AUTONOMOUS WANDERING OF MOBILE ROBOT USING DEEP Q LEARNING

A Project Report

Submitted by

Mr. JITHIN M

REG NO : TKM20MEAI10

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 WANDERING OF MOBILE ROBOT USING REINFORCEMENT 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: JITHIN M

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 ***AUTONOMOUS WANDERING OF MOBILE ROBOT USING REINFORCEMENT LEARNING*** is a bonafide record of the **Project** presented by **JITHIN M (TKM20MEAI10)**, 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.

JITHIN M

Abstract

Every robotic navigation application need to ensure that the robots avoid collisions with the obstacles in its path. Traditional path planning algorithms can be used to drive a robot from one point to the other in a static environment but they fail in dynamic environment cases. Ensuring the continuous motion of the robot without colliding with the obstacle is necessary for exploration. Hence to enhance the area exploration by self learning a novel method based on deep reinforcement learning is proposed. The current state of the environment are obtained from a 360 degree laser reading and the robot chooses an action from a predefined set of actions. Each action contains a combination of linear and angular velocity helping the robot to move smoothly in the environment. The trained robot is tested under an unknown environment and has shown good generalization of learning. The experiments were conducted on ROS-gazebo framework integrated with openai gym.

Chapter 1

Introduction

Robotics is an interdisciplinary branch of computer science and engineering. Robotics involves design, construction, operation, and use of robots. The goal of robotics is to design machines that can help and assist humans. With the rapid advancement in robotics humans are being replaced by robots in many areas. This has reduced employing humans in hazardous environments. Also robot can do tiring and repetitive jobs more easily than humans. One such area where robots find it application is area exploration and mapping. Exploring an unknown area can be risky or life-threatening for humans. This has led to the deployment of autonomous robots in area exploration and mapping.

Reinforcement Learning is a method of machine learning wherein an agent takes an action and the environment returns a reward. The reward is given based on the quality of the action. Good actions fetch good rewards and wrong actions are penalised. The goal of the agent is to maximise the cumulative reward over long time. The agent learns the optimal actions through trial and error method. This makes reinforcement learning particularly useful when the agents needs to learn itself rather than telling everything.

This work focuses on training a turtlebot3 waffle-pi robot to wander in any environment without collision for infinite time steps. Hence a novel method based on Deep reinforcement learning is proposed to make the robot wander around the environment thereby achieving the task of exploration. The turtlebot3 waffle-pi robot is initially equipped with a 360-degree laser to obtain a good knowledge of the current state of the environment. A set of predefined actions are given to the robot to choose from. A customized reward function is developed to enhance wandering and area exploration. A Deep Q-Network is used to derive the optimum policy based on the states described by the laser readings. Simulation environments were constructed to train the policy evaluate the performance of collision avoidance and area exploration. finally the robot was tested with different simulation environments by changing the position of obstacles in real-time. All the process of training and evaluation is done in the ROS-gazebo framework integrated with openai gym.

The remainder of this report is organized into the following chapters: Chapter 2 reviews different traditional and reinforcement learning techniques for wandering and area exploration using a mobile robot. Detailed implementation framework described in chapter 3. Chapter 4 includes Methodology and RL framework. The experimental results after performing various case analysis are discussed in Chapter 5. Conclusions from this study is presented in chapter 6.

Chapter 2

Related Works

In this section, several studies of path planning and obstacle avoidance using reinforcement learning and other techniques are discussed.

J Choi et al. [1] 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.

Fox D et al. [2] proposed a Dynamic Window Approach to path planning and collision avoidance for mobile robots. This technique creates a velocity search space consisting of all possible velocities and chooses an optimum velocity based on current position, the destination and an objective function. The robot only considers velocities that can be reached within the n ext time interval. These velocities form the dynamic window which is centred around the current velocities of the robot in the velocity space. It converts a Cartesian point (destination) to a velocity command. However this technique mostly provides a curved path and may not be the shortest path. This technique also fails in handling dynamic environments.

H Li et al. [3] 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.

P Fiorini et al. [4] developed a method for robot path planning in dynamic environments by selecting avoidance maneuvers to avoid static and moving obstacles in the velocity space based on the current positions and velocities of the robot and obstacles. The avoidance maneuvers are generated by selecting robot velocities outside of the Velocity Obstacles which represent the set of robot velocities that would result in a collision with a given obstacles that moves at a given velocity, at some future time. Computing new avoidance maneuvers at regular time intervals accounts for general obstacle trajectories. The trajectory from start to

goal is computed by searching a tree of feasible avoidance maneuvers computed at discrete time intervals. An exhaustive search of tree is applicable to on-line planning. However this method has the disadvantage of requiring heavy computing power due to the combination of complex conditions and equations.

Y F Chen et al. [5] developed a socially aware multiagent collision avoidance with deep reinforcement learning algorithm (SA-CADRL) framework exhibiting socially compliant behaviors .They trained the robots to obey time-efficient navigation policy respecting common social norms in a pedestrian-rich environment. In their setting a pair of simulated agents navigate around each other to learn a policy that respect human navigation norms, such as passing on the right and overtaking on the left in a right-handed system. Moreover, SA-CADRL is implemented on robotic hardware, which enabled fully autonomous navigation at human walking speed in a dynamic environment with many pedestrians. The proposed method is shown to enable fully autonomous navigation of a robotic vehicle moving at human walking speed in an environment with many pedestrians.However, if a person is not detected, avoidance cannot be performed, and since detection is not carried out on non-human dynamic objects, it has a disadvantage that collision cannot be avoided.

