Q learning algorithm with it's code and explains the reinforcement learning framework. (pp 7-11)

Chapter 3. Terminology: This chapter explains the terms used in the dissertation like Reinforcement learning and also discusses about the Path planning algorithm with different variable and it explains how the algorithm uses the reward and action statement to get the best dynamic path available in the network. (pp 12-20)

Chapter 4. Implementation: This chapter discusses about How the Deep neural network is used with convolutional neural network to find a optimal path and to detect the best way to go around the environment by detecting the bad states with use of lidar signals. It also illustrates the different set of environments in which the system is tested. (pp 21-25)

Chapter 5. Conclusion and Future Scope: This chapter compiles all the information together from different subgroups that met during the dissertation and it summarizes the main points of evidence for the reader. It also discusses its future enhancement and a way to enhance the network capability by handling large number of attributes. (pp 26-27)

List of References: It shows all the References and link of various research paper that has been used to make this dissertation paper it also includes various citations of the online research paper. (pp 28-29)

CHAPTER-2. LITERATURE REVIEW

Various research papers, review papers have been studied about the Q learning algorithm in order to make this dissertation paper. It also provides a brief summary about all the studied papers and the scope of this research paper including its contribution to this field of study.

2.1 Related Literature

In this paper, we present a novel path planning algorithm and solve the generalization problem by means of local path planning with deep reinforcement learning DDQN based on lidar sensor information. In the aspect of the recent deep reinforcement learning models, the original training mode results in a large number of samples which are moving states in the free zone in the pool, and the lack of trial-and-error punishment samples and target reward samples ultimately leads to algorithm disconvergence. So, we constrain the starting position and target position by randomly setting target position in the area that is not occupied by the obstacles to expand the state space distribution of the pool of sample. To evaluate our algorithm, we use TensorFlow to build the DDQN training frameworks for simulation and demonstrate the approach in real world. In simulations, the agent is trained in a lower-level and intermediate dynamic environment. The starting point and target point are randomly generated to ensure diversity and complexity of local environment, and the test environment is a high-level dynamic map. We show details of the agent's performance in an unseen dynamic map in the real world. The training network in based on the Reward function and reward as mentioned in the studied research papers that when the learning agent moves from one state to the next state in the environment, it receives some reward for taking that action from the environment. This reward is used to update the policy that the agent follows at each state. In any reinforcement learning setting, so we need to define a reward function specific to the Reinforcement Learning problem context. Thus, the reward function is always contextual because it changes with the change in the problem context. In case of single objective RL problem, the reward function maps each state-action pair to a single numerical value. Thus, the reward is simply a scalar value. The objective of the RL-agent is to maximize the total reward it receives in the long run. This implies

that there is a direct correlation between the reward received and the quality of the determined policy. Single objective Deep Q learning algorithm is given as Algorithm 1:

Algorithm 1: Single Objective Deep Q learning Algorithm

Initialize the environment

Begin

Initialize **Q(s,a)**

For each episode e **do**

Initialize **s**

While s is not terminal **do**

Choose action **a** to perform in state **s** using policy derived from the current Q values

Perform action a, receive reward r and next state s' from the environment

Use the update rules as follows:

$$\hat{Q}(s,a) = (1 - \alpha t)\hat{Q}(s,a) + \alpha t(r + \gamma \max a' \hat{Q}(s', a'))$$

Update the current state value to s'

end

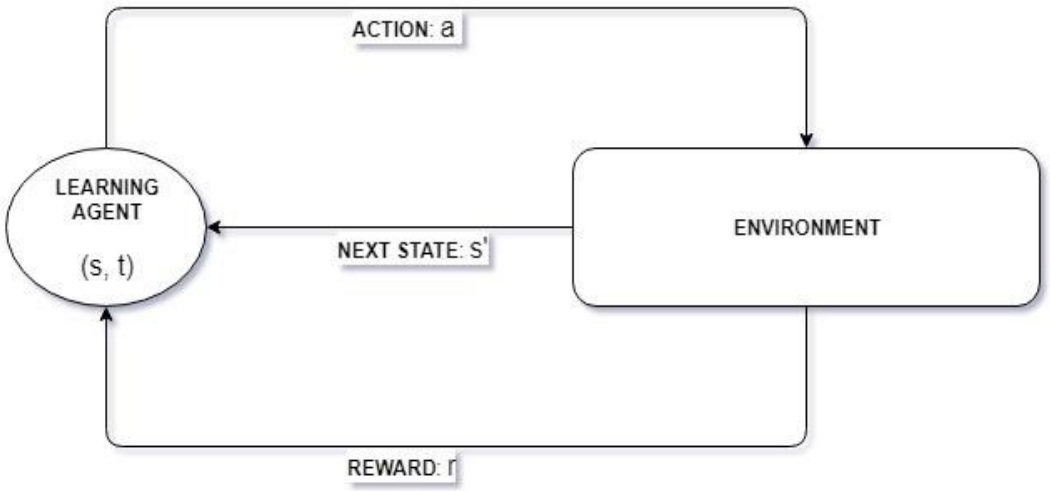

Fig. 1: Reinforcement framework network

Here, the Figure 1 describes the single objective reinforcement learning framework. Where the agent, represented by an oval, is in state s at time step t. The agent performs the action a based on the Q-value. The environment receives this action and in response it returns the

