AUTONOMOUS ROBOT NAVIGATION USING DEEP Q LEARNING

A Project Report

Submitted by

Mr. SAJIN S

REG NO : TKM20MEAI12

SEMESTER : IV

In partial fulfillment for the award of the degree of

MASTER OF TECHNOLOGY
IN
Mechanical Engineering (Artificial Intelligence)

Under the guidance of
Dr. IMTHIAS AHAMED T P

Thangal Kunju Musaliar College of Engineering
Kerala

JULY 2022

DECLARATION

I undersigned hereby declare that the project report "AUTONOMOUS ROBOT NAVIGATION USING DEEP Q LEARNING", submitted for partial fulfillment of the requirements for the award of degree of Master of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under supervision of Dr. Imthias Ahamed T P. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Kollam
Date: SAJIN S

Thangal Kunju Musaliar College of Engineering
Centre for Artificial Intelligence

C E R T I F I C A T E

This is to certify that, this report titled **ROBOT NAVIGATION USING DEEP Q LEARNING** is a bonafide record of the **Project** presented by **SAJIN S (TKM20MEAI12)**, under our guidance and supervision, in partial fulfillment of the requirements for the award of the degree, **M.Tech in Mechanical Engineering (Artificial Intelligence)** in **APJ Abdul Kalam Technological University** .

Project coordinator

Prof. Sumod Sundar
Assistant Professor
Centre for Artificial Intelligence

Head of the Department

Dr. Imthias Ahamed T P
Professor & HOD
Centre for Artificial Intelligence

Internal Supervisor

Dr. Imthias Ahamed T P
Professor & HOD
Centre for Artificial Intelligence

Internal examiner

External examiner

ACKNOWLEDGEMENT

A successful project is a fruitful culmination of efforts by many people, some directly involved and some others indirectly, by providing support and encouragement. Firstly I would like to thank the almighty for giving me the wisdom and grace for making my project a successful one. I thank him for steering me to the shore of fulfillment under his protective wings.

I would like to express my heartfelt thanks to our project coordinator cum internal supervisor **Prof. Sumod Sundar**, Assistant Professor, Centre for Artificial Intelligence, TKMCE, for his constant support and encouragement throughout the project work. I would like to express my heartfelt thanks to our project coordinator cum my internal supervisor **Dr. Imthias Ahamed T P**, Professor and Head of the Department, Centre for Artificial Intelligence, TKMCE, for his expert guidance and cooperation. With a profound sense of gratitude, I would like to thank **Dr. Santhi Natarajan**, Honorary Professor, for her immense encouragement. I would like to express my gratitude to **Syed Muhammed Fahd**, Asst. Professor, Department of Mechanical Engineering, TKMCE and **Mr. Nijesh P S**, Project Manager, Tata Elxsi, for their expert guidance, and cooperation. I also extend my thanks to the entire faculty and staff members of the Centre for AI, TKMCE, who has encouraged me throughout this work.

I also express my thanks to my loving parents, brother and friends, for their support and encouragement in the successful completion of this project work.

SAJIN S

Abstract

Artificial Intelligence has been a topic of great interest in a wide range of fields, such as gaming, computer vision, and robotics. One of the impressive applications is the AI-based Alpha Go that has superhuman performance in the games of Go, with the help of Deep Reinforcement Learning based techniques. Majority of the research in robotics is concerned with mobile robots, especially mobile robot navigation using Deep Reinforcement Learning. Our interests focus on investigating whether a Deep Q Learning-based solution is beneficial to improve the robot behaviors when solving constrained autonomous navigation problems.

This work, focus on implementing a robust, point to point navigation module (local path planner, decision making and control modules) using Deep Q Learning for autonomous navigation of indoor robot. And this path planner is able to navigate robot autonomously through cluttered, unstructured environments satisfying constrains like the robot should not collide with any static or dynamic obstacles, it should keep a safe distance from all objects while moving, the robot should traverse on the left-hand side of the corridor and should navigate through shortest path.

The frameworks used in this work are ROS, OpenAI Gym and Gazebo Simulator. Training neural network architecture in Deep Q Learning requires a large data set which is obtained by stimulating robot on Gazebo environment. And this data set is used by Deep Q Learning algorithm (which is written as a node in ROS) for training.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Deep reinforcement learning is one of the most interesting field of machine learning that combines reinforcement learning with deep learning. The strong human like learning ability of deep reinforcement learning is widely used for a diverse set of application like robotics, video games, machine vision, transportation and health care. Robots are being developed in a rapid pace to assist humans and the quality of automation in robot has to be as good as humans. The automated robots should replicate human actions. Humans are very good at their navigation skills, that is when coming to unknown complex environment humans use their prior knowledge for smooth and safe navigation. The need for autonomous navigation of robots comes into picture.

The robot with autonomous navigation capability is able to find a path in real time from a starting point to an end point without collision. For indoor environments, traditional navigation techniques uses simultaneous localization and mapping (SLAM) to map unknown environments and then use localization techniques like AMCL (Adaptive Monte Carlo Localization) to move robot to the desired location.

This work, focus on implementing a robust, point to point navigation module (local path planner, decision making and control modules) using Deep Q Learning for autonomous navigation of indoor robot. And this path planner is able to navigate robot autonomously through cluttered, unstructured environments satisfying constrains like the robot should not collide with any static obstacles or walls, it should keep a safe distance from all objects while moving, the robot should traverse on the left-hand side of the corridor and should navigate through shortest path following a smooth trajectory. All the process of training and evaluation is done in the ROS-gazebo framework integrated with OpenAI Gym.

Major contributions are summarized as follows:

- The robot is able to navigate autonomously through unknown environment satisfying all the constraints mentioned above.
- New reward functions are designed in order to increase the convergence rate of the DQN algorithm.
- Three different strategies are carried out in order to investigate generalization of algorithm to its application.

The frameworks used in this work are ROS, OpenAI Gym and Gazebo Simulator.
The remainder of this report is organized into the following chapters: Chapter 2 reviews

different deep reinforcement learning techniques for robot path planning.Chapter 3 includes problem statement,high-level architecture and detailed methodology. The experimental results are summarised in Chapter 4.

Chapter 2

RELATED WORKS

In this section, several studies of Robot Path Planing utilizing both traditional as well as Reinforcement based techniques are discussed.

Valentyn et al. [1] implemented two Reinforcement Learning algorithms, Q-Learning and Sarsa for global path planning for mobile robots. In this work the Reinforcement Learning algorithm showed difference in learning time.It is found that Q-learning algorithm showed faster convergence than Sarsa.However ,the Sarsa algorithm provides safer path for mobile robot.The main drawback is that both Q-Learning and Sarsa algorithms are used for solving problems with discrete states and discrete actions.In most of the cases Path Planning should be formulated as a continuous state MDP.

Ee Soong Low et al. [2] found slow convergence rate of Q-Learning algorithm and proposed an improved Q-learning to solve path planning of a mobile robot.Here flower pollination algorithm(FPA) is used to initialize Q-table prior to Q-Learning implementation. The performance of the proposed algorithm is compared with classical Q-Learning and improvement in computational time is observed. Here in this work only static environment is considered.

Thi Thoa Mac et al. [3] conducted different experiments in order to find strengths and drawbacks of each heuristic-based algorithms in robot path planning.Work comprised of neural network, fuzzy logic, nature-inspired algorithms and hybrid algorithms.The author also considered potential field methods knowing the ability of the approach in solving path planning problem.The integrated approach based on neural and fuzzy provide much better results than individual techniques since they archive both advantage of similarity to human thinking(fuzzy) and the ability to learn(neural network).

Jiexin et al. [4] developed two dense reward functions, including azimuth reward function and subtask-level reward function that could improve the efficiency of Deep Reinforcement Learning for robot trajectory planning in unstructured environment with obstacles.The experiment conducted by author shows that the proposed reward is able to improve convergence rate of the algorithm by three times when compared with other DRL methods.More studies are needed in order to extend this method to multi-objective trajectory planning task and nowhere in this work the author talks about the ability of algorithm generalizing to any given environments.

Jeevan Raajan et al. [5] proposed a real time path planner for mobile robot using Deep Reinforcement Learning.The aim of the work is to find a path in real time from a given initial position to the goal position avoiding all the obstacles, both static and dynamic.Deep Q Learning is implemented with dueling architecture and Prioritized experience replay.Numerical experiments are performed for the proposed method(Deep Q Learning algorithm) and results are compared with the state of the art sampling based path planning algorithms.It is found that the algorithm is significantly faster compared to the traditional methods of path planning.The main drawback of this work are,the environment considered here is assumed to be completely known and the proposed algorithm is able to provides only a sub-optimal path.