P Long et al. [6] presented a decentralized sensor-level collision avoidance policy for multi-robot systems, which directly maps raw sensor measurements to an agent's steering commands in terms of movement velocity. Inorder to reduce the performance gap between decentralized and centralized methods, a multi-scenario multi-stage training framework was developed to learn an optimal policy. The policy is trained over a large number of robots on rich, complex environments simultaneously using a policy gradient based reinforcement learning algorithm. We validate the learned sensor-level collision avoidance policy in a variety of simulated scenarios with thorough performance evaluations and show that the final learned policy is able to find time efficient, collision-free paths for a large-scale robot system. The learned policy generalized well to new scenarios that do not appear in the entire training period, including navigating a heterogeneous group of robots and a large-scale scenario with 100 robots.But, a learned policy focusing on local collision avoidance cannot replace a global path planner when scheduling many robots to navigate through complex environments with dense obstacles.

Chapter 3

Implementation Framework

3.1 Robot Operating System

ROS (Robot Operating System) is a software framework for programming robots. Unlike the other operating systems like Windows or Linux it is not an operating system. It is a meta operating system which provides all services provided by an operating system, including hardware abstraction, low-level device control etc. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. It consists of infrastructure, tools, capabilities and ecosystem. ROS also performs processes such as scheduling, loading, monitoring and error handling by utilizing virtualization layer between applications and distributed computing resources.

Master: The master acts as a name server for node-to-node connections and message communication. Every node can be registered with the master its information can be retrieved when needed. This feature helps ROS to work in very large and complex environments.

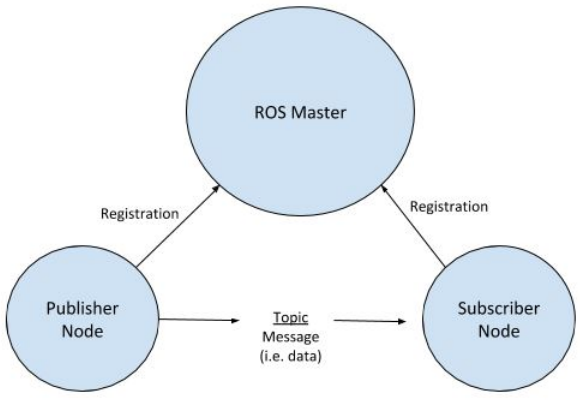

Figure 3.1: Node in ROS

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 is first 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.

3.2 Turtlebot3 waffle-pi

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.

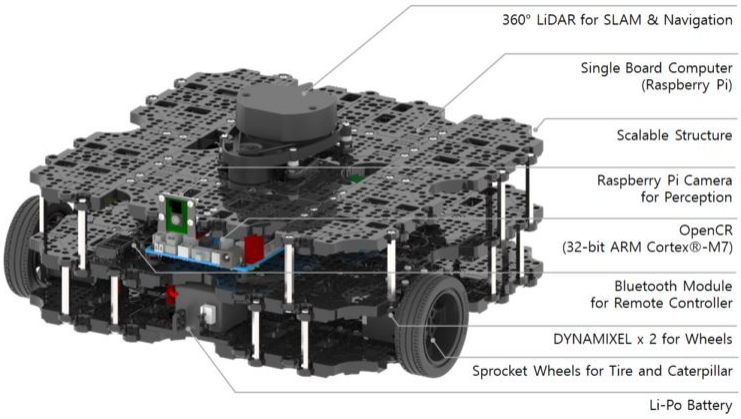

Figure 3.2: Specifications of Turtlebot3 waffle-pi robot

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 Gazebo

Gazebo is an open-source 3D robotics simulator.It provides realistic rendering of environments including high-quality lighting, shadows, and textures. It performs physics computations, generates synthetic sensor data and offers convenient interfaces. An environment required 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.

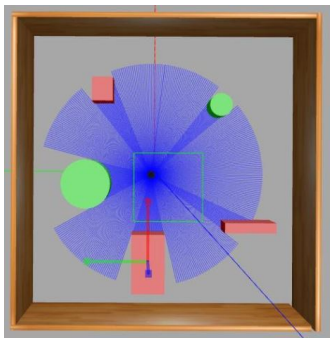

Figure 3.3: Training environment created in gazebo

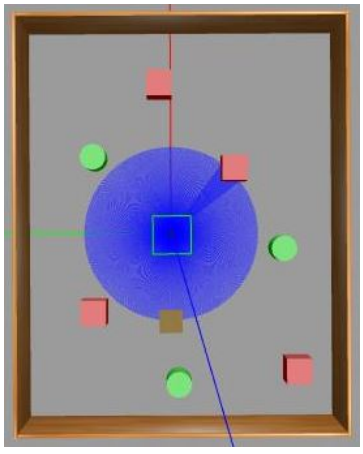

Figure 3.4: Testing environment created in gazebo

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 provide an interface to train ROS based robots using reinforcement learning.

