#### 

**ENPM808F - Robot Learning**

Project – V

**Autonomous Navigation using Q-Learning for Indoor Environment**

Janakiraman Kirthivasan Rama Prashanth

UID: 116349086 UID: 116428941

13th May 2019

# Abstract

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Mobile Robot Navigation in an indoor environment which is an unstructured and uncertain environment, implementation of technique such as Simultaneous Localization and Mapping is not effective as it requires less uncertainty for its operation and also the memory usage and time complexity of using planning techniques or supervised machine learning are high. In this project navigation issue of controlling the robot in an unknown environment is being handled by using an efficient Reinforcement Learning approach called Q learning. This algorithm is implemented in Gazebo ROS to make Point Robot and Turtlebot learn the optimal policy to navigate a grid-based paradigm which is done through reinforcement signal that is received from the environment through which the state action pair for the entire environment is learnt. This approach is model free and ensures the system learn the shortest path between any random initial position and the goal position. Finally, the learned values can be seeded to a new environment from which the system can learn based on its configurations.

# Contents

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[Abstract ii](#_Toc8718339)

[Contents iii](#_Toc8718340)

[List of Figures iv](#_Toc8718341)

[CHAPTER 1 1](#_Toc8718342)

[Introduction to Mobile Robots and Q Learning 1](#_Toc8718343)

[1.1 Introduction 1](#_Toc8718344)

[1.2 Background 2](#_Toc8718345)

[CHAPTER 2 4](#_Toc8718346)

[Q Learning Approach and Implementation 4](#_Toc8718347)

[2.1 Approach 4](#_Toc8718348)

[2.2 Implementation 4](#_Toc8718349)

[2.2.1 Point Robot 4](#_Toc8718350)

[2.2.2 Turtlebot Controller 6](#_Toc8718351)

[CHAPTER 3 10](#_Toc8718352)

[Results and Analysis 10](#_Toc8718353)

[3.1 Trained Q Table 10](#_Toc8718354)

[3.2 Optimal Path being generated: 11](#_Toc8718355)

[3.3 Learning Rate vs Time 11](#_Toc8718356)

[3.4 Learning Rate vs Time for obstacle space 12](#_Toc8718357)

[3.5 Time vs number of obstacles 13](#_Toc8718358)

[CHAPTER 4 14](#_Toc8718359)

[Conclusion and Future Scope 14](#_Toc8718360)

[4.1 Conclusion 14](#_Toc8718361)

[4.2 Future Work 14](#_Toc8718362)

[Bibliography 15](#_Toc8718363)

[Appendix 16](#_Toc8718364)

# List of Figures

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[Figure 1 Q Function 2](#_Toc8706128)

[Figure 2 Q-learning cheat sheet 2](#_Toc8706129)

[Figure 3 Q-learning Algorithm 3](#_Toc8706130)

[Figure 4 System Architecture 4](#_Toc8706131)

[Figure 5 Point Robot Class Diagram 5](#_Toc8706132)

[Figure 6 moveRobot Flowchart 5](#_Toc8706133)

[Figure 7 TurtleBotClass 7](#_Toc8706134)

[Figure 8 Q learning Module Class 8](#_Toc8706135)

[Figure 9 System Architecture/ Q Learning Module 9](#_Toc8706136)

[Figure 10 Learning Rate vs Number Of Iterations and Learning Rate vs Q-Values for Free space 11](#_Toc8706137)

[Figure 11 Learning Rate vs Number Of Iterations and Learning Rate vs Q-Values for Obstacle Space 12](#_Toc8706138)

[Figure 12 Number of Iterations vs Number of Obstacles 12](#_Toc8706139)

# CHAPTER 1

# Introduction to Mobile Robots and Q Learning

## 1.1 Introduction

Mobile robots need to be capable of moving in any given environment and can leverage navigation

guidance from pre-defined route in a controlled space. By contrast, mobile robots can be attributed as autonomous which means they can navigate an uncontrolled space without relying on physical guidance devices. Owing to this, mobile robots have become widely used across commercial, industrial, hospitals, warehouses, military, security and even household settings. Mobile robots are usually defined by the environment space where they travel and the guidance devices used.