H Li et al. [6] constructed a general exploration framework for mobile robots by decomposing the exploration process into three independent modules namely decision, planning, and mapping, which increased the modularity of the robotic system. The exploration strategies were learned from a deep reinforcement learning algorithm using a deep neural network. The decision making algorithm takes partial map of the environment as input. It also combines the traditional navigation algorithms like A* and hence has faster learning and convergence. But it is difficult to apply this technique where a map is not readily available.

Wu C et al. [7] proves that deep reinforcement learning can be successfully applied to an ancient puzzle game Nokia Snake.For playing snake game the author proposed an algorithm called Snake algorithm and the backbone of the snake algorithm is Deep Q Learning.Input to the DQN is pre processed frames of the game.Most of the work is done in this pre processing stage where current frames are stacked with previous three frames and is converted in to gray scale and finally resized. And after, training the snake is able to track its target.In shot the Snake algorithm proposed here was able to find the target path autonomously.Working of the Snake algorithm in real world scenarios are sill unknown.

J Choi et al. [8]proposed a method in which a mobile robot was able to avoid both static and dynamic obstacles and can drive to the target point based on reinforcement learning. A map of the environment was initially created for better localization of the mobile robot. A path planning algorithm was integrated to enhance the target reach-ability. For effective dynamic obstacle avoidance laser data from past two instances were also considered. To reduce the difference between the real driving environment and the training environment, they developed a training environment where dynamic characteristics were considered and implemented using soft actor critic as the reinforcement learning algorithm to learn policies.

Chapter 3

METHODOLOGY

This chapter describes the detailed methods and techniques used in training robot using Reinforcement Learning frame work.

3.1 Problem Statement

To implement a robust, point to point navigation module (local path planner, decision making and control modules) using Reinforcement Learning (RL) for autonomous navigation of indoor robot. The path planner shall be able to generalize and navigate robot autonomously through cluttered, unstructured environment satisfying the following constraints:

- The robot shall navigate autonomously from Point A to Point B in shortest path.
- The robot shall not collide with any objects or walls
- The robot shall keep a safe distance from all objects while moving.
- The robot shall follow smooth trajectory.
- The robot shall respond to change in object locations.
- The robot should try to align with the given path.

3.2 High-level Software Architecture for RL in robotics

The architecture is build with three software blocks: OpenAI Gym, ROS and Gazebo. Environment developed in OpenAI Gym interact with Robot Operating System, which is the connection between the Gym itself and Gazebo simulator. This is possible because both Gym and Gazebo is integrated with ROS.

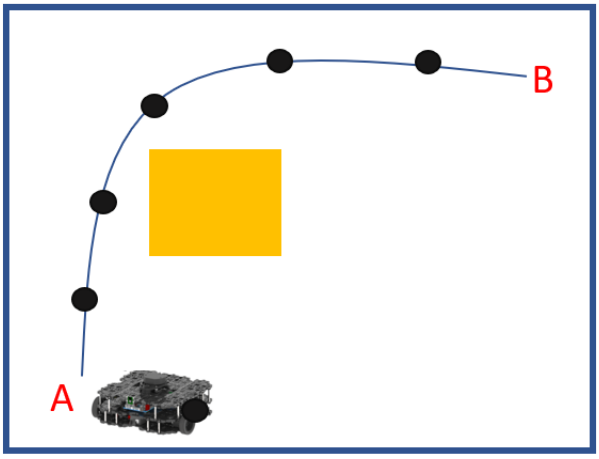

Figure 3.1: Problem statement

Figure 3.2: High-level Software Architecture

3.2.1 Robot Operating System(ROS)

ROS (Robot Operating System) isn't an operating system like Linux, Mac or Windows but it is a open source framework for robotics. ROS framework contain tools and libraries.ROS tools include RVIZ for 3D visualization, Foxglove studio, Webvis, live plotting : rqt plot etc...ROS supports both python libraries as well as C++ libraries.

3.2.2 ROS terminology

ROS Framework consist of:

- ROS Master
- Nodes
- Topic
- Messages

Ros Master:ROS Master is a server that tracks the network addresses of all nodes and it holds information's like which nodes are publishers and which nodes are subscribers and in

which topic they are subscribing or publishing.ROS master should be kept running in Order for the whole system to work.
Nodes: Nodes represents the smallest unit in a ROS framework. It is an executable program. Each process associated with the robot like navigation, grabbing, mapping etc can be programmed as individual nodes.Upon startup, a node registers information such as name, message type, URI address and port number of the node. The registered node can act as a publisher, subscriber, service server or service client based on the registered information, and nodes can exchange messages using topics and services. A node that sends a message via a topic is called as a Publisher Node and the node which receives a message via a topic is called as Subscriber node. Node can send or receive multiple messages via topics.

Topics: Topic represents a channel for communication between nodes. Through topics nodes send and receive messages. However a topic can send only one type of message through it.

Messages: A node sends or receives data between nodes via a message. Messages are variables such as integer, floating point, and boolean. Nested message structure that contains another messages or an array of messages can be used in the message.

A robotic process can be started by launching the Master node. Other nodes containing various functionalities can be started by running them together from a launch file or running them separately. Upon launching the nodes these start sending and receiving message from other node via topics. The integration of all these nodes helps in seamless working of the robot.
This study uses ROS noetic version which is supported in ubuntu 20.04. This version of ROS has integrated gazebo simulator and hence separate package is not required. ROS also has additional tools for plotting node graph and sensor visualisation called as rqt and rviz respectively.

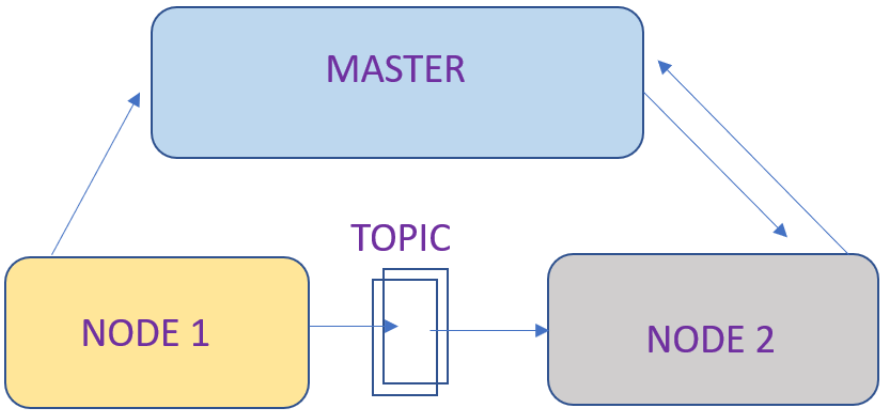

Figure 3.3: ROS Framework

3.2.3 Create a ROS project

Works in ROS are done by creating ROS projects and every ROS project contain at least 1 Package.Steps to be followed while building ROS projects are listed below:

- Create workspace folder to store and build our ROS project.
- Inside workspace folder create another folder named src.src folder is called source folder ,this is the place where needed packages are added.
- Run catkin make command to compile and convert Packages to ROS Packages.

3.2.4 Gazebo

Gazebo is a 3D simulator which provide a robust physics engine, high-quality graphics, and graphical interfaces.And with the help of Gazebo we create training data set for DQN. It performs physics computations, generates synthetic sensor data and offers convenient interfaces. An environmentrequired for training and testing a reinforcement learning agent is saved as a world in gazebo. Apart from that gazebo also has many inbuilt models that can be used in any environment. Models constitute a part of the gazebo world. Gazebo uses SI units for measurement like kilogramns for weight, metres for distance etc.Gazebo also uses a 3-dimensional coordinate system. The simulator also take the physical parameters like gravity,wind etc into consideration while running an environment. Hence an environment developed in gazebo will be highly accurate for real world applications. Gazebo can be integrated with ROS to work together. Gazebo acts as a node in the ROS framework and send all the sensor information to ROS nodes via topics. ROS nodes executes the data and sends commands to gazebo which is visualised during simulation. This work uses gazebo simulator to create simulation environments for both training and testing of the robot.