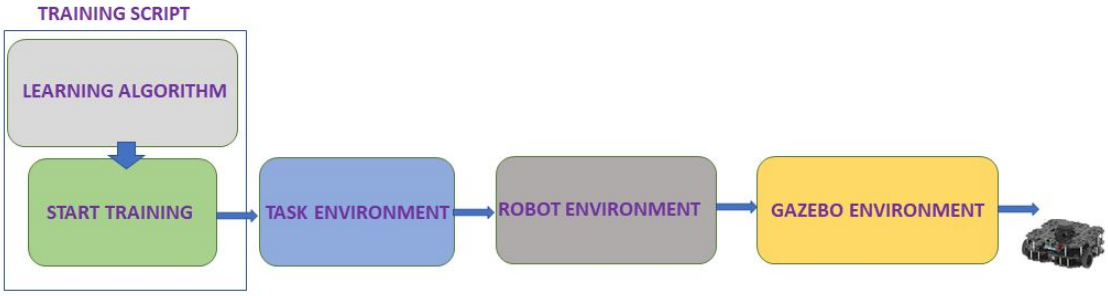

Figure 3.5: Blocks in open-ai package

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

Chapter 4

Methodology

This chapter describes the detailed methods and techniques used in training a robot using RL framework.

Reinforcement learning: Reinforcement learning is one of the three types of machine learning alongside supervised and unsupervised learning. 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. Every reinforcement learning cycle begins with an agent using some sort of policy to perform an action on the environment. Upon performing this action it gets back the information about the new state of the environment and a reward. The goal of the agent at all times is to take actions that maximises the cumulative reward it obtains.

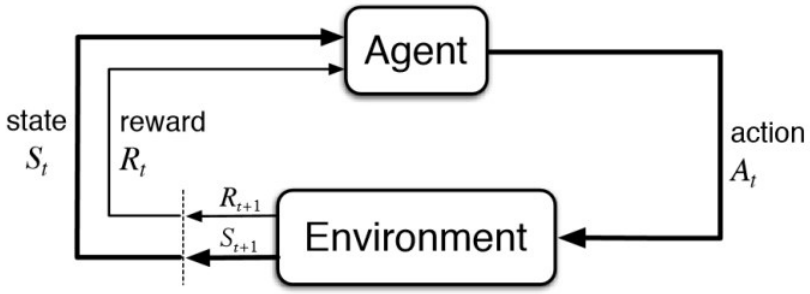

Figure 4.1: Reinforcement learning cycle

Figure 4.1 shows one cycle of reinforcement learning. Here an agent receives the state information at time t (S_t), it performs an action (A_t) and receives the information about about

the new state and reward at next time instant denoted by S_{t+1} and R_{t+1}.
The agent tries to maximise the cumulative reward from all the sequence of cycles as mentioned above. Let the cumulative reward be represented by G_t.

$$G_0 = r_0 + \gamma_1 + \gamma^2 r_2 + ...r_\infty \tag{4.1}$$

A discount factor γ is exponentially multiplied with successive rewards. The value of γ lies between 0 to 1. This is to give weightage for immediate and future rewards depending upon the certainty those rewards. Higher values of γ implies more certain about future rewards and vice-versa. Multiplying with a discount factor also prevents the sum from attaining infinite reward value in case of continuous tasks. Every RL algorithm tries to derive a policy such that the agent takes action to maximise cumulative reward at all time step. The remaining part of this chapter define how to frame a problems statement followed by representing it in MDP Formulation which involves defining state , action and rewards and then finally applying RL algorithm to solve the problem.

4.1 Problem Statement

The objective of the turtlebot3 waffle-pi robot is to wander around a closed environment containing various obstacle satisfying the given constraints:

1. It should always avoid collision with the obstacles or with the walls of the environment.
2. It should travel in a smooth trajectory avoiding abrupt changes in its path.
3. It should never stop wandering unless a dynamic object is placed in its path very close to it all of a sudden.
4. It should avoid or minimise repeated path while wandering.

4.2 Markov Decision Process (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 possible configuration of the system 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}.

4.3 State/Observations

To formulate the collision avoidance problem within the reinforcement learning framework the state should contain information about the environment. This is obtained by using the data from the lidar sensor mounted on top of the turtlebot. The lidar data contains 360 values for each angle. Each value in the lidar data represents the distance from the obstacle at that angle.

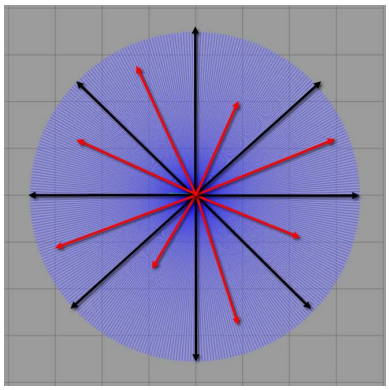

Figure 4.2: Sector division of 360 degree laser data

Inorder to reduce the number of elements in the state space the 360 degree laser data is divided into 8 sectors divided by 45 degrees. The minimum from each of this sectors is obtained to know the range of the closest obstacle in this sector. In Figure 4.1 the circle represents the 360 degree laser scan. The black arrows divide the laser scan into 8 sectors and the red line represents the individual minimum values from each sector. These 8 minimum valued laser data is used to obtain the state of the environment. Hence

$$l_{data} = [l_0, l_1, l_2, ...l_{359}] \tag{4.2}$$

Where, l_0,l_1, . . .l_{359} represents the distance between robot objects (in metres) by respective angles. These total laser data is divided into 8 sectors each containing 45 values from 45 degree sectors.

$$l^{s1} = [l_0, l_1, l_2, ...l_{44}] \tag{4.3}$$

$$l^{s2} = [l_{45}, l_{46}, l_{47}, ...l_{89}]... \tag{4.4}$$

$$l^{s8} = [l_{315}, l_{316}, l_{317}, ...l_{359}] \tag{4.5}$$

The minimum values from all these sector data is used to represent the state of the environment.

$$l_{min} = [l^{s1}_{min}, l^{s2}_{min}, l^{s3}_{min}, ...l^{min}_{s8}] \tag{4.6}$$

Hence the state of the environment at any time instance s_t is represented as:

$$s_t = [l_{min}] \tag{4.7}$$

4.4 Actions

The action spaces consists of six discrete actions.

$$A_t = [forward,\, left\,turn,\, right\,turn,\, rotate\,left,\, rotate\,right,\, stop] \tag{4.8}$$

Each action is encoded with a combination of linear and angular velocities. These velocities are set considering the motion dynamics of the turtlebot3 to enable smooth navigation.