next states to the agent and a corresponding reward r for taking the action a. my research applies reinforcement learning techniques to real-world robots. Reinforcement learning has been tested in many simulated environments but on a limited basis in real-world scenarios. A real-world environment poses more challenges than a simulated environment, such as enlarged state spaces [2], increased computational complexity, significant safety issues (a real robot can cause real damage), and longer turnaround times for results. This research measures how well reinforcement-learning technique, such as Q-learning, can apply to the real robot for navigational problem, and in our research, i modified the classical Q-learning algorithm, hereafter called improved Q-learning for increasing its performance one in the path-planning problem. In [9], the authors employed an interesting strategy of internal prediction/estimation to control the balance between exploration and exploitation for efficient adaptability of an agent in a new environment. Here, a reliability parameter RI has been introduced as an internal variable to estimate the 'expected prediction error'. The variable RI is updated after every action by comparing itself to the actual error. The RI is introduced in the expression of Boltzman exploration by replacing temperature parameters T by RI=R(S)/η, where η is a positive constant and S denotes the current state of the agent. The factor η is used to normalize the magnitude of Q(s, a) with R(s). In the context of path-planning by a mobile robot, the robotic agent usually has additional knowledge about the distance from the current state to both the next state and the goal. This knowledge has been efficiently used here during the learning phase of the robotic planner to speed up learning through greedy selection of actions. Four properties (rules) concerning computation of Q-values at state S/ from the current estimate of Q-value at state S, where S/ is a neighbour of state S, has been developed. Each rule has a conditional part and an action part. If the conditional part is found true, the action part is realized. The conditional part involves checking a locking status of a state along with a distance comparison, where locking of a state indicates that its Q-value needs no further updating. The action part ensures that the Q-value at S/ can be evaluated in one step only, and the state S/ will also be locked. Thus, when conditional part of a property is activated, Q-value at a new state is evaluated once only for ever, and the new state is locked. The proposed improved Q-learning algorithm is terminated when all the states in the workspace are locked. It is apparent from the above discussion that exploration in the improved Q-learning algorithm takes place when all the

rules' conditional parts do not satisfy at a given situation, whereas exploitation takes place when conditional part of one rule is activated. The balancing of exploration and exploitation is done naturally, and no parameter is involved to control balancing.
present paper is an extension of the Extended Qlearning (EQL) [17] algorithm, where too the authors presumed that their learning algorithm has knowledge about distance measure of the current state to the next state and the goal. They also listed four properties (rules) of EQL without proof, two of which are derived in the current paper. But the remaining two properties presented here are novel. These two new properties provide alternative conditions for locking, thereby improving relative learning speed of the proposed algorithm in comparison to EQL.

The work presented in this paper is better than the classical and the extended Q-learning as described in some papers.

1. In classical Q-learning (CQL), Q-values of the states are updated theoretically for infinite number of steps. For all practical purposes however, the algorithm is terminated when the difference in Q-values of each state in two successive iterations is within a prescribed limit. The algorithm is said to have converged under this circumstance, which too requires excessive computational time. The time-complexity of the proposed Q-learning algorithm has been reduced here by locking selected states, where Q-value update is no longer required. The conditions used for identifying the states to be locked are derived here.

2. The extended Q-learning presented in stores only the best action at a state. Naturally, the knowledge acquired by Q-learning in a world map without obstacles cannot be correctly used for planning, particularly when the next state due to the best action is occupied with an obstacle. In the modified Q-learning presented here, the agent is capable of ranking all the actions at a state based on the Q-values at its neighboring states. Consequently, during the planning cycle, if the state corresponding to the best action is occupied with an obstacle, the robot would pick up the next best action. This in one way overcomes one fundamental limitation of the extended Q-learning.

3. Since the extended Q-learning stores only the best action at a state, it cannot take care of the multiplicity of the best actions. Thus, if there exist two or more best actions, it selects one of them arbitrarily and saves the selected action with the state. Naturally, the stored

action may sometimes involve more turning of the robot during planning than it could have been obtained by an alternative best action. In our algorithm, if we have more than one competitive action at a state during the planning cycle, we select the one ensuring minimum turning of the robot. Thus, our present algorithm is energy optimal.

4.Both the extended and the improved Q-learning, algorithms are terminated when all the states are locked. However, the improved Q-learning has 4 locking conditions, including the two of the extended Q-learning. Because of these two additional conditions of locking, the probability of locking of a state in a given interval of time by the improved Q-learning is higher than the extended Q-learning

CHAPTER-3. TERMINOLOGY

This chapter explains the general words and expressions used in this dissertation. It also explains about the basic Q learning algorithms and how the algorithm is applied to a graph present in a unknown environment. It explains the definition of RL and its goals and the deterministic policy of the RL.

3.1 Reinforcement Learning

Reinforcement Learning [2] is a machine learning technique in which, given an environment and an artificially intelligent software agent, the goal for the agent is to automatically determine the ideal behaviour at each step based on its experience to maximize its performance in the context of that environment. Unlike supervised learning, in case of Reinforcement Learning, the agent learns from the consequences of its actions. It selects the actions either by exploitation or exploration. After every action in each state, the RL-agent receives a reward as shown in (Fig. 2). And based on this reward value, the agent tries to learn a policy that maximizes the overall reward.

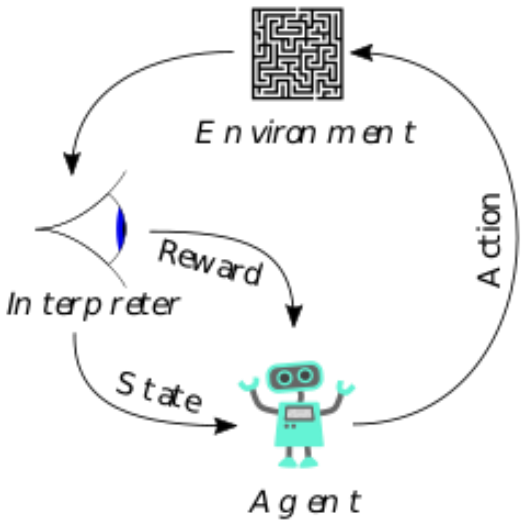

Fig. 2: Reinforcement Learning

This learning technique, which is inspired by behavioural psychology, is usually described as follows. An agent in any environment makes certain movements in this environment and gains rewards as a result of these movements. The main aim is to maximize the total reward and learn the optimal policy for the longest range.