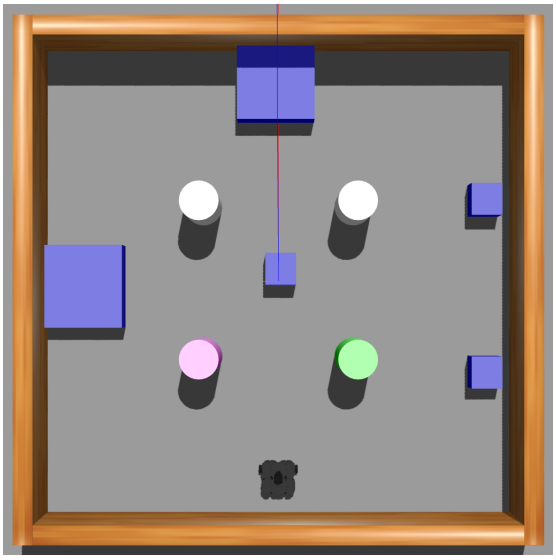

Figure 3.4: Gazebo environment

3.3 Development Environment and Robot

3.3.1 Robot

TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The goal of TurtleBot3 is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expandability. In addition, TurtleBot3 is evolved with cost-effective and small-sized SBC that is suitable for robust embedded system, 360 degree distance sensor and 3D printing technology. Turtlebot3 waffle-pi is the latest version in this series of turtlebot3 robots after Burger and Waffle robot. This study uses turtlebot3 waffle-pi as the robotics agent.

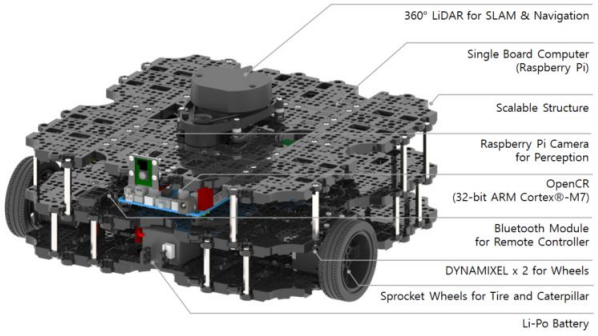

Figure 3.5: Turtlebot3 waffle-pi robot

Table 3.1: Specification of Turtlebot3 waffle-pi

Item	Specification
Maximum translational velocity	0.26 m/s
Maximum rotational velocity	1.82 rad/s (104.27 deg/s)
Size (L x W x H)	281mm x 306mm x 141mm
Weight	1.8kg
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS-01 or LDS-02
IMU	Gyroscope 3 Axis Accelerometer 3 Axis
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C

3.3.2 Small Environment

Custom environment of dimension (10 X 10 m^2) is build in Gazebo simulator as shown in Figure. This environment contain various obstacles between starting location and target location. The obstacles considered here are of different shape and size, and obstacle locations can change with time. Major training of the agent is done on this small environment and the robot should be able to generalize its ability to search target and to avoid collisions. Since training in larger environments with initial random weights take more time to converge ,so the optimal weight obtained from this small environment is transferred to replace initial random weights of the network when training in larger environment.

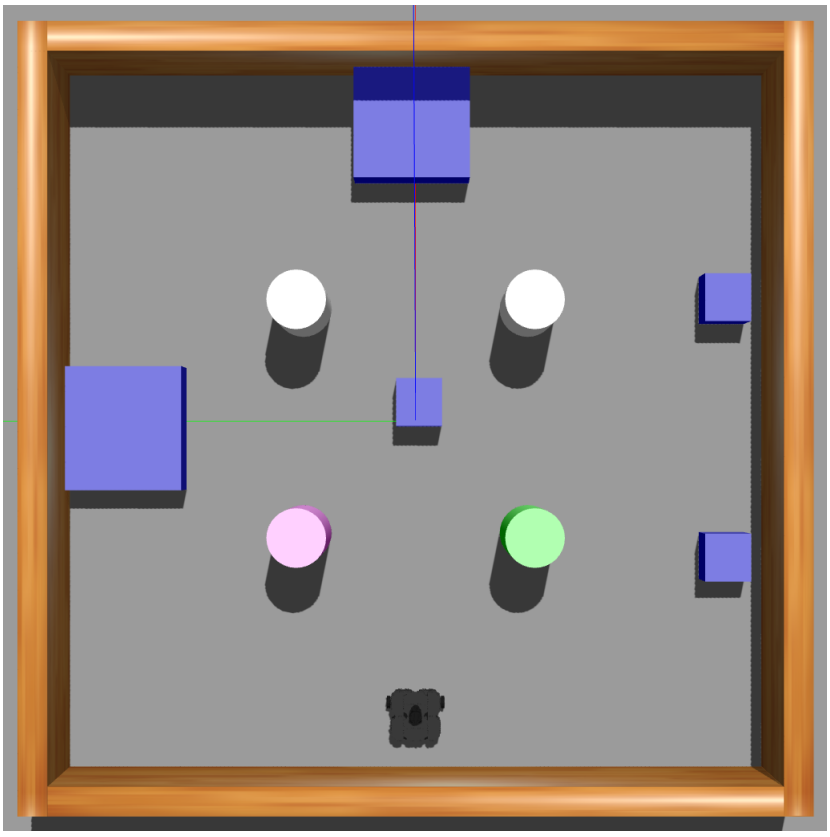

Figure 3.6: (10 X 10 m^2) environment

3.3.3 Large environment

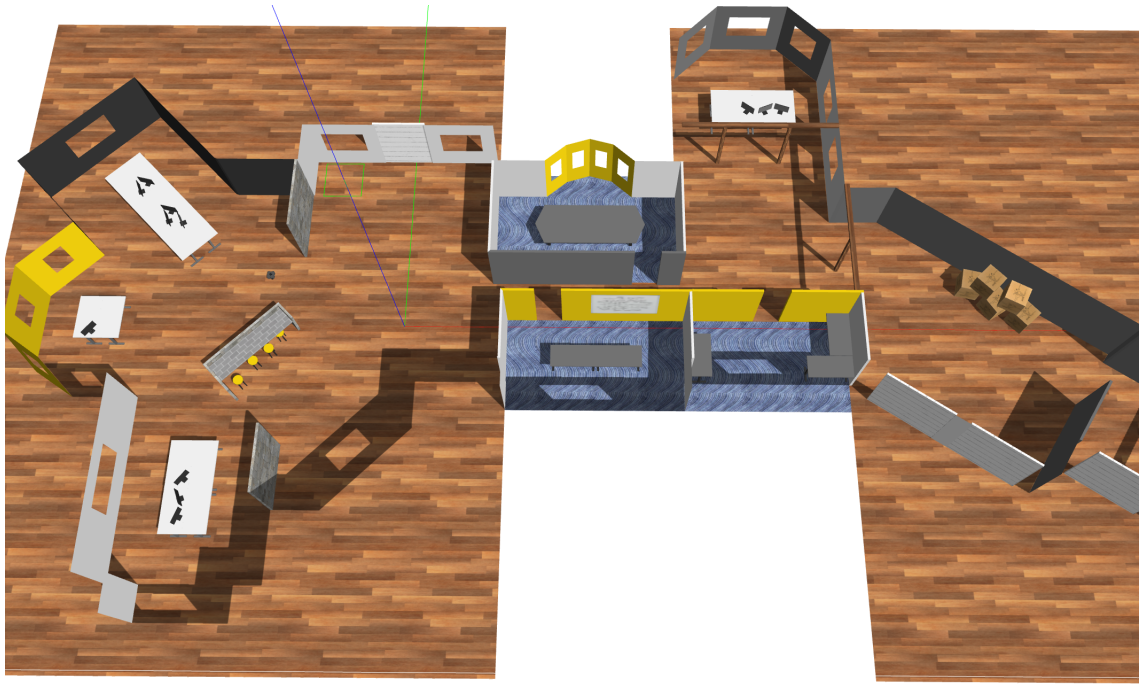

Figure 3.7: Clearpath environment

Clearpath robotics have created new simulation environments for the Gazebo 3D robotics simulator which is shown in above Figure.This simulation worlds resembles that of a real office environment.This environment is equipped with a set of pre-made terrains which include structures and assets like walls, windows, rooms, tables, chairs, door ways, wooden frames, and more. This world is great for testing 2D or 2.5D autonomy using LiDARs and depth sensors.