Table 4.1: Action space of Turtlebot3 waffle-pi

Action	Velocities
Forward	Linear velocity = 0.3 m/s Angular velocity = 0.0 m/s
Left turn	Linear velocity = 0.1 m/s Angular velocity = 0.1 m/s
Right turn	Linear velocity = 0.1 m/s Angular velocity = - 0.1 m/s
Rotate left	Linear velocity = 0.0 m/s Angular velocity = 0.3 m/s
Rotate right	Linear velocity = 0.0 m/s Angular velocity = - 0.3 m/s
Stop	Linear velocity = 0.0 m/s Angular velocity = 0.0 m/s

4.5 Rewards

In this study, the robot aims to wander around the environment continuously without collision, thereby exploring the environment. To achieve this rewards were considered separately. The total reward is the sum of four individual rewards.

$$R_t = [\, r_a \,+\, r_l \,+ r_e \,+ r_c \,] \tag{4.9}$$

where R_a, R_l, R_e denote the rewards for actions,laser data and exploration and collision respectively.

4.5.1 Action Reward:

Positive rewards are given for forward and turn actions to enhance exploration. Forward action is preferred over turn actions for exploration as sudden turn actions make would make the robot wobble. Hence forwards rewards is weighed higher than turn actions. Inorder to discourage repeated rotation and stoppage these actions are given Negative rewards. Among these two, rotate actions are more preferred than stop actions as stop actions destroys the momentum of the robot. Hence rotate reward is given higher weightage than stop reward.

These conclusions about action rewards were derived after training and testing for three different cases as mentioned in chapter 5

$$r_a = \begin{cases} r_1 & \text{if action} = \textbf{forward} \\ r_2 & \text{if action} = \textbf{left turn} \text{ or } \textbf{right turn} \\ r_3 & \text{if action} = \textbf{rotate left} \text{ or } \textbf{rotate right} \\ r_4 & \text{if action} = \textbf{stop} \end{cases} \tag{4.10}$$

4.5.2 Laser Reward:

At all times the robot should avoid getting too close to the obstacle during exploration. Inorder to incorporate this in its learning a laser based reward is awarded depending upon the distance between the robot and the smallest laser reading (which implies the closest point) from the state space. This reward function is obtained by drawing a straight line from (6,1) to (0,0). Here the 6 and 1 (6,1) represents the maximum laser range in metres and the corresponding maximum laser reward respectively. Following this reward curve the robot gets higher rewards with increase in distance and vice-versa.

$$r_l = 0.127 * l_{min}^{sec} - 0.0345 \tag{4.11}$$

where l_{min}^{sec} is the smallest value of the laser reading from the state space. The reward curve is shown in Figure 4.2

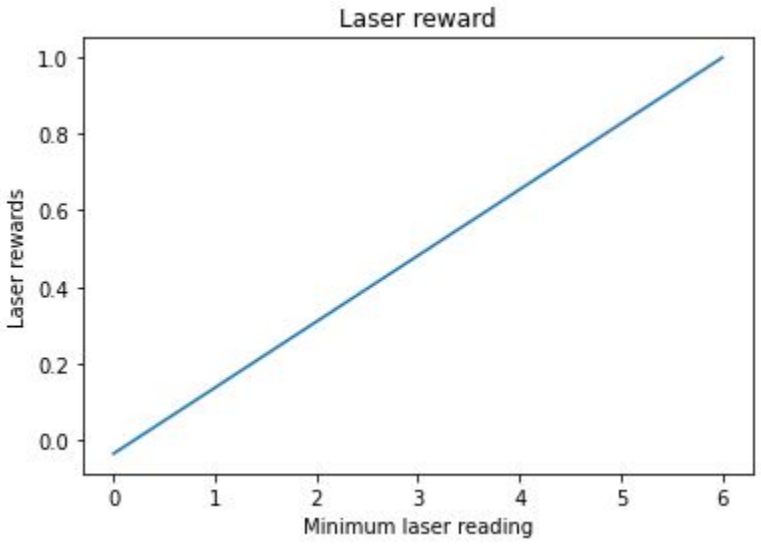

Figure 4.3: laser based reward

4.5.3 Exploration Reward:

Inorder to make the robot to explore the entire environment 40 different explorations points were randomly generated in each episode. The position of these points were varied in each episode. The robot received a positive reward upon reaching each of these points. Hence

$$r_e = 10 \tag{4.12}$$

where R_e represents the reward on reaching an exploration point.

The robot should avoid collision with the objects in the environment. Collision with an object is considered as a worst case scenario in this study. Hence a high negative reward is given upon the occurrence of such an event and the episode is terminated.

$$r_c = \begin{cases} -100 \ \ if \ \ collided \\ 0 \ \ if \ \ not \ \ collided \end{cases} \tag{4.13}$$

Where r_c represents the reward based on collision.