An autonomously guided mobile robot at the very least is expected to know where it is and various goal points it has to reach. In the former, localization used several means such as sensors to determine its current location. For the latter, positioning systems come into play to determine orientation in addition to location from which it can plan its next goal points to proceed. Commonly relative position, Monte-Carlo or Markov localization are used for the positioning. The important aspect of navigation lies on the capability of selecting a path and moving from current location to target destination. The challenge in these targets is that they are neither visible or known prior by any cues and controlling mobile robots in such scenarios are vital in unknown environments with high risks. Mobile robots are utilized in applications of these kind to reduce risks to humans and improve efficiency.

Given the advantages of the mobile robots offer, the challenges of venturing in uncertain and

unstructured environments have been addressed in variety of research and competitions most of which involve optimal planning strategies with considerable uncertainty. Many machine learning strategies have been developed with ability to learn complex problems by its experience of performing a task. In case of a supervised learning setup, we would essentially need huge amount of data which can be difficult to obtain and define. Among these practices, Reinforcement learning learns by several trial and error on an uncertain environment with states to actions mapping. Reinforcement learning approach emphasizes the problem of learning to perform action and maximizing it in an unknown environment. The goal here is to tune the environment so that its mapping of input and output minimizes the reinforcement.

## 1.2 Background

The purpose of using reinforcement learning is to observe if we can predict the values of each states

accurately through a model-free approach. To learn from trial and error, Temporal Difference, an agent learning through episodes with no prior knowledge of the environment is used to explore the state space and estimating with model-free approach.

Q-learning is a value-based reinforcement learning algorithm. Q-learning learns the function Q which tells how good to take an action at a particular state. In Q-learning we will store Q-values for all possible combinations of states and actions in Q-Table which is just a simple look up table to calculate the maximum expended future rewards for action at each state. Q-Table will guide us to the best action at each state where each Q-table score will be the maximum expected future reward for the robot when it takes that action at that particular state. This is iterated to improve the Q-Table at each iteration.

The Q-function uses the Bellman equation and takes state “s” and action “a” as inputs

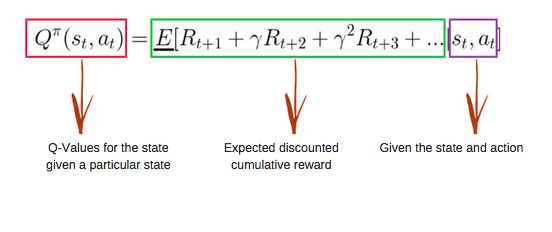


Figure 1 Q Function

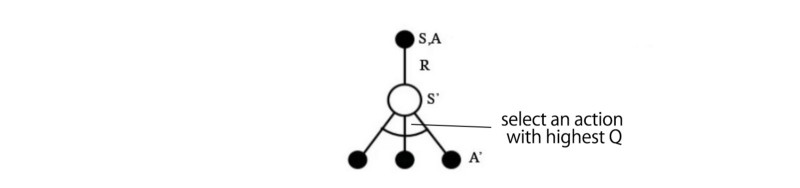


Figure 2 Q-learning cheat sheet

The Q-learning algorithm to fit Q with the sampled rewards is given below. When an action a is taken to see what reward R we get from state s to new state s’ which gives us a one-step look ahead. R + Q(s’,a’) becomes the target that we want Q(s,a) to be. As we change, we maintain a running average for Q and the values will get better and with some tuning, the Q values will converge.

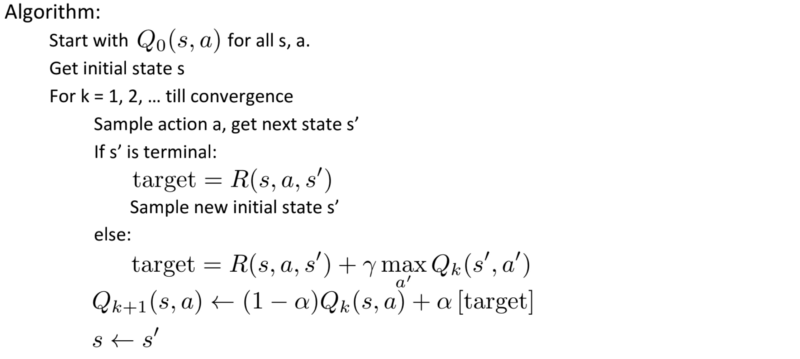


Figure 3 Q-learning Algorithm

# CHAPTER 2

# Q Learning Approach and Implementation

# **2.1 Approach**

In this project, the problem of finding an optimal path from any start point to a goal point is done based on Q learning. The environment is divided into grids. The position of the robot in a grid defines the state of the system. We define a heuristic for the reinforcement signal as the robot moves while it is being trained. With a trained Q table, the robot is able to find a path from any start point to a given goal point with obstacles in its environment.