3.2 Q-learning algorithm

In classical Q-learning, all possible states of an agent and its possible action in a given state are deterministically known. In other words, for a given agent A, let S0, S1, S2, ..., Sn, be n possible states, where each state has m possible actions, 0 a, 1 a, 2 a ..., m a. At a particular state-action pair, the specific reward that the agent acquires is known as immediate reward. For example, r (Si, aj) denotes the immediate reward that the agent A acquires by executing an action j a at state Si. An agent selects its next state from its current states by using a policy. This policy attempts to maximize the cumulative reward that the agent could have in subsequent transition of states from its next state. The basic rationale of Q-Learning is to find out how the agent can finish the journey with maximum rewards using experience on the environment. For this, it is aimed that the agent learns the environment in the best way by making different movements on the environment. Q-learning is a value-based Reinforcement Learning algorithm that tries to maximize the q function [2]. The q function is a state-action value function and is denoted by Q (st, at). It tries to maximize the expected reward give a state and action performed on that state:

$$\mathbf{Q(st, at)} = \mathbf{E[Gt|st, at]}$$

The tabular Q-learning approach works well for small environments, but becomes infeasible for complex environments with large multidimensional discrete or continuous state-action spaces. To deal with this problem, a parameterized version of the q function is used for approximation called the Deep Q-Network. It has become the predominant method for approximating complex networks [11]. Q-learning and DQN are capable of achieving performance beyond the human level on many occasions. However, in some cases Q-learning performs poorly and so does its deep neural network counterpart DQN. The main reason behind such poor performance is that Q-learning tends to overestimate action values. These overestimations are caused due to a positive bias that results from the max function in Q-learning and DQN updates which outputs the maximum action value as an approximation of the maximum expected action value. The Double Q-learning method was proposed in [5] to alleviate this problem and later extended to DQN [3] to produce the Double DQN (DDQN) method. Since Q-learning uses the same estimator to select and evaluate an action, which results in overoptimistic action values, we can interpret it as a

single estimator. In Double Q-learning, the task of evaluation and selection is decoupled by using double estimator approach consisting of two functions: Qa and Qb. In newly proposed Improved Q-learning, best action at each state is selected dynamically. This help in reducing the energy required by the agent while moving. In the planning algorithm the best action is selected by comparing the Q-value of the 4-neighbour states.

87 (q1)	**86** (q2)	**85** (q3)
88 (q5)	**O**	**86** (q4)
89 (q6)	**88** (q7)	**87** (q8)

Table 1: Portion of environment with Q-value

If there are more than one state with equal Q-value than the action that requires minimum energy for turning is selected. Next feasible states are the 4-neighbour states and corresponding Q value are given in each state as shown in (Table 1). There are two states with maximum Q-value of 88[10]. If the next state is located at south then the robot has to rotate toward right by 90 degree and if the next state is located at west then the robot has to rotate by 180 degree. Therefore, the presumed next state will be the state at south as the robot requires less energy to move to that state. During path planning the robot selects one out of several best actions at a given state. If an obstacle is present in the selected direction then next optimal path is selected. If there is no action to be selected then a signal is passed to the robot to backtrack one step.

3.3 Deep Reinforcement learning with DDQN algorithm

The conventional Q-learning algorithm [1] cannot effectively plan a path in random dynamic environment because of the lack of generalization ability and a large Q table. To solve the problem of the curse of dimensionality in high dimensional state space, the

optimal action value function Q in Q-learning can be parameterized by an approximate value function.

$$\boldsymbol{Q(s,a;\theta) \approx Q*(s,a)}$$

where θ is the Q-network parameter. We approximate the value of Q in a definite environment state by function equation, which is mainly linear approximation; thus, there is no need to build a large Q table to determine the corresponding Q values for different state-actions. Neural network is used to approximate the linear function which can obtain nonlinear approximation with generalization ability. (s,a) is regarded as the input of the neural network and the output is the value of Q, where $\boldsymbol{\theta}$ is the weight of Q-network. Q table is replaced by Q-network. The training process is to constantly adjust the network weights to reduce the bias of the output of Q-network and the target value of Q. Assuming that the target value of Q is denoted by y, thus the loss function of Q-network is yielded

$$\boldsymbol{Li(\theta i) = Es,a[(yi - Q(s,a;\theta i) * 2)]}$$

where iteration i is the current iteration times and () is a pair of state-action.

The update mode of the value of Q is similar to that of Q-learning; that is,

$$\boldsymbol{yi = Es'\sim s[r + \gamma maxa'Q(s',a';\theta i - 1)|s,a]}$$

where r is the current reward and γ is a discount factor.

Stochastic gradient descent algorithm is adopted to train the neural network. According to the back propagation of the derivative value of loss function, the network weights are constantly adjusted, resulting in the network output approaching the target value of Q.

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action value (also known as Q) function. This instability has several causes: the correlations present in the sequence of observations and the correlations between the action values (Q) and the target values. To address these instabilities, DeepMind presents a biologically inspired mechanism termed experience replay that randomizes over the data. To perform experience replay, DeepMind stores the agent's experiences, environment state, action, and reward at each time-step t in a dataset. The pool of data samples is extended with running e-greedy

policy to search the environment. During the learning, train the network with random data from the pool of stored samples, thereby removing correlations in the observation sequence and improving the stability of algorithm. In 2015, DeepMind improved the original algorithm in the paper "Human-Level Control through Deep Reinforcement Learning" published in Nature. The framework illustration of DDQN is shown in Figure 3 below as proposed by DeepMind.