4.6 Deep Q-Learning

With increase in number of states and/or actions, storing all Q-values corresponding to each state-action pair 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.

$$Q^*(s,a) = r(s,a,s') + \gamma max_{a'} Q^*(s',a') \tag{4.14}$$

$$Q_\theta(s,a) \sim Q^*(s,a) \tag{4.15}$$

Here θ is the parameter of the network.

4.6.1 DQN Architecture

The no of neurons in the input layer is equal to the number of state variables that is n[0]=8. 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 8 actions. The activation functions used are 'elu' and the optimizer used is Adam.

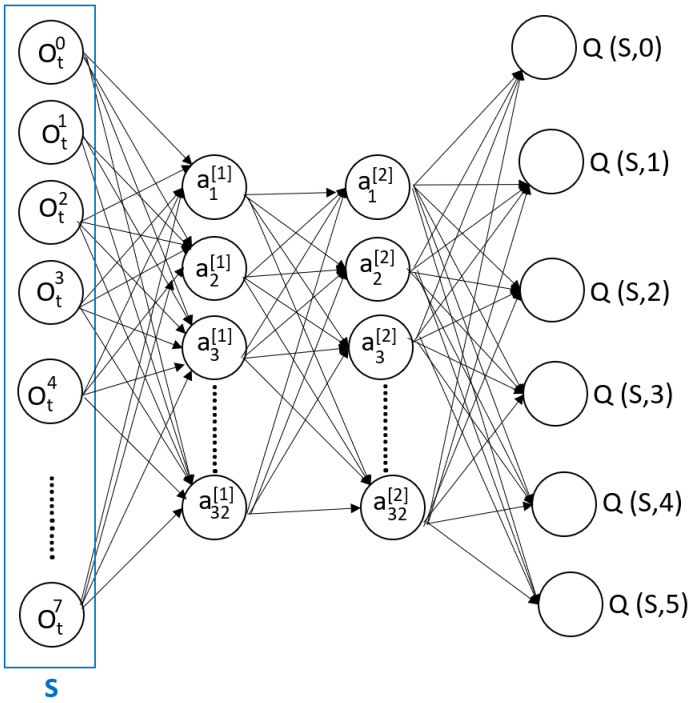

Figure 4.4: DQN Architecture

4.6.2 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)

$$L(\theta) = (y - Q(s, a; \theta))^2 \tag{4.16}$$

$$y = \begin{cases} r(s, a, s') + \gamma \times maxQ(s', a'; \theta^-) \;\; if s' \neq goalstate \\ r \;\; otherwise \end{cases} \tag{4.17}$$

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
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 during last phase of the training process.

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 $\epsilon = 1.0$
4. Initialize epsilon decay = 0.998
5. For episode in range(max episodes):
6. Initialise the starting state (s_0)
7. For j in range(max time step):
8. With a probability ϵ select a random action a_j and with a probability 1 - ϵ select $a_j = argmax_{a_j}{}'(Q(s_j, a'_j; \theta))$
9. Execute action a_j and observe next state (s_{j+1}) and reward r_j
10. if s_{j+1}==s_g:
11. create a boolean done = True
12. else:
13. create a boolean done = False
14. Store tuple $< s_j, a_j, s_{j+1}, r_j$,done>in replay buffer.
15. if done ==True:
16. break()
17. else:

18. set current state is equal to next state ($s_j \leftarrow s_{j+1}$)
19. if episode greater than η then do:
20. Sample mini-batch of size m from replay buffer.
21. Compute target $y = [\text{reward}]_{m*1} + (1 - [\text{done}]_{m*1}) * \gamma \max[Q(s', a'; \theta^-)]_{m*1}$
22. Estimate $L(\theta)$ using the mini-batch samples.
23. Update θ using Adam optimizer.
24. For every n episodes,update target network $\theta^- \leftarrow \theta$.
25. For every episodes, update $\leftarrow \epsilon \times epsilondecay\theta^- \leftarrow \theta$.
26. end for.
27. end for

4.7 Experiment

The turtlebot3 waffle-pi is imported and its parameters are adjusted for the specific use of this study. The laser visualisation is enabled to help sense the environment. However the inbuilt camera is disabled as camera readings are not used in this work.

Table 4.2: Specification of Turtlebot3 waffle-pi

Parameters Values	
Initial linear forward speed	0.0 m/s
Initial linear turn speed	0.0 m/s
Linear forward speed	0.3 m/s
Linear turn speed	0.1 m/s
Angular speed	0.1 m/s
Angular rotation speed	0.25 m/s
Maximum linear acceleration	1.0 m/s2
Laser sectors	8
Minimum laser value	0 m
Maximum laser value	6 m
Minimum range to consider collision	0.2 m

A customised environment is created in gazebo using the models and objects in the simulator. The objects are placed randomly in the environment. The robot is initially set fixed at the origin of the environment in the gazebo. This gazebo environment containing various models including the robot is launched along with the ROS master node.
The training environment as shown in Figure 4.4 consist of the robot at the origin of the gazebo environment. The blue circle surrounding the robot represents the 360 degree laser

beams. Square and cylindrical blocks of red and green colour are randomly placed in the environment. The size of these blocks is varied to impart more generalisation while training. All the above mentioned models are enveloped using four walls describing the boundary of the environment. This boundary restricts the otherwise infinite exploration option of the robot.