3.4 OpenAI-Ros

OpenAI provides a complete Reinforcement Learning set of libraries that allow to train software agents on tasks, so the agents can learn by themselves how to best do the task. One of the best tools of the OpenAI set of libraries is the Gym. The Gym allows to compare Reinforcement Learning algorithms by providing a common ground called the Environments.Unfortunately, even if the Gym allows to train robots, it does not provide environments to train ROS based robots using Gazebo simulations. Hence a robot developed in Ros cannot be trained using reinforcement learning algorithm and the openai environments cannot be used for ROS based applications. Hence to bridge this gap a special package called as Ros-openai has been developed to pro- vide an interface to train ROS based robots using reinforcement learning.

Gazebo environment: It connects OpenAI programs to Gazebo. It takes care of func-

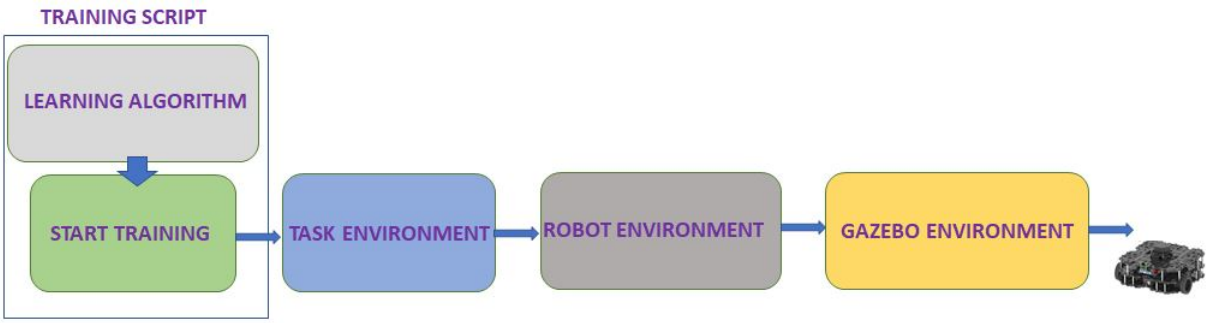

Figure 3.8: Blocks in open-ai package

tions like pausing and resetting of the simulator and environments after every actions and episodes respectively. Hence this block takes care of all the steps that need to be done on the simulator when doing a training step or a training reset (typical steps in the reinforcement learning loop). Basic openai gym functions such as step, seed, reset. close etc are implemented by this block.This block also publishes the last episode reward on the topic named as /openai/reward. This block is independent of the type of environment or the task of the environment.

Robot environment: The robot environment contains all the functionalities associated with a robot. This block check if all the sensor reading are ready and working. The nature of the actions performed by the robot, the type of signals it receives, what those signals do etc are all specified in this block. Hence this block depends on the type of robot and the environment it acts on.

Task environment: Task environment provides a framework for framing the task associated with the robot.This block changes when different robots performing different tasks needs to be trained. However it does not change upon training a new robot having similar interface performing same task. This block specifies how a selected action is applied on the robot, how the observations are obtained, how the reward is computed and whether the episode is terminated or not.

Training script: The reinforcement learning algorithm used for training and testing is described in this block. Hence all the blocks of code start running upon running the training script.

3.5 MDP Formulation

Many multi-stage decision problems could be solved using Reinforcement Learning methods. In this approach we should formulate the problem as a Markov Decision Process(MDP). MDP is a five tuple [S,A,P,γ,R].

- Where, S is set of states called state space.
- A is set of actions called action space.
- P is state transition probability.$\sum p(s') = 1$
- γ is discount factor.$\gamma \in$ [0,1]
- R is reward function.

At any time instant (t) the state of the robot is represented as $s_t \in$ S and action executed by robot is $a_t \in$ A and probability of agent reaching the state s_{t+1} is given by state transition probability p(s_{t+1}/s_t,a_t). When the robot reaches s_{t+1} the agent gets an immediate reward of r_{t+1}.

Reinforcement learning: Reinforcement learning is one of the three types of machine learning alongside supervised and unsupervised learning.And it has many potential application in the field of robotics, computer vision and games.It can be seen that deep learning plugged in a reinforcement learning setting can train an agent that beats human level in a variety of Atari games.In reinforcement learning an agent is made/taught to perceive and interpret its environment, take actions and learn through trial and error. An environment in reinforcement learning refers to the agents world where it interacts. Hence environment comprises of everything outside the agent. The term state is used in RL to describe the configuration of the environment at any time. The agent makes transitions from one state to another by performing actions. Every agent chooses its action by using a policy. Upon taking an action the environment return the new state and a reward to the agent. These actions are rewarded or penalised based on the task the agent needs to achieve. Actions that align with the task are rewarded and that does not are penalised. Hence rewards is used to modify the policy of the agent and thereby its actions.The main aim of the Reinforcement learning algorithm is to find a sequence of decision that maximizes long term reward(R).

$$R = r_0 +_1 +\gamma^2 r_2 + ... \tag{3.1}$$

Through trial and error and reward on the long term the robot need to learn optimal behaviour.

3.6 State/Observations

State representation is important in MDP Formulation. And by observing the state variables the robot is able to make decisions. So, the State representation should contain all relevant information's (about obstacles and target points) required to make a good decision. At any given time step (t) the state vector $s_t \in$ S consists of eight laser data o_t, the distance d_t between robot and target point and angle δ between robot to the target points.

$$s_t = [o_t^0, o_t^1, o_t^2, o_t^3, o_t^4, o_t^5, o_t^6, o_t^7, d_t, \delta] \tag{3.2}$$

The turtlebot3 waffle pi robot is equipped with 360-degree LDS sensors which provide 360-degree scans of the environment. And these 360 laser ranges are being published in topic called /scan. Here instead of taking 360 laser ranges only eight laser ranges are taken from laser. This eight laser range can be done by splitting 360-degree view of laser in to eight equal sectors and minimum of laser ranges in each eight sectors are taken to be included in state representation.

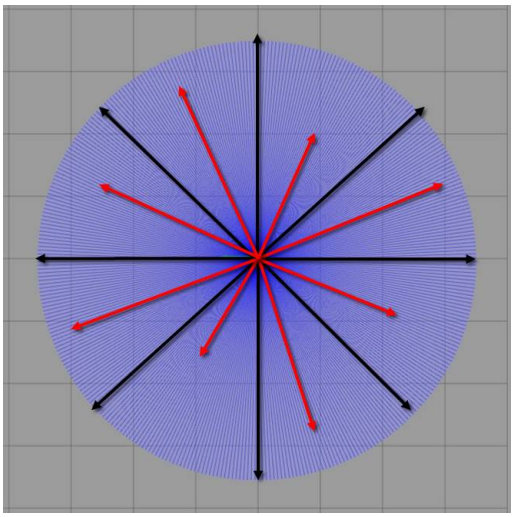

Figure 3.9: Sector division of 360 degree laser data

360-degree laser data:

$$o_{data} = [o_0, o_1, o_2, ...o_{359}] \tag{3.3}$$

360-degree laser data is split in to 8 equal sectors:

$$o_t^0 = min[o_0, o_1, o_2, ...o_{44}] \tag{3.4}$$

$$o_t^1 = min[o_{45}, o_{46}, o_{47}, ...o_{89}]... \tag{3.5}$$

$$o_t^7 = min[o_{315}, o_{316}, o_{317}, ...o_{359}].. \tag{3.6}$$

At any time, instant (t) when the robot is navigating in gazebo environment the gazebo node publishes the information about the Position and Orientation of the robot in the form of float values through a topic called /Odom. Here the position of the turtlebot3 waffle pi robot is represented in Cartesian coordinates X, Y, Z and orientation in Quaternion X, Y, Z, W.Here the motion of the robot is restricted to X-Y plain.And the current orientation of the turtlebot3 waffle pi robot (η) with respect to a fixed frame can be obtained by transforming Quaternion to Euler using suitable transformation matrix.
The angle suspended by goal point with respect to fixed frame can be called as goal angle and is obtained from equation shown below:

$$Goalangle = tan^{-1}[(Y_g - Y)/(X_g - X)] \tag{3.7}$$

Angle δ between robot to goal point is obtained by subtracting angle η from goal angle.