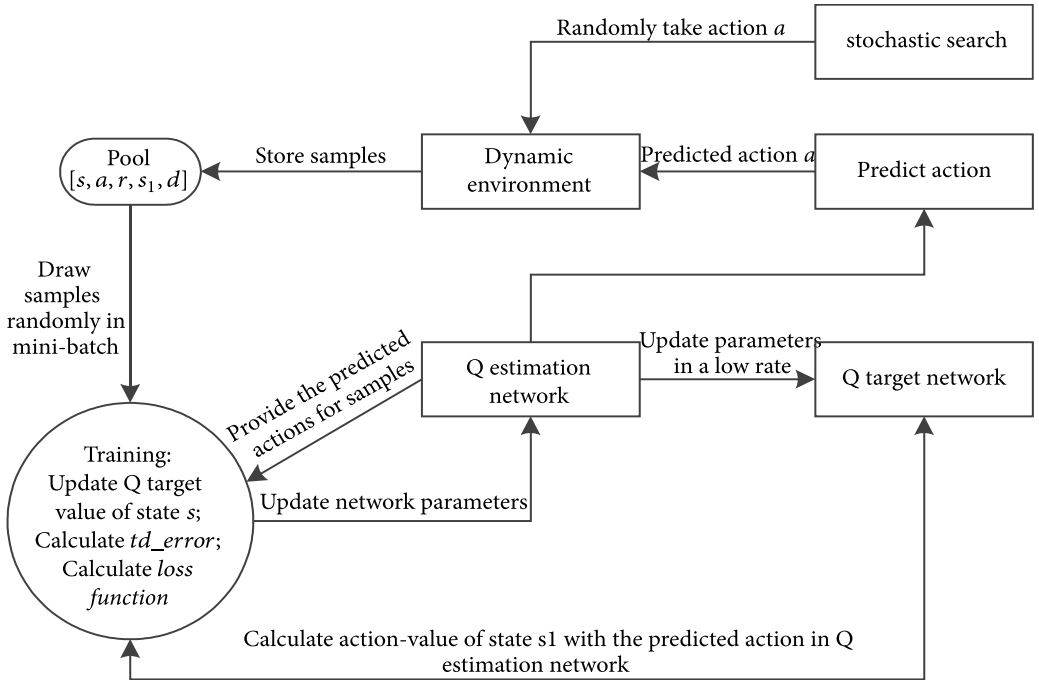

Fig. 3: Framework illustration of DDQN

They added a target Q-network, which is delayed updating in comparison with the predicted Q-network, to update the target values of Q, thereby restraining a big bias of Q resulting from a fast dynamic updating of the target Q-network. The update mode of Q-learning algorithm results in a problem of overestimate action values. The algorithm estimates the value of a certain state too optimistically, consequently causing that the Q value of subprime action is greater than that of the optimal action, thereby changing the optimal action and affecting the accuracy of the algorithm. To address the overestimate problem, in 2016, DeepMind presented an improved algorithm in the paper "Deep Reinforcement Learning with Double Q-Learning" as shown in (Fig. 3) with framework of the new model namely, Double Q-network algorithm, which selected an action by maximizing the value of the predicted Q-network and updating the predicted Q-network

with the selected action value in Q-network instead of directly updating the predicted Q-network with the maximum value of target Q-network.

3.4 Local Path Planning with DDQN Algorithm

In this paper, we achieve local path planning with deep reinforcement learning DDQN. Lidar is used to detect the environment information over 360 degrees. The sensor range is regarded as observation window. The accessible point nearest to the global path at the edge of the observation window is considered as the local target point of the local path planning. The network receives lidar dot matrix information and local target point coordinates as the inputs and outputs the direction of movement.

Considering the computation burden and actual navigation effect, i set angle resolution to be 1 degree and range limit to be 2 meters so each observation consists of 360 points indicating the distance to obstacles within a two-meter circle around the robot. The local target point is the intersection point of the observation window circular arc and the global path. If there were several intersection points, the optimal point would be chosen by heuristic evaluation. As to the lidar dot matrix information, the angle and distance can be denoted by [angle, distance] . i set a rule that the angles of the lidar points increases clockwise relative to the front of the mobile robot. 360 points result in 720 inputs of data. The relative coordinate of local target point and the current center coordinate of the mobile robot body $(\Delta x, \Delta y)$ are also regarded as the input. To achieve the network output weights of the relative target points in training and conveniently design the convolutional network, we make 40 copies of the relative target points as the inputs; namely, the total inputs are 800 datasets.

The outputs are fixed-length omnidirectional movements of eight directions, where the movement length is 10 cm, and the directions are forward, backward, left, right, left front, left rear, right front, and right rear, which are denoted by 1 to 8 in order as follows.

$$a = [a1, a2, a3, a4, a5, a6, a7, a8]$$

Each lidar detected point receives continuous measurement values changing from 0 cm to 200 cm, which means that the state space tends towards infinitude and it is impossible to represent all of the states with Q table. With DDQN, the generalization ability of neural

network makes it possible that the network is able to approach all states after training; therefore, when the environment changed, the agent could plan a proper path and arrive at the target position according to the weights of the network as described in the pseudo code of the path planning below.

Pseudo Code for Path Planning

BEGIN

- CURRENT= starting – state;
- Get next state NEXT, where Q – value of the NEXT >= Q-value in the neighbourhood of CURRENT and NEXT is Obstacle free;
- Go to NEXT;
- CURRENT = NEXT
- Repeat from step 2 until NEXT = GOAL;

END

To ensure the deep reinforcement learning training converging normally, the pool should be large enough to store state-action of each time-step and keep the training samples of neural network be independent identical distribution; besides, the environment punishment and reward should reach to a certain proportion. If the sample space was too sparse, namely, the main states were random movements in free space, it was difficult to achieve a stable training effect. As to the instability of DDQN in training and the reward sparsity of the state space, the starting point is randomly set in the circle with the target point as the center and the radius L. The initial value of L is small, thereby increasing the probability that the agent reaches the target point from the starting point in the random exploration and ensuring a positive incentive in the sample space. The value of L gradually increases with the update of neural network and the increase of greedy law probability. The termination of each episode is to achieve a fixed number of moving steps instead of directly terminating the current training episode when encountering obstacles or reaching the target point. The original training mode results in a large number of samples which are moving states in the free zone in the pool, and the lack of trial-and-error punishment samples and target reward samples ultimately leads to algorithm disconvergence.

3.5 The Neural Network in DDQN

Neural network is a primary method for reinforcement learning to enhance the ability of generalization. The optimization design of the neural network architecture is able to reduce overfitting and improve the prediction accuracy with high training speed and low computational cost. Considering 800 inputs, if we directly adopted fully connected layers, the number of parameters needed to be trained would increase exponentially with the increase of the layers, resulting in a heavy computation burden and overfitting. Convolutional neural networks (CNNs) have made breakthrough progress in the field of image recognition in recent years. CNNs are featured with a unique network processing mode which is characteristic of local connection, pooling, sharing the weights, etc. The unique mode effectively reduces the computational cost, computational complexity, and the number of the trained parameters. With the method of CNNs, the image model in the translation, scaling, distortion, and other transformations is able to be invariant to a certain degree, thereby improving the robustness of system to failures. Based on these superior features, CNNs surpass fully connected neural network in information processing task where the data are relevant to each other. Generally, the CNNs consist of input layer, convolutional layer, pooling layer, fully connected layer, and output layer as shown in the below (Fig. 4). convolutional layer adopts local connection; namely, each node is connected with some nodes of the upper layer.