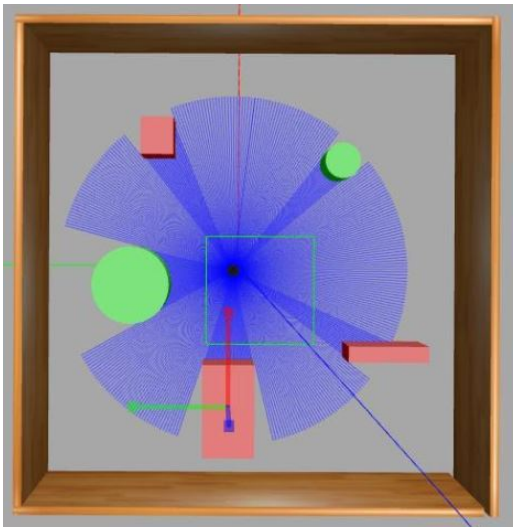

Figure 4.5: Training environment for turtlebot3 waffle-pi

The agent is subjected to training by running the training script from the openai ros package.This enables the agent to run for a predefined number of episodes or until terminated manually. A deep reinforcement learning algorithm called Deep Q-Network is used to derive a policy for the training agent. The parameters of the network are given in Figure. During training the rewards obtained and total steps taken in each episode is calculated.

After training, testing is done for same and different environments. The results obtained during training and testing and elaborated and discussed in next chapter.

Table 4.3: DQN learning parameters

Feature Name	Description
Input layer neurons	8
Number of Hidden layers	2
Hidden layer neurons	32
Output layer neurons	6
Alpha (learning rate)	0.001
Gamma	0.8
Batch size	64
Weight update episode	10
Replay buffer size	20000
Epsilon (initial)	1.0
Epsilon discount per episode	0.998
Minimum epsilon	0.3
Number of training episodes	475

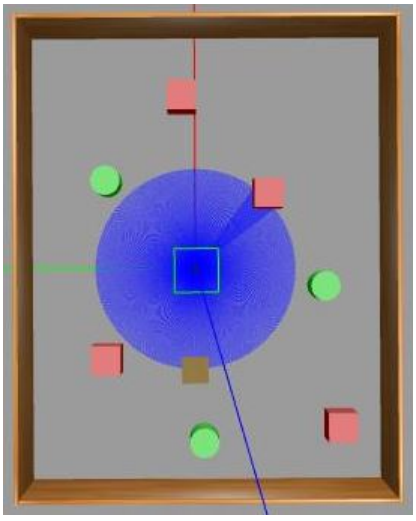

Figure 4.6: Testing environment for turtlebot3 waffle-pi

Chapter 5

Results and Discussion

Training and testing were done for three different cases.

Training: The robot was trained for 500 episodes using epsilon-greedy action policy. The value of epsilon was initially set at unity at the beginning of training. This value was gradually decremented by a factor of 0.998 after every episodes. This made the robot to explore the environment taking random actions initially. At later stages of training it took more actions by using the Q network. Hence at later stages of training its rewards and number steps increased. However it collided with the obstacles even at this stage of training due to smaller yet retaining value of epsilon at 0.3. This is shown by figure 5.5 and 5.6.

Testing: After training the robot was tested with same and different environments. During testing all the actions were chosen by using the trained Q network. The detailed results during testing with different cases is mentioned below.

Case 1: Positive rewards for all actions without exploration reward

Case 2: Negative rewards for stop and rotate action without exploration reward

Case 3: Negative rewards for stop and rotate action with exploration reward

5.1 Case 1:

Initially the robot was trained with positive rewards for all actions. But, During testing it was observed that robot sometimes preferred stop and rotate actions more than other actions. Though it ensured collision avoidance, it stopped the robot from exploring the environment. This is observed by a very small area generated in the map because the robot stopped when it reached close to an object. This is shown in figure 5.1. Inorder to solve the problem of collision avoidance and area exploration together the rewards were later revised.

Table 5.1: Rewards in case 1

Rewards	value
Action rewards	$r_1 = 0.5$ $r_2 = 0.5$ $r_3 = 0.5$ $r_4 = 0.5$
Laser reward	$r_l = 0.127 * l_{min}^{sec} - 0.0345$
collision reward	$r_c = \begin{cases} -100 \ \ if \ \ collided \\ 0 \ \ if \ \ not \ \ collided \end{cases}$

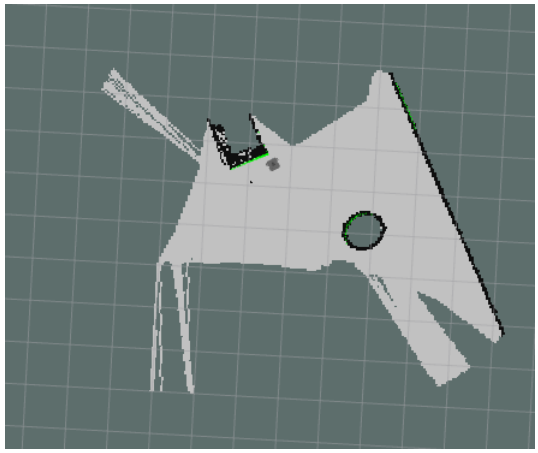

Figure 5.1: Testing environment for turtlebot3 waffle-pi

5.2 Case 2:

In the first it was observed that the robot preferred to take the stop and inplace rotate action. Inorder to discourage this behaviour negative rewards were given to these to actions. This ensured that the robot did not stop during the exploration phase. However it suffered from a looping problem where it repeats the exploration in a small area. This is observed by generating a map while testing and is shown in figure 5.2