# **2.2 Implementation**

The system implementation architecture is shown in the figure below.

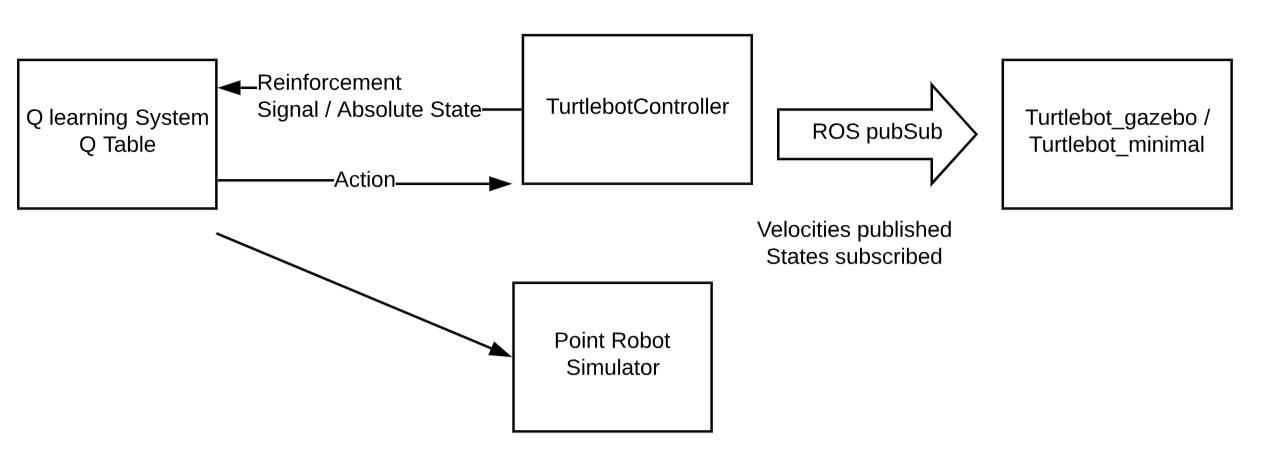


Figure 4 System Architecture

The system addresses two parts of the project:

1. Training of the Q learning system for a simulated point robot. This part of the project focuses on

building an optimal Q learning system and testing it with a simulated point robot.

2. Training of the Q learning system for a turtlebot. This phase of the project focussing on Q learning

with an actual robot. The implementation of this system is tested on gazebo ROS simulator

### 2.2.1 Point Robot

The function of the point robot is to take an action input from the q learning module and perform the action. It in turn makes use of its sensors to detect the obstacles and return the corresponding status of its action to the Q learning module. Through this system we are able to try the different Q learning parameters and choose optimal values. Below figure indicates the class level implementation of the point robot.

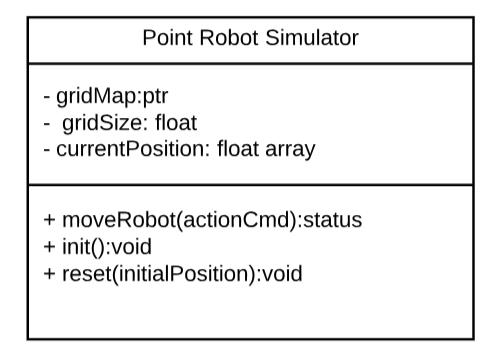


Figure 5 Point Robot Class Diagram

The important member variables and methods are explained in detail below:

1. Current Position: The class variable maintains the current position of the robot with respect to the

environment. This is an indicator of the state of the robot in the environment and gets updated in

the moveRobot method

2. Grid Map: This member variable holds the map of the system. This is used as a look up for the

point robot as it traverses in the environment. In the case of the turtlebot we get the presence of

obstacles in the environment through its sensors.