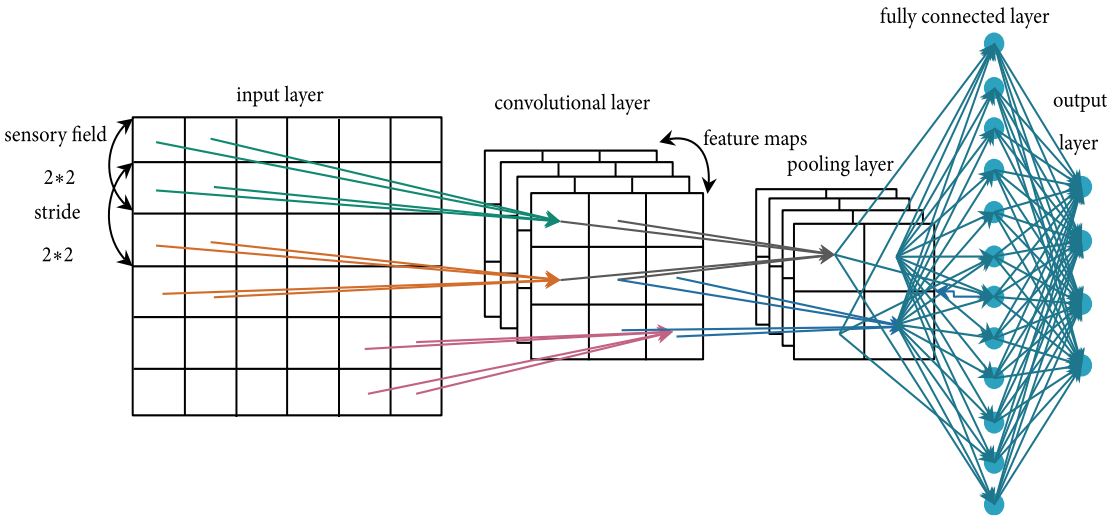

Fig. 4: The architecture of CNN in DDQN algorithm

Four adjacent pixels of input layer form a 2-by-2 local sensory field which is connected with a neuron in convolutional layer. Each local sensory field represents a local feature, and the size of the field can be adjusted artificially. The next layer of convolutional layer is pooling layer, which can also reduce the matrix size. The spatial invariant feature is obtained by reducing the resolution of the feature surface. The number of feature maps in the pooling layer is equal to the number of convolutional layers, and each neuron performs a pooling operation on the local sensory field. Pooling is suitable for separating the sparse features.

A fully connected layer is added after one or more convolutional layers and pooling layers. The fully connected layer integrates the partial information involved in category distinguishing in convolutional layers or pooling layers. Besides, rectified linear unit (ReLU) is a recent used activation functions in CNNs, which can improve the performance of a CNN. The output value of the fully connected layer is directly transferred to the output layer. For deep reinforcement learning, the output layer is the action space.

The lidar data between adjacent points reflects the distribution of obstacles and the width of the free zone. The convolutional network can effectively extract the information of these characteristics and reduce the network parameters greatly. Therefore, the CNNs are used to train the path planning of local dynamic environment. In this paper, the architecture of Deep Q-network consists of three convolutional layers and a fully connected layer. The input layer is a three-dimensional matrix with the size of 20×20×2 formed by a vector with 800 elements, where the data in the third dimension represents the angle and distance of a lidar point. The size of the input layer is 20×20×2. According to the size of the input layer, we design the first convolutional layer, of which the size of the receptive field is 2-by-2, the stride is 2-by-2, and the number of feature maps is set to be 16. Thus, the size of the output layer is 10×10×16. The kernel size of the second convolutional layer is 2-by-2; the stride is 2-by-2; and the number of the characteristic plane is 32. The size of the output of this convolutional layer is 5×5×32. The kernel size of the third convolutional layer is 5-by-5; the stride is 1-by-1; and the number of the characteristic plane is 128. The size of the output of the third convolutional layer is 1×1×128. Then a three-dimensional structure is transformed to a one-dimensional vector with 128 elements, which are connected to the fully connected layer, of which the size is 128-by-256. The size of the output layer is the number of the actions, namely, 8. We use ReLU activation function and Adam optimizer.

CHAPTER-4. IMPLEMENTATION

In this chapter, I will be discussing about how I implemented the project using the python open source machine learning framework like (Pygame and TensorFlow) to build a DDQN training network. This chapter also discusses about the performance comparison on the grid-world environments.

4.1 Setting Up the Dynamic Environment

In this project i used the open source machine learning framework TensorFlow provided by python to build the DDQN training framework. Pygame module is to build dynamic environment. As shown in Figure 5, the white area denotes obstacles, with the size of 20-by-20 cm, are moving in a constant speed, and the corresponding blue arrow represents its moving range and directions. The green point denotes a local target position. The gray square object represents the mobile robot (agent). During the training, the local path planning is trained in the lower-level and intermediate environment. The starting point and target point are randomly generated to ensure diversity and complexity of local environment, and the test environment is a high-level dynamic map. Two CNNs of the same architecture are established, which are Q estimation network and Q target network. The network parameters are randomly initialized with normal distribution, of which the mean value is 0.