$$\delta = goalangle - \eta \tag{3.8}$$

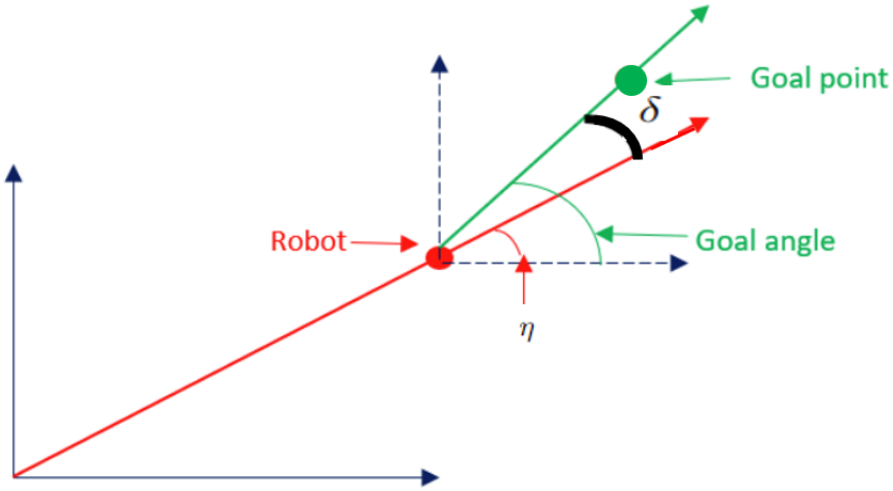

Figure 3.10: Angle δ between robot to goal point

Also, distance between robot and goal point can be calculated from the equation shown below.

$$d_t = \sqrt{[(Y_g - Y)^2 + (X_g - X)^2]} \tag{3.9}$$

Where X_g, Y_g are the co ordinate position of goal point and X,Y are the current co ordinate position of turtlebot3 waffle pi robot.

3.7 Action space

The action space contains six discrete actions:

$$A = [Forward1,\ Forward2,\ Leftturn1,\ Leftturn2,\ Rightturn1,\ Rightturn2] \tag{3.10}$$

which is mapped to values from 0,1,2,3,4, 5. And each entries in action space A is a combination of linear and angular velocities. When an action is selected the corresponding linear and angular velocities are given to the robot. Once the robot archives these velocities then the action is said to be executed.

Table 3.2 shows different actions that are possible for the robot and their corresponding linear and angular velocities.Suppose if the robot choose action forward1 then only a linear velocity of 0.15m/s will be given to the robot, and the current velocity of the robot is continuously monitored.When the current velocity of the robot reaches 0.15m/s the action will be executed and at that time the robot should have traveled some distance let say d_1 in time interval Δt_1.Here d_1 and Δt_1 depends on velocities of both current action and previous action.If both this velocities are same then the current action will be executed in no tme ie Δt_1=0.So a minimum execution time of Δt_{min} for an action is also kept.So if the robot is moving with constant speed then d_1=0.15*Δt_{min}.Two forward actions with different velocities are also

Table 3.2: Different actions of the robot

ACTION	LINEAR VELOCITY (m/s)	ANGULAR VELOCITY (rad/sec)
FORWARD1	0.15	0
FORWARD2	0.26	0
LEFT TURN1	0.15	0.80
LEFT TURN2	0.15	1.45
RIGHT TURN1	0.15	-0.80
RIGHT TURN1	0.15	-1.45

given to robot.So robot can also accelerate its velocity from 0.15m/s to 0.26m/s.For turn actions this happens with angular velocities.

3.8 Reward function

The instant reward obtained by the robot is key element in Deep Reinforcement Learning, which can be obtained by the reward function at each time step.After completing an action the robot get a total reward R_t ,which is a combination of four individual rewards.

$$R_t = [\, R_a \, + \, R_b \, + R_c \, + R_d \,] \tag{3.11}$$

Where R_a is the reward awarded to the robot to learn obstacle avoidance and goal reaching by shortest path.

$$R_a = \begin{cases} -10 \;\; if \;\; \mathrm{d}_o < d_{min} \\ +10 \;\; if \;\; \mathrm{d}_t < d_g \\ -0.1 \;\; otherwise \end{cases} \tag{3.12}$$

Here d_t represents the current distance between robot and the goal point,d_o is the current distance between obstacle and robot,d_{min} is the minimum distance the robot should keep from all objects while moving and d_g is the threshold distance below which we consider robot has reached its target.

Collision with obstacle is the worst case for the robot, so we can give a negative reward of -10 to robot if it hits with the obstacle.By doing so the robot can avoid obstacle, but there is a possibility that the robot should come close with the obstacles and then avoid.But here we need a safe distance of d_{min} from all obstacles while moving so we give negative reward of -10 when $d_o < d_{min}$.Robot is given a positive reward of +10 if $d_t < d_g$, this will encourage the robot to reach goal.The robot also need to find the shortest path from its initial location to goal point, so a negative reward of -0.1 is to robot at every time step.

Reward functions R_b and R_c are designed in order to increase the convergence rate of DQN algorithm. Learning become much more easier when reward functions R_b and R_c are added in total reward.

$$R_b = 0.44 * e^{-0.5*[\delta/0.75]^2} \tag{3.13}$$

From reward R_b robot can get information about the direction in which goal point is present.The reward curve is shown in Figure 3.12.
Reward R_c encourages robot to decrease distance between robot and goal point because agent get more reward when d_t is less.Hence accelerates the training of DQN.

$$R_c = 0.4 - [0.05 * (d_t - d_g)] \tag{3.14}$$

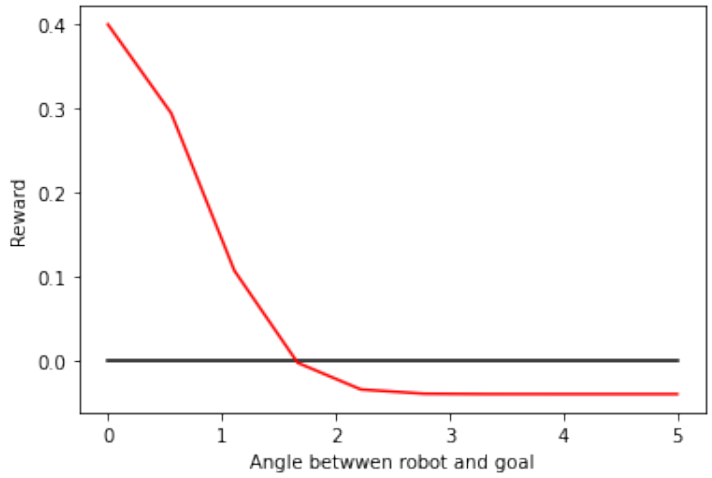

Figure 3.11: Reward R_b

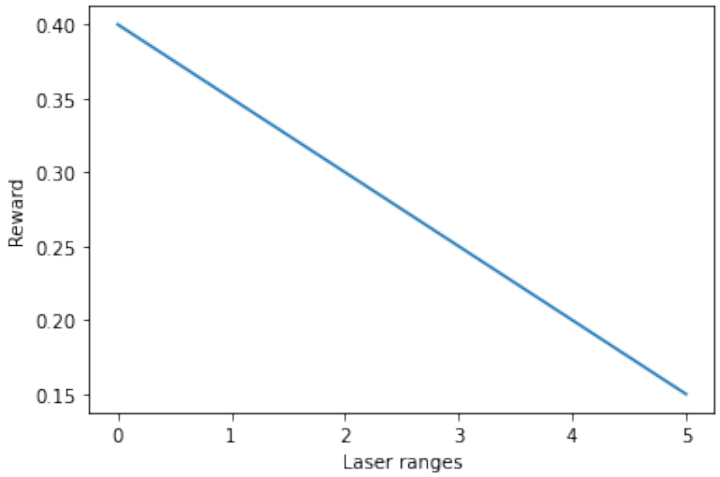

Figure 3.12: Reward R_c

The robot should follow a smooth trajectory ,so reward R_d id defined.It is clear from the equation that the robot will get high negative reward when change in angle δ_t - δ_{t-1} is high.So robot will try to avoid large turns hence trajectory can be made smooth.