5.3 Case 3:

Inorder to end the problem of collision avoidance and area exploration simultaneously case2 was later modified with an additional reward known as exploration reward. This helped the robot to learn to explore the environment covering maximum area. This is

Table 5.2: Rewards in case 2

Rewards	value
Action rewards	$r_1 = 0.5$ $r_2 = 0.25$ $r_3 = -0.25$ $r_4 = -0.5$
Laser reward	$r_l = 0.127 * l_{min}^{sec} - 0.0345$
collision reward	$r_c = \begin{cases} -100 \ if \ collided \\ 0 \ if \ not \ collided \end{cases}$

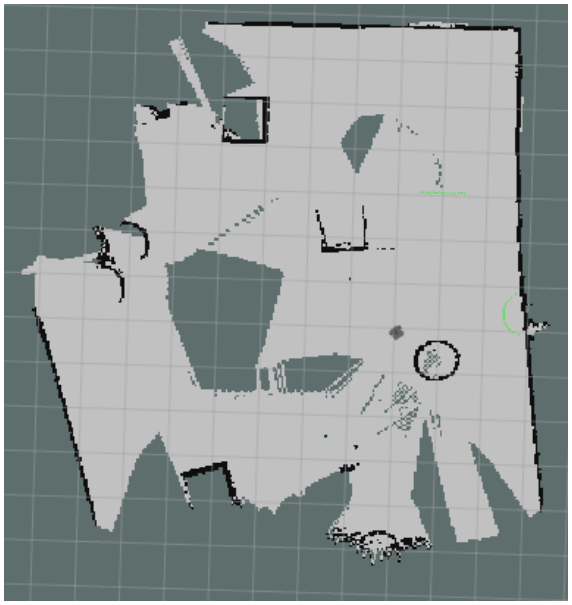

Figure 5.2: Testing environment for turtlebot3 waffle-pi

The moving average reward obtained during training for 475 episodes is shown in figure 5.4. From the figure it is evident that the curve almost flattens after episode 355. Hence the robot will get a consistent maximum reward after 355 episodes.
From figure 5.5 it is seen that the number of steps per episode shows an increasing trend with increase in episode. The moving average cure also shows an increasing trend as shown in figure 5.6.

Hence from the above discussions it can be concluded that choosing the best weight and testing with the same gives better performance.

Table 5.3: Rewards in case 2

Rewards	value
Action rewards	$r_1 = 0.5$ $r_2 = 0.25$ $r_3 = -0.25$ $r_4 = -0.5$
Laser reward	$r_l = 0.127 * l_{min}^{sec} - 0.0345$
Exploration reward	$r_e = 10$
collision reward	$r_c = \begin{cases} -100 \ if \ collided \\ 0 \ if \ not \ collided \end{cases}$