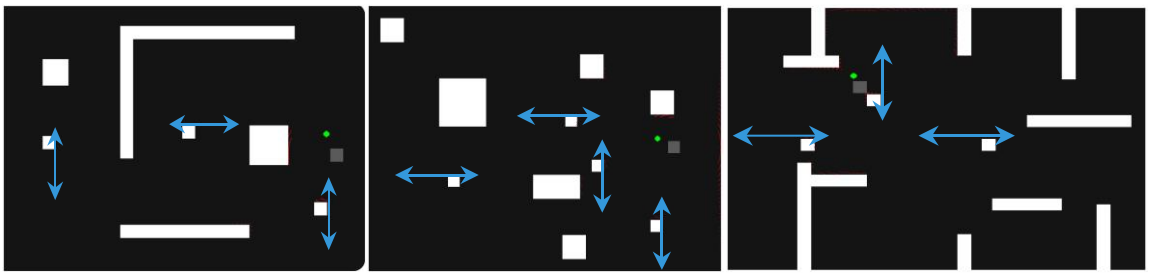

Fig. 5: Dynamic Environments

The training policy is a variable random greedy rule. At the beginning of the training, the pool of experience is updated by the method of random exploitation because of the lack of environmental information. The sample consists of five components, namely,[s,a,r,s1,d] ,where s denotes the current state; a denotes the adopted action; s1 is the state after taking action; r is the reward value obtained for the current state

transition; d is a flag representing that whether the current iteration epoch ends. We set the size of the pool to 40000 samples. If the samples stored in the pool reached a certain number, the network was to be trained with the randomly selected samples in the pool. In the first 5000 time-steps of the random exploration, the network parameters are not updated, but the samples of the pool are increased. After the sample size reaches to 5000, the network is trained in every four time-step movements. We randomly draw 32 samples in sequence by means of minibatch to update the Q estimation network and update the Q target network with a low learning rate to make the parameters of Q target network approach the parameters of Q estimation network, thereby ensuring a stable learning of Q target network. If the sample size of the pool was up to the upper limit, the earliest sample would be excluded from the pool in the policy of first-in first-out after a new sample is added, thereby ensuring a continuous updating of the pool.

According to the actual experimental effect, if the position of agent and the target point were completely randomly set at first, there would be a large probability that the distance between the agent and target point was too large, resulting in that the agent was unable to reach the target point in fixed steps with random exploration. Consequently, to ensure the target point being detected by the agent, we set the initial distance between the agent and target point randomly in a range of L and gradually increase L with exploration and training. This process is also a process of constantly changing the environmental states which leads to a more extensive distribution of samples.

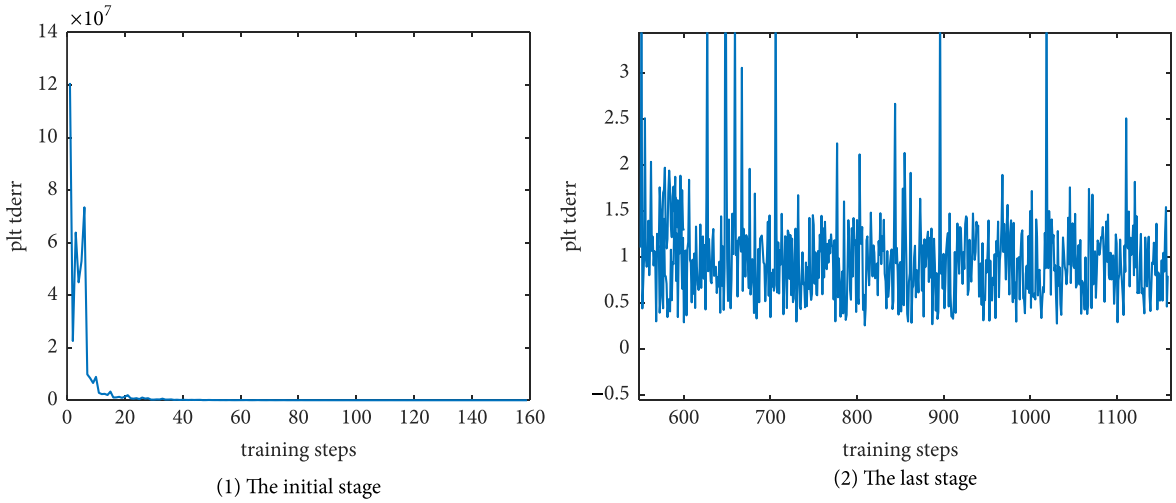

Fig. 6: Training curves of loss function.

During the training process, the values of loss function of Q estimation network and Q target network decrease continuously. Figure 6 shows the training curves of the loss function of Q target network where each point is the average loss function value achieved per ten epochs. The y-axis denotes the value of loss function and x-axis denotes iteration epoch. Each point is the average loss function value achieved per ten epochs. Figure 6(1) indicates that, in initial stage of the training, the magnitude of loss function value is 10^7. while, as we got from our data, after training for 1000 epochs, actually, the value dropped to about 10,000 (seen from the figure, the curve tends to be close to x-axis); after 7000 epochs, as we can see from Figure 6 (2) that it dropped to less than 1. With the whole process completed, the loss function value is less than 0.25, and the Q-network converges successfully. The cumulative reward of each iteration is gradually increased; that is, in the process of exploration, the agent gradually learns to take an action which can increase the reward. Figure 7 shows the average cumulative reward curves. Each point is the average cumulative reward achieved per hundred epochs. The figure indicates that the average reward

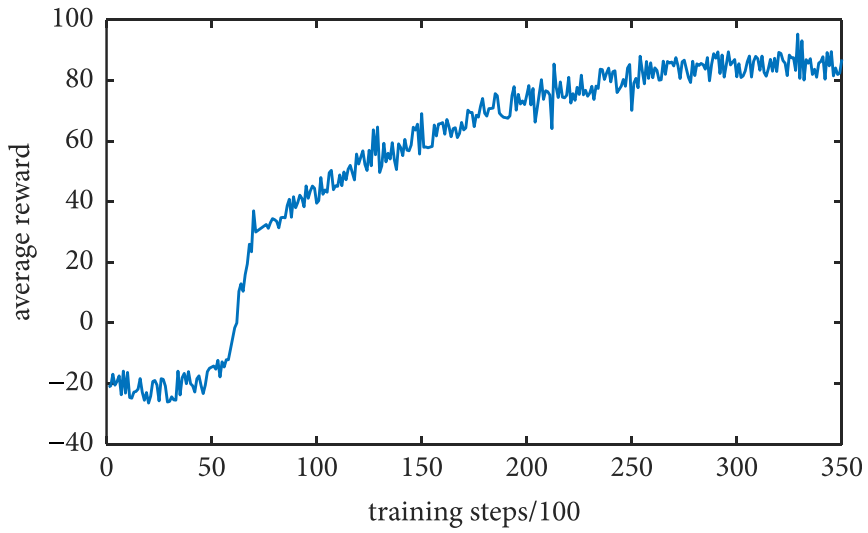

Fig. 7: The average cumulative reward curve.

gradually increases with the increase of training epochs. After training for 30000 epochs, the reward value tends to be stable and average cumulative reward is greater than 80. We set the fixed step times to be 100, which means the agent can avoid obstacles in a dynamic environment and quickly reach the target point to obtain a continuous reward until the current training epoch ends. After training for 40000 epochs, Q estimation network and Q target network converge. We store the network parameters and test in an experimental environment map as discussed in the below results section.