3. moveRobot: In this method the actual motion of the point robot is performed. The actions are

input as a 4 connected action space of top, left, right and down motion. The point robot looks up

into the grid map and in case the system is free of obstacles, it moves to the next position and

updates its current position. Based on the ag returned, an appropriate reinforcement signal is

applied.

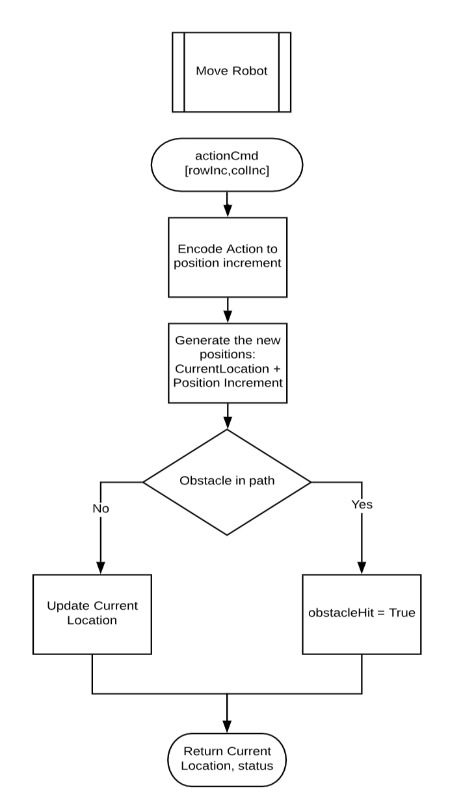


Figure 6 moveRobot Flowchart

4. resetPosition: This method is used to spawn the robot at a random location in the environment based on the start position input

### 2.2.2 Turtlebot Controller

The turtlebot controller performs the function of handling the turtlebot in the Gazebo simulator. This module interfaces the robot and the program under execution through ROS interface. The following sensor and actuator data is received from the robot:

1. Laser data: The laser data from the turtlebot provides the range of the obstacle. This sensor data is received by subscribing to the "/scan" ROS topic. Since the range data in the direction of the robot's heading is required for the case of a point robot, we make use of the laser range in the angle -10 deg to 10 deg. The maximum range that can be received is 50 m
2. Model States: The model states ROS topic contains the information of the models that are

spawned in the system along with the pose and orientation information. We make use of this topic to spawn the turtlebot in the simulation environment in different positions. The ground truth pose and orientation of the robot is obtained from this topic. It is a simulation of the case of an actual hardware in which the pose is obtained based on GPS, ODOM or any other sensor data

1. Velocity publisher: The vel\_cmd topic is used to publish the velocity data to the robot. It contains the linear velocity input in the x, y, z direction and angular velocity input in the x, y, z direction. The data is published using twist message. The rate at which data is published is defined in the ros rate

The main functionality of the controller is as given below:

1. Spawning and respawning the turtlebot at the end of each episode in the simulator environment
2. Rotate to the given orientation and navigate to the new position. A closed loop feedback is applied here. The angle of rotation and increment in position is determined based on the action input. In the rotation motion, we compare the increment in the angle with the current orientation and based on this an appropriate rotation is performed. Thereby achieving the required position. In case of increment in position, the pose of the robot is continuously monitored and compared with the target position. Based on this a linear velocity of 0.2ms is applied to the robot.
3. Continuously read the sensor data and provide inputs to the actuator

Below figure represents the class diagram of the turtlebot controller.

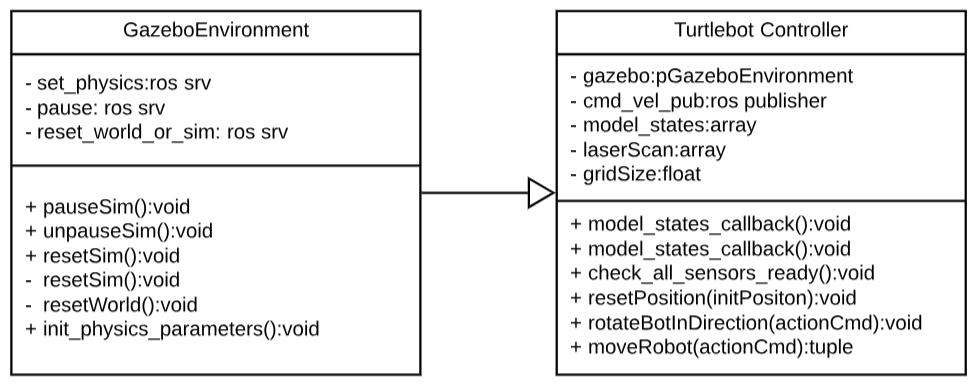


Figure 7 TurtleBotClass

The important class members and variables are detailed below:

**Q Learning Module** This module is central to the system. The module forms the Q learning

implementation of the system. Since the environment can be discretized into finite number of actions and state spaces, we make use of Q Table based Q learning approach.