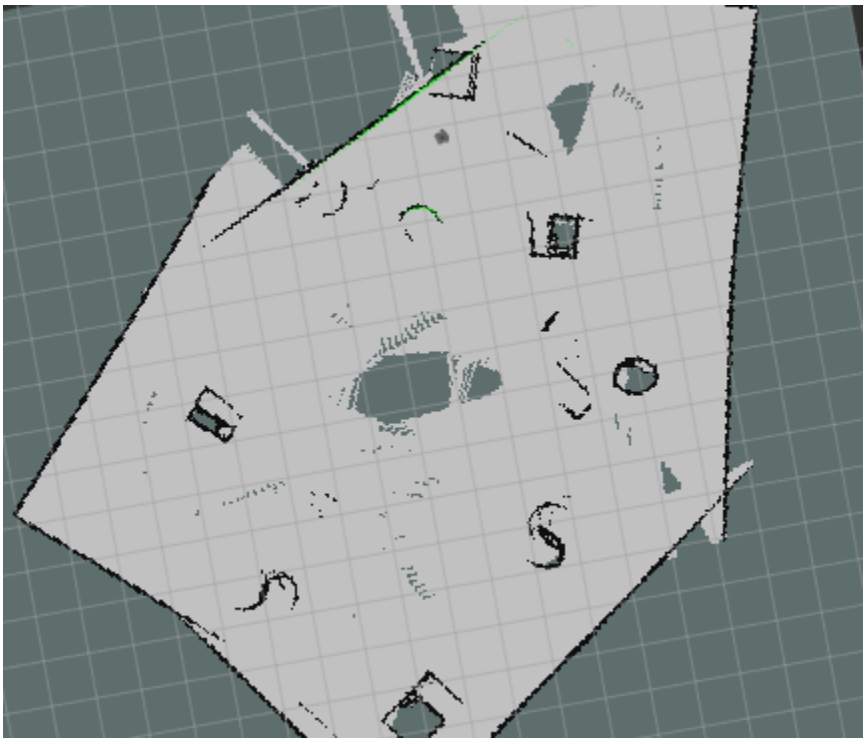

Figure 5.3: Testing environment for turtlebot3 waffle-pi

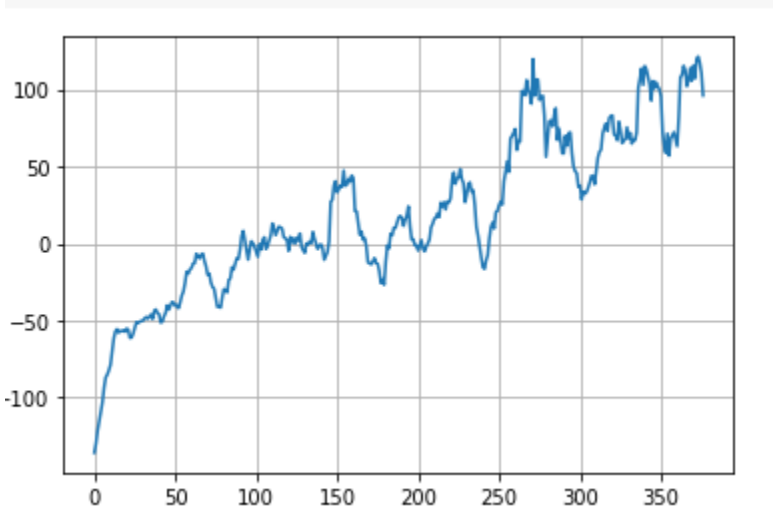

Figure 5.4: Moving rewards obtained while training

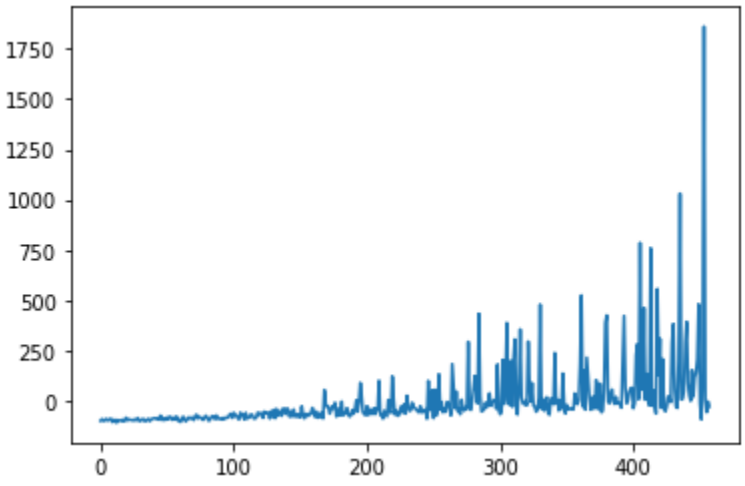

Figure 5.5: Steps taken in each episode while training

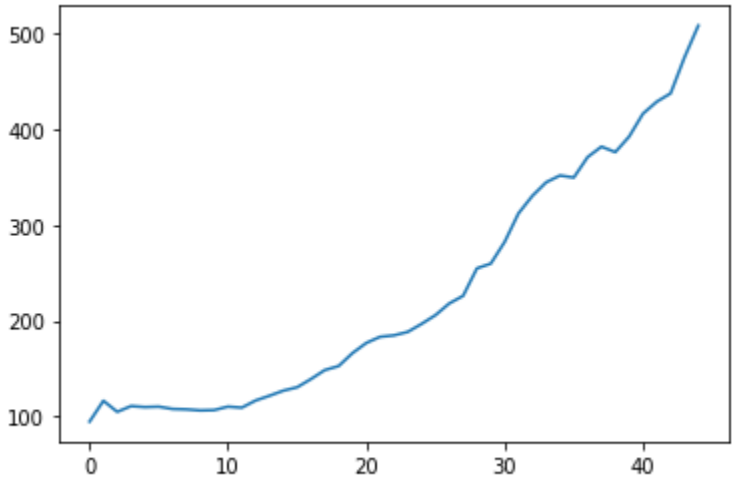

Figure 5.6: Steps taken in each episode while training

Chapter 6

Conclusion

In this paper,a novel framework for robot wandering based on RL is developed. The key contribution of this paper is proposing a customized reward function which is a function of robots action, distance from the objects, area explored and collision. Inorder to choose the correct combination of reward values various cases of experiments was done and analysed. The final experiments verified that the combination of these reward function makes the robot wander an area efficiently. This is verified by plotting of maps with the help of slam package in ROS. The trained robot has also shown high adaptability and generalisation ability in other test environments. algorithm based on DRL which takes the partial map as the input.

References

[1] Choi J, Lee G, Lee C. Reinforcement learning-based dynamic obstacle avoidance and integration of path planning. Intelligent Service Robotics. 2021 Nov;14(5):663-77.

[2] Fox D, Burgard W, Thrun S. The dynamic window approach to collision avoidance. IEEE Robotics Automation Magazine. 1997 Mar;4(1):23-33.

[3] 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.

[4] Fiorini P, Shiller Z. Motion planning in dynamic environments using velocity obstacles. The international journal of robotics research. 1998 Jul;17(7):760-72.

[5] Chen YF, Everett M, Liu M, How JP. Socially aware motion planning with deep reinforcement learning. In2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2017 Sep 24 (pp. 1343-1350). IEEE.

[6] Long P, Fan T, Liao X, Liu W, Zhang H, Pan J. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In2018 IEEE International Conference on Robotics and Automation (ICRA) 2018 May 21 (pp. 6252-6259). IEEE.

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

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

[9] Lapan M. Deep reinforcement learning hands-on. Packt publishing; 2020.

[10] Géron 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.

[11] 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.

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

[13] 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.

[14] Fan T, Long P, Liu W, Pan J. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. The International Journal of Robotics Research. 2020 Jun;39(7):856-92.

[15] Botteghi M, Khaled M, Sirmacek B, Poel M. Entropy-based exploration for mobile robot navigation: a learning-based approach. InPlanning and robotics workshop, PlanRob 2020.