4.2 Results

The Results I obtained in this thesis are as the following. Assuming a global path, we set several local target points in the lidar searching area regardless of the positions of the dynamic and static obstacles to test the agent's local path planning ability. Figure 8 is a new environmental map that I used in the making of my thesis using python module. This map is used to evaluate the generalization ability and the path planning ability of DDQN in a new unknown dynamic environment. There are three free moving obstacles with constant speed and the moving directions are denoted by the blue arrows. In the figure below, state A demonstrates that the local path is blocked by dynamic obstacle No. 2; then, the agent waits for the obstacle to move downwards.

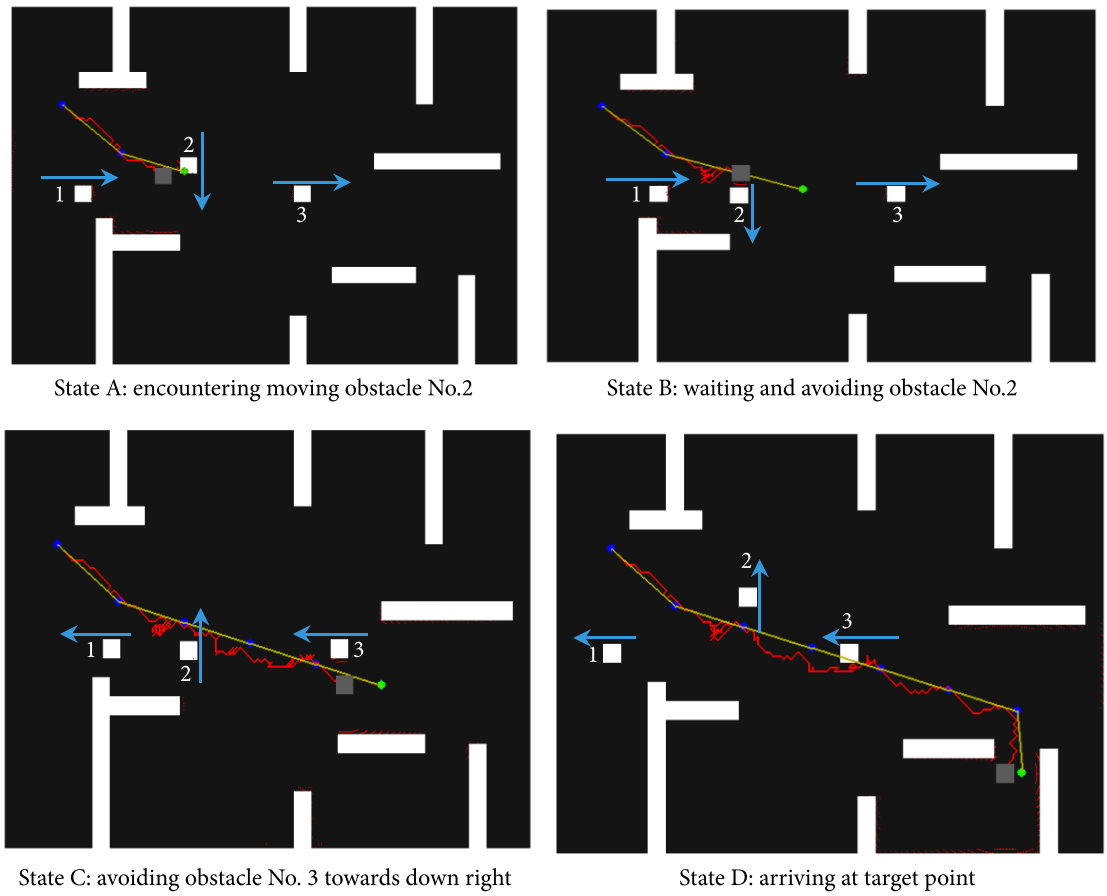

Fig. 8: Local path planning in a test map.

When the obstacle is out of the path, the agent moves towards upper right. As shown in state B, the agent successfully reaches the third local target. In state C, obstacle No. 3 moves towards the agent. The agent perceives obstacle No. 3 before collision and gets to the sixth local target point by moving toward the bottom right. In state D, the agent reaches the end point and completes the

path planning without any accident collision with the obstacles. The whole process demonstrates that the agent after being trained by DDQN is able to perceive the moving obstacles in advance with the knowledge of lidar data in unknown dynamic environment. An intelligent planning method is presented by Q target network which makes each step move towards a higher cumulative reward. The local path planning is completed by the trained DDQN algorithm. The inputs are angles, distance detected by lidar, and local planning points. The output is moving direction. And, meanwhile, the trajectory is smoothed. During the movement, the agent builds map and navigates in real-time. The local path planning effect of the algorithm and generalization ability of the convolutional network are tested in unknown environment. The agent followed the global path but modified its path and successfully avoided the obstacles when it encountered obstacles. A new environmental map was gradually built in the process of moving.

CHAPTER-5. CONCLUSION AND FUTURE SCOPE

In this chapter, conclusion is described where I summarized the main points of evidence for the reader and it also discussed about the future possibilities that can be obtained if we have a research planning with specific project goal, ideas, costs and deadlines about the dissertation.