$$R_d = -0.04 * (\delta_t - \delta_{t-1}) \tag{3.15}$$

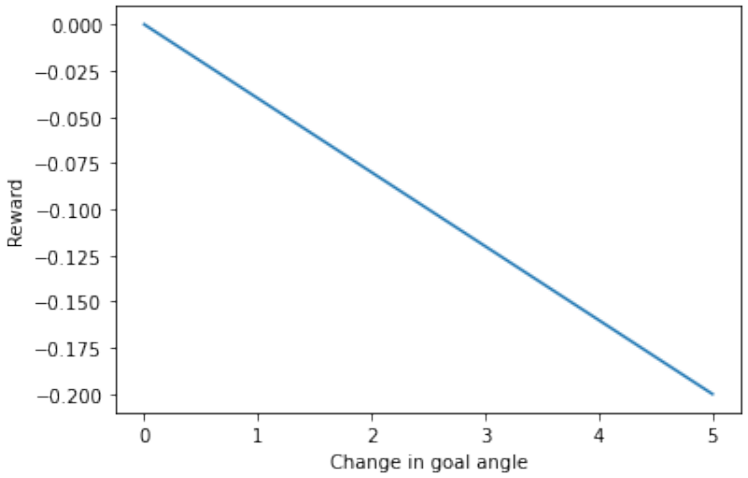

Figure 3.13: Reward R_d

3.9 Deep Q-Learning

With increase in state space or/and action space, storing all of the Q-values corresponding to each state-action combination in the form of a Q-table becomes challenging. It becomes difficult to display a large number of states in the form of a table. The solution to this challenge with Q-learning is to design a network-based function approximator [6] that can approximate the Q-values for any feasible state-action pair (s,a). This non linear function is approximated by a network based function approximator (Where g and h are dimension of state space and action space respectively). The non-linear function is a deep Q-network, which is a deep neural network. The main idea of the deep Q-learning is to replace Q-table with deep Q-network called Q-function. And the optimal Q-function satisfies bellman's equation.

$$Q^*(s,a) = r(s,a,s') + \gamma max_{a'} Q^*(s',a') \tag{3.16}$$

$$Q_\theta(s,a) \sim Q^*(s,a) \tag{3.17}$$

Here θ is the parameter of the network.

3.9.1 DQN Architecture

The no of neurons in the input layer is equal to the number of state variables that is n[0]=10. After input layer there are two hidden layers having n[1]=n[2]=32 neurons respectively. And finally the output layer contains n[3]=6 neurons corresponding to 6 Q-values for a given states and all possible 6 actions. The activation functions used are 'elu' and the optimizer used is Adam.

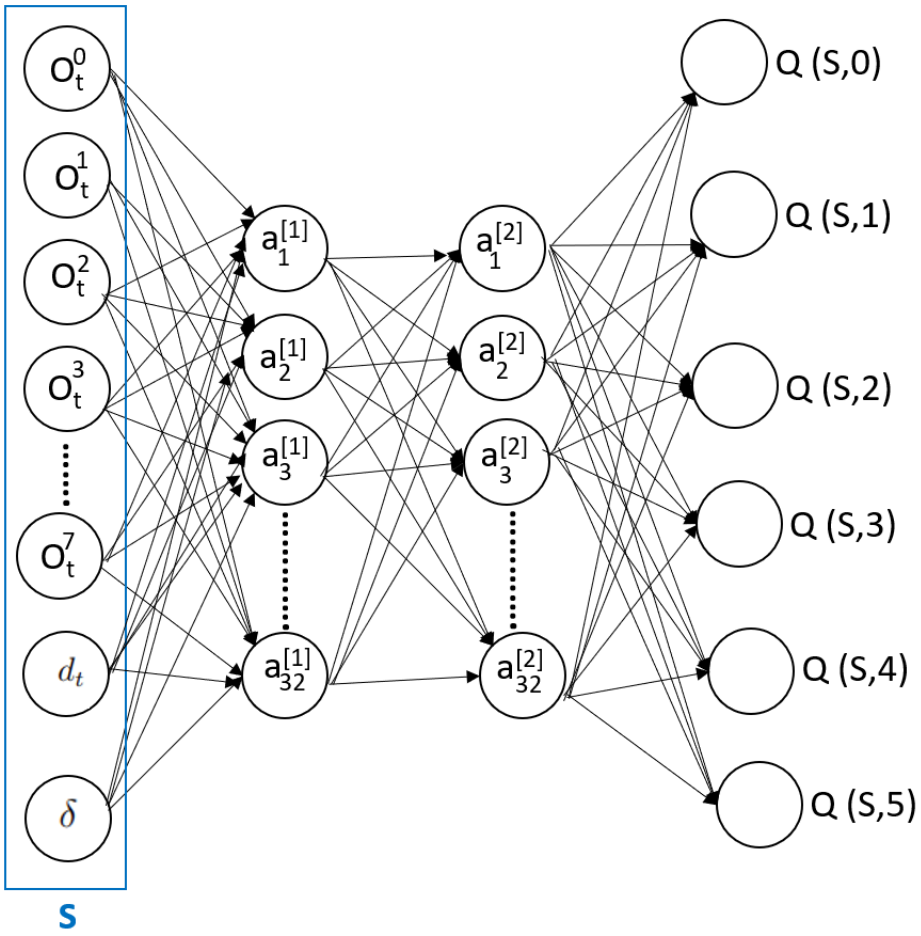

Figure 3.14: DQN Architecture

3.9.2 Training of DQN

Training of DQN starts after accumulating all experiences acquired with each step taken (s,a,s',r) in a replay buffer of fixed size. Each experience consist of four elements: the current state, the action taken by the agent, the reward obtained, the resulting next state. Sample a random batch of experiences from the replay buffer at each training iteration, which reduces the correlation between the experiences in the batch. And train the DQN by performing one gradient descent step with learning rate α on this batch of experiences.
For an experience (s, a, r, s', done)

$$L(\theta) = (y - Q(s, a; \theta))^2 \tag{3.18}$$

$$y = \begin{cases} r(s, a, s') + \gamma maxQ(s', a'; \theta^-) \;\; if s' \neq goalstate \\ r \;\; otherwise \end{cases} \tag{3.19}$$

where y is the Target value.

Instead of one DQN here it make use of two network models the first one is online network and the other one is target network. Here the target network i just a clone of the online network but with parameter θ^{-}.

Online network update parameter at the end of each training iteration and is used to move the agent around. For this the online network takes a state s S as input and outputs all Q-values corresponding to that state s and a A. Taking the argmax of all these Q values provide us with an action

$a = argmax_{a}{}'(Q(s, a'; \theta))$

The target network is used for estimating target Q-values. And the weight of target network $\theta^{-}$is updated using the weight of the online network θ periodically after n episodes. Since the target models are updated much less then online network, the Q-value target are more stable . Since all states has to be visited for learning their Q-values, so agent explores the environment with probability ϵ. For this the algorithm makes use of -greedy strategy.

ϵ Greedy Strategy Algorithm:
With a probability ϵ select a random action
otherwise select $a = argmax_{a}(Q(s, a; \theta))$
Initially the value of ϵ is set to $\epsilon = 1$ and slowly decayed to $\epsilon = 0$. 3 inorder to exploit more on last phase of the training process.
$\epsilon = \epsilon * 0.998$

DQN ALGORITHM:

1. Initialize an online network θ and a target network θ^- randomly such that $\theta = \theta^-$.
2. Initialize α,γ,η,n,k,m.
3. Initialize $\epsilon = 1.0 Initializeepsilondecay = 0.998$

4. For episode in range(max episodes):
5. Initialise the starting state (s_0)
6. For j in range(max time step):
7. With a probability ϵ select a random action a_j and with a probability 1 - ϵ select $a_j = argmax_{a_j'}(Q(s_j, a_j'; \theta))$
8. Execute action a_j and observe next state (s_{j+1}) and reward r_j
9. if s_{j+1}==s_g:
10. create a boolean done = True
11. else:
12. create a boolean done = False
13. Store tuple $< s_j, a_j, s_{j+1}, r_j$,done>in replay buffer.
14. if done ==True:
15. break()
16. else:
17. set current state is equal to next state ($s_j \leftarrow$s$_{j+1}$)
18. if episode greater than η then do:
19. Sample mini-batch of size m from replay buffer.
20. Compute target $y = [\text{reward}]_{m*1} + (1 - [\text{done}]_{m*1}) * \gamma max[\text{Q}(s', a'; \theta^-)]_{m*1}$
21. Estimate L(θ) using the mini-batch samples.
22. Update θ using Adam optimizer.
23. For every n episodes,update target network $\theta^- \leftarrow \theta$.
24. For every episodes, update $\leftarrow \epsilon \times epsilondecay$.
end for.
25. end for

Table 3.3: DQN parameters

DQN parameters	Values
Input layer neurons	10
Number of hidden layers	2
Hidden layer neurons	32
Output layer neurons	6
Gamma	0.99
Batch size	64
Target update frequency	After every 50 time steps
Replay buffer size	20000
Epsilon(initial)	1.0
Epsilon discount per episode	0.998
Minimum epsilon	0.3
Maximum time step in an episode	500

Chapter 4

RESULTS AND DISCUSSION

4.1 Training

Three different training strategies are used to train the agent:

4.1.1 Fixed target training

In fixed target training the robot is trained for a stationary target i.e the target or goal point remains fixed till the training completes. When trained with fixed target the robot is able to learn quickly and find optimal path as shown in Figure 4.1 .The robot learned optimal behaviour before 800 episodes which can be seen from Figure 4.2.The major drawback is that when the position of the target or position of the robot is changed, then the robot is not able to reach target location.This shows poor generalization ability of DQN-enabled robot when trained with fixed target shown in Figure 4.3

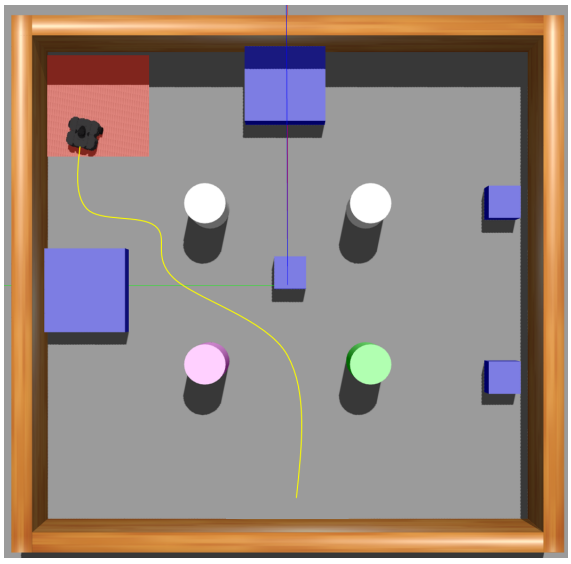

Figure 4.1: Robot reached the target

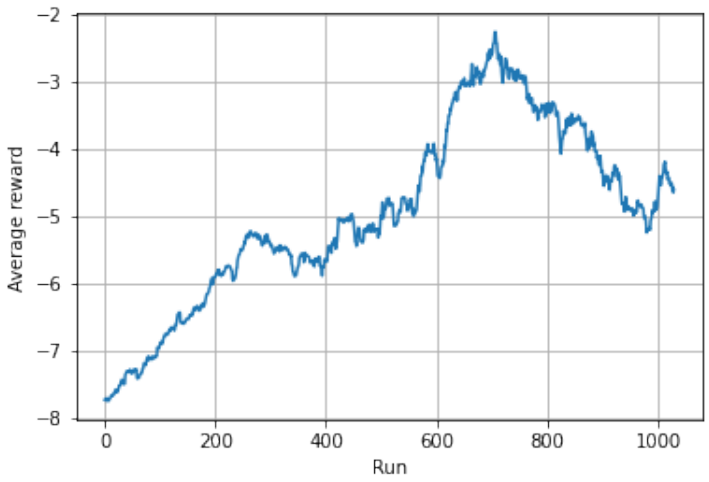

Figure 4.2: Moving average over 100 runs

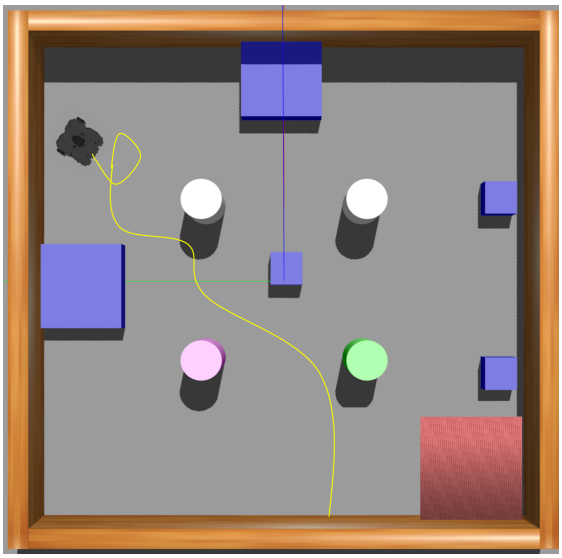

Figure 4.3: Robot unable to reached target

4.1.2 Moving target training

In the case of moving target training the position of the target or goal point changes with episodes.In each episodes there will be new target for the robot to chase.This is done in order to increase the generalization ability of the DQN-enabled robot.From the reward curve shown in Figure 4.4 it is evident that the robot is able to accumulate maximum reward in

between episode 2250 to episode 2500.

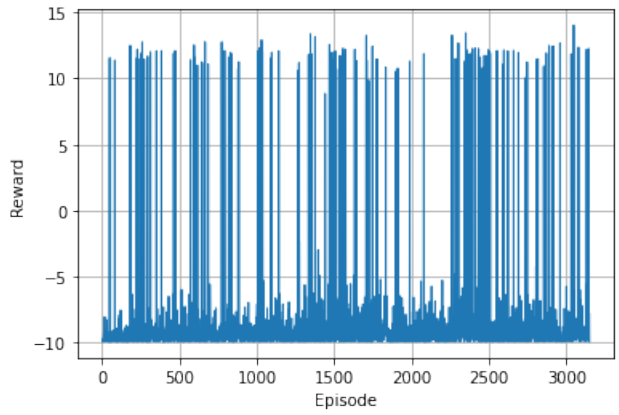

Figure 4.4: Reward curve for moving target training

The Figure 4.5 shows the moving average reward curve obtained by the robot when trained for 3000 episodes.It is clear from the curve that the robot get its maximum consistent reward at episode 2350. So the weight corresponding to episode 2350 is the optimal weight of the network. And when the robot is tested with this weight ,it is able to reach target(which can be at any location) Figure 4.6. So the robot is able to generalize its ability of target finding.

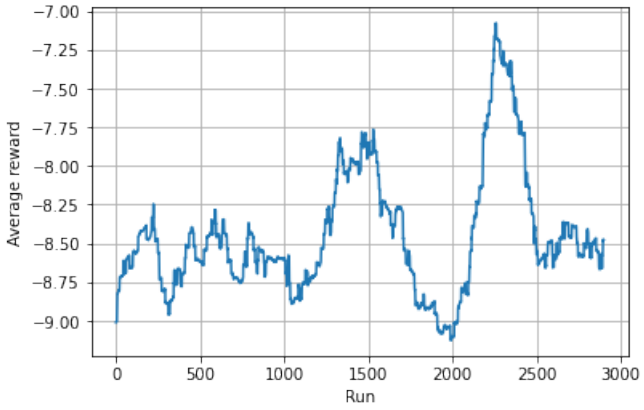

Figure 4.5: Moving average over 100 runs

4.1.3 Target update training

In target update training there will a sequence of goal points. When the training starts the robot select a random goal from the available goal points and fix it as the current goal point.

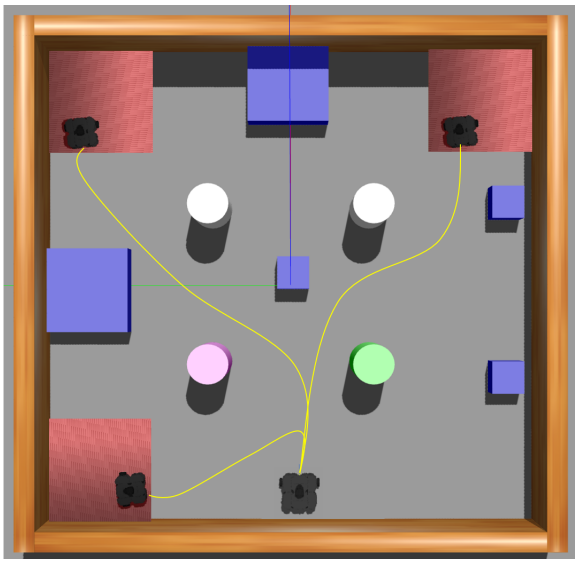

Figure 4.6: Robot is able to reach different target locations

Once the robot reaches the current goal the robot will be awarded with positive reward and current goal point will be updated with new goal point. Unlike moving target training the goal point is not continuously updated with episode's. In this approach the convergence rate is very low when compared with first two training strategies.In some cases we can see divergence when the robot is trained with target update strategy. One such case is shown in Figure 4.7.