5.1 Conclusion

This paper proposed an alternative algorithm for deterministic Deep Q-learning, presuming that the background knowledge about the distance from the current state to both the next state and the goal state are available. The proposed algorithm updates the entries of the Q-table only once unlike the classical Q-learning, where the entries in the Q-table were updates many times until the convergence was ensured. It also helps us approach a way to reduce both time and space complexity of the classical Q-learning algorithm. The algorithm is coded in Python and is tested in python platform. I used a tabular Q learning to generate Q values for shortest path to the exit using the adjacency matrix of the graph-based environment. We make the agent learn the environment with the proposed new algorithm and the original classical Q-learning algorithm for verifying the applicability of the proposed algorithm. We stop the classical Q-learning after 50 iterations. For the new algorithm we allow to learn till all the Lock variables are set. Total numbers of iteration vary from 100 to nearly 250 as the position of the goal changes. Learning is done without any obstacle. After completion of learning we have kept the agent, facing left, in different environments and allow traversing to the goal using the stored Q-table by both the algorithm. Then the result of Deep Q learning and Q table values are used to pretrain the network to incorporate shortest path information in the agent. Deep reinforcement learning solves the problem of curse of dimensionality well and is able to process multidimensional inputs. In this paper, we design a specific training method and reward function, which effectively addresses the problem of disconvergence of the algorithm due to the reward sparsity of state space. The experimental results show that DDQN algorithm is capable and flexible for local path planning in unknown dynamic environment with the knowledge of lidar data. Compared with visual navigation tasks, the local lidar information, which can

be transplanted into more complex environments without retraining the network parameters, has wider applications and better generalization performance.

5.2 Future Scope

Deep Q-learning has proven its worth in approximating real-world environments. It can help solve many problems that, to date, have remained challenging for machines to tackle. With the combination of deep learning and RL, we're much closer to solving these problems. In this dissertation, I will be considering the action space to be limited to only four allowed actions. However, in the future, it would be interesting to look at the problems with larger state and action space. It would be interesting to see how the agent behaves using Deep Q-learning when there is a higher density of number of obstacles in the path towards the goal state. Applications of this algorithm on a complex real-world problem can be more meaningful as opposed to the computer simulated grid-based problems evaluated in this dissertation. In future it can also be possible for Deep Q-learning algorithm to handle lots of attribute and handle the tricky classification present in the system.

LIST OF REFERENCES

[1] Jihee Han, “A surrounding point set approach for path planning in unknown environments,” *Computers & Industrial Engineering*, 2019.

[2] Jiexin Xie, Zhenzhou Shao, Yue Li, Yong Guan, and Jindong Tan, “Deep Reinforcement Learning With Optimized Reward Functions,” *Robotic Trajectory Planning, IEEE Access*, vol. 7, pp. 105669–105679, 2019

[3] B. Tozer, T. Mazzuchi, and S. Sarkani, “Many-objective stochastic path finding using reinforcement learning,” *Expert Systems with Applications, vol*. 72, no. *Supplement C*, pp. 371 – 382,2017.

[4] H.S Behera “An Improved Q learning algorithm for path planning of a mobile robot” *International Journal of Computer Applications (0975 – 8887)* Volume 51– No.9, August 2012

[5] Thombre, Prashant, "Multi-objective Path Finding Using Reinforcement Learning" *MORL Voting policy* (2018). Master’s Projects.643.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[7] The Reinforcement Learning Repository University of Massachusetts, Amherst

[8] Tesauro, Gerald, Temporal Difference Learning and TD-Gammon, *Communications of the Association for Computing Machinery*, March 1995 / Vol 38, No. 3

[9] Perez, Andres, Reinforcement Learning and Autonomous Robots - collection of links to tutorials, books and applications

[10] P. Mirowski, *Learning to navigate in complex environments*, 2016.

[11] S. Legg and M. Hutter, “Universal intelligence: A definition of machine intelligence,” *Minds and Machines*, vol. 17, no. 4, pp. 391–444, 2007.

[12] V. Mnih, K. Kavukcuoglu, D. Silver et al., “Playing atari with deep reinforcement learning,” https://arxiv.org/abs/1312.5602, 2013.

[13] V. Mnih, K. Kavukcuoglu, D. Silver et al., “Human-level control through deep reinforcement learning,” *Nature, vol. 518, no. 7540, pp. 529–533, 2015*.

[14] Emrah Mete “Optimal Path detection with Reinforcement learning” Jun 2018,

[15] Alind gupta ,Noida “Python implementation using Deep Q learning algorithm”

Link to source “https://www.geeksforgeeks.org/ml-reinforcement-learning-algorithm-python-implementation-using-q-learning/

[16] Q. Zhang, “Convolutional Neural Networks,” in Proceedings of the 3rd International Conference on *Electromechanical Control Technology and Transportation*, pp. 434–439, Chongqing, China, January 2018.

[17] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, “Deep convolutional neural networks for LVCSR,” in Proceedings of the 38th IEEE International Conference on *Acoustics, Speech, and Signal Processing (ICASSP '13)*, pp. 8614–8618, IEEE, Vancouver, Canada, May 2013.

[18] Willow Garage, *Robot Operating System*, Willow Garage, 2015,

[19] M. Cashmore, M. Fox, D. Long et al., “Rosplan: Planning in the robot operating system,” in Proceedings of the 25th International Conference on *Automated Planning and Scheduling*, (ICAPS '15), pp. 333–341, June 2015.

[20] M. Quigley, K. Conley, B. Gerkey et al., “ROS: an open-source Robot Operating System,” vol. 3, no. 3.2, ICRA workshop on *open source software*, May 2009.

[21] L. J. Lin, “Reinforcement learning for robots using neural networks,” *Tech. Rep. CMU*-CS-93-103, Carnegie-Mellon University Pittsburgh PA School of Computer Science, 1993.

[22] J. O'Neill, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari, “Play it again: reactivation of waking experience and memory,” *Trends in Neurosciences*, vol. 33, no. 5, pp. 220–229, 2010

[23] Z. Tang, K. Shao, D. Zhao, and Y. Zhu, “Recent progress of deep reinforcement learning: from AlphaGo to AlphaGo Zero*,” Control Theory & Applications*, vol. 34, no. 12, pp. 1529–1546, 2017.