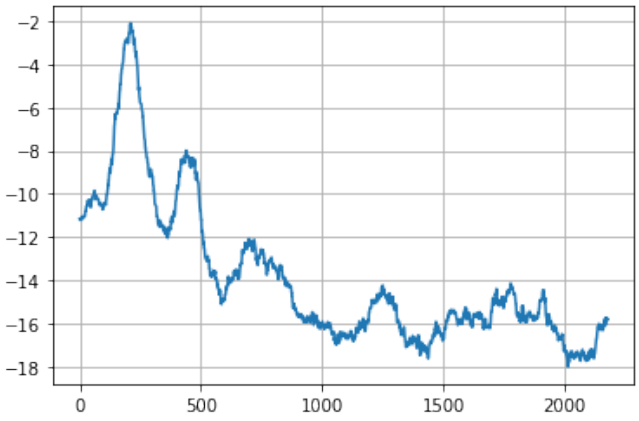

Figure 4.7: Divergence

From the above discussion it can be concluded that proposed moving target training technique is able to provide more generic solution when compared to fixed target training and convergence rate of moving target training is more when compared with target update training.Shown in table 4.1.

Table 4.1: Different training techniques

Training strategy	Convergence rate of of DQN	Generalization	Shown divergence
Fixed target training	Converged in less than 800 episodes	Poor	No
Moving target training	Converged in less than 2500 episodes	Good	No
Target update training	Convergence rate is very low	Unable to learn	Yes

4.2 Effect of rewards on robot behaviour

Here in this section effect of rewards on robot behaviour is investigated.This section is sub divided in to three subsections:

4.2.1 Case 1: Robot trained with R_a

Here the robot is trained with reward R_t=R_a in 10x10 m^2 environment with obstacles.It is found that during exploration phase when the target is close to the starting point the robot is able to reach target most of the time, so after training the robot is able to reach goal in shortest path avoiding obstacles.But when the target is at corner of the environment during exploration the robot was unable to reach target, most of the time the robot hits the obstacle and reset immediately.So the robot has no idea about target.In this case the robot has learned obstacle avoidance but was unable to reach target.

4.2.2 Case 2: Robot trained with R_a+R_b+R_c

In case 2, two additional rewards R_b and R_c are added with reward R_a.From reward R_b robot can get information about the direction in which goal point is present because when angle δ decreases the robot is coming aligned with goal point and the robot is getting an exponentially increasing reward.Similarly from reward R_c the robot get information about the distance between goal and robot.So when robot is trained with R_t= R_a+R_b+R_c the robot is able to reach target in shortest path avoiding obstacles.The down side is that the trajectory made by the robot was not smooth i.e the robot was turning with large δ.

4.2.3 Case 3: Robot trained with R_a+R_b+R_c+R_d

By introducing reward R_d the robot is able to avoid abrupt change in δ angle making the trajectory of the robot smooth.

So total reward R_t=R_a+R_b+R_c+R_d is chosen to train the robot to come up with optimal behaviour.
The above conclusions are summarised in table 4.2.

Table 4.2: Effect of rewards in robot behaviour

Reward R_t	Target reached while training	Constraints satisfied	Constraints not satisfied
R_t=R_a	Very few time	Obstacle avoidance	Target
R_t=R_a+R_b+R_c	Most of the time	Obstacle avoidance, target	Smooth trajectory
R_t=R_a+R_b+R_c+R_d	Most of the time	Obstacle avoidance, target and smooth trajectory	All satisfied

4.3 Testing

It is found that when robot is trained with reward R_t=R_a+R_b+R_c+R_d and with moving target technique the robot is able to come up with an optimal behaviour.And the optimal weight obtained during training is able to navigate mobile robot autonomously from goal 1 to goal 4 by visiting all intermediate goals(Figure 4.8).Here the robot satisfies all the constraints mentioned in the above problem statement.

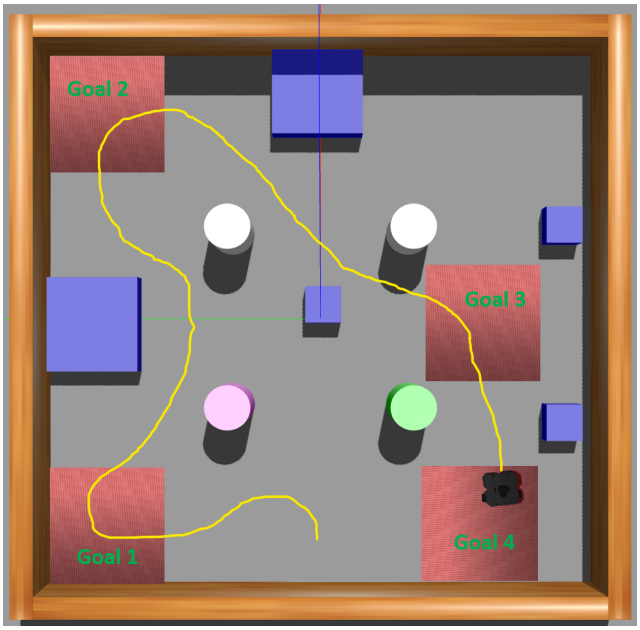

Figure 4.8: Path found by the robot

Chapter 5

CONCLUSION

Here the path planning problem is solved using Deep Reinforcement Learning algorithm mainly Deep Q Learning.The local path planner is able to move the mobile robot autonomously from one point to another point in shortest path avoiding collision's with any obstacles or walls. Other constraints like the robot should follow smooth trajectory, should keep a safe distance from all objects while moving and the robot should try to align with the given path are also addressed.New reward functions R_t=R_a+R_b+R_c+R_d is designed in order to increase the convergence rate of the Deep Q Learning algorithm.Here in this work the proposed moving target training technique is compared with two other training techniques and found that moving target training technique is able to provide more generic solution when compared to fixed target training and convergence rate of moving target training is more when compared with target update training.The developed navigation module(local path planner, decision making and control modules) is able to navigate mobile robot in unknown environment.The generalization ability of DQN-enabled robot is tested in clear path robotic environment (office) and found that the robot is able to navigate with ease.

References

[1] Sichkar VN. Reinforcement learning algorithms in global path planning for mobile robot. In2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM) 2019 Mar 25 (pp. 1-5). IEEE.

[2] Low ES, Ong P, Cheah KC. Solving the optimal path planning of a mobile robot using improved Q-learning. Robotics and Autonomous Systems. 2019 May 1;115:143-61.

[3] Mac TT, Copot C, Tran DT, De Keyser R. Heuristic approaches in robot path planning: A survey. Robotics and Autonomous Systems. 2016 Dec 1;86:13-28.

[4] Xie J, Shao Z, Li Y, Guan Y, Tan J. Deep reinforcement learning with optimized reward functions for robotic trajectory planning. IEEE Access. 2019 Jul 31;7:105669-79.

[5] Raajan J, Srihari PV, Satya JP, Bhikkaji B, Pasumarthy R. Real time path planning of robot using deep reinforcement learning. IFAC-PapersOnLine. 2020 Jan 1;53(2):15602-7.

[6] Li H, Zhang Q, Zhao D. Deep reinforcement learning-based automatic exploration for navigation in unknown environment. IEEE transactions on neural networks and learning systems. 2019 Aug 6;31(6):2064-76.

[7] Wu C, Ju B, Wu Y, Lin X, Xiong N, Xu G, Li H, Liang X. UAV autonomous target search based on deep reinforcement learning in complex disaster scene. IEEE Access. 2019 Aug 5;7:117227-45.

[8] Choy J, Lee K, Oh S. Sparse Actor-Critic: Sparse Tsallis Entropy Regularized Reinforcement Learning in a Continuous Action Space. In2020 17th International Conference on Ubiquitous Robots (UR) 2020 Jun 22 (pp. 68-73). IEEE.

[9] Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N. Dueling network architectures for deep reinforcement learning. InInternational conference on machine learning 2016 Jun 11 (pp. 1995-2003). PMLR

[10] Schaul T, Quan J, Antonoglou I, Silver D. Prioritized experience replay.arXiv preprint arXiv:1511.05952. 2015 Nov 18

[11] G eron A. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc."; 2019 Sep 5

[12] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press; 2018 Nov 13

[13] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S. Human-levelcontrol through deep reinforcement learning. nature. 2015 Feb;518(7540):529- 33.

[14] Li S, Xu X, Zuo L. Dynamic path planning of a mobile robot with improved Q-learning algorithm. In2015 IEEE international conference on information and automation 2015 Aug 8 (pp. 409-414). IEEE.

[15] Panov AI, Yakovlev KS, Suvorov R. Grid path planning with deep reinforcement learning: Preliminary results. Procedia computer science. 2018 Jan 1;123:347-53.

[16] Kim B, Pineau J. Socially adaptive path planning in human environments using inverse reinforcement learning. International Journal of Social Robotics. 2016 Jan;8(1):51-66.