The state of the system is defined by the x and y position of the robot. The position of the robot is

attributed to a grid position based on the grid size. The number of states in the system is determined

based on the following equation:

stateNumRows = maximumMapLocationXAxis=gridSizeAlongX

stateNumCols = maximumMapLocationAlongY Axis=gridSizeAlongY

TotalNumberofStates = stateNumRows \_ stateNumCols

The basic working of a Q learning system is as shown in the figure below. The system which is the point robot or the turtlebot in our case takes an action. The action results in a change in the environment which is reflected as the states. A reinforcement signal is applied to the system. The reinforcement signals acts as a supervisory information to the system. In each iteration the system takes an action thereby continuously. The goal is to reach the end position. The reinforcement signal is used to train the system. After a number of episodes for which the system is trained it is capable of achieving optimal policy that maximizes its reward of achieving a goal.

The reward signal that is applied in the q learning module is as shown below. The manner in which the systems behaviour is affected by the reinforcement signal is explained in the analysis section

1. Taking a step = -1

2. Hitting an obstacle = -5

3. Reaching a goal = 1000

Overall the functionality of the Q learning modules is as detailed below:

1. Generate actions based on Q Table. Make use of an epsilon greedy policy to generate the actions

2. Decodes the state based on position of the system.

3. Predict actions and update the Q values based on the reward signal obtained from the environment

The class level architecture of the Q learning module is shown in the figure below.

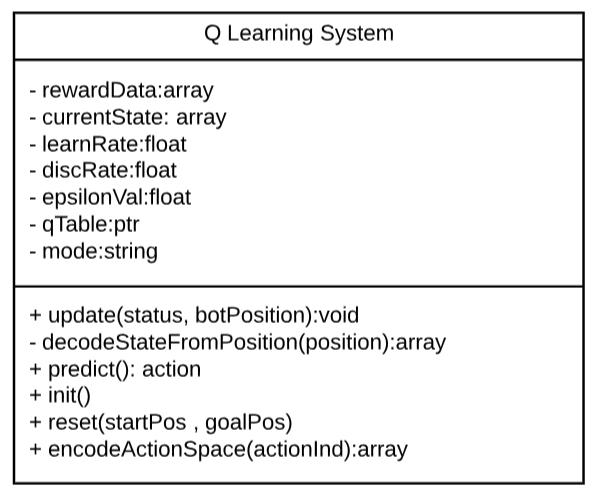


Figure 8 Q learning Module Class

The following methods and attributes are key for the system and are detailed in brief:

1. Member variables learnRate, epsilonVal, discRate: These are parameters of Q Learning. The

learning Rate defines the rate at which the system learns the optimal policy. A fine choice has to

be taken here to ensure that the system converges to the global optimum. Higher value of learning

rate results in divergence. discRate parameter determines the rate at which the future states and

actions influence the system epsilonVal determines the rate at which random actions are chosen.

This is required to ensure that the system explores and exploits. Higher the epsilon val greater is

the exploration

2. Q-Table: The size of the Q-Table is equal to number of states multiplied with the number of

actions. The Q-Table is updated with the *update()* function.

3. update(): The update function ensures that the Q-Table corresponding state for which an action

taken is updated. The update rule for the state action pair is given in the equation below:

q(s; a) = q(s; a) + α \* (rewardSignal + γ \* max(q(sn, :) - q(s; a)))

Here α represents the learning rate, γ represents the discount rate, sn represents the next state.

4. predict(): This function predicts the next action that is to be taken. In each iteration a random

number is chosen between 0 to 1. In case the random number is greater than the epsilon value,

then the action taken for the current state is the maximum of all the q values else a random action

is chosen for the current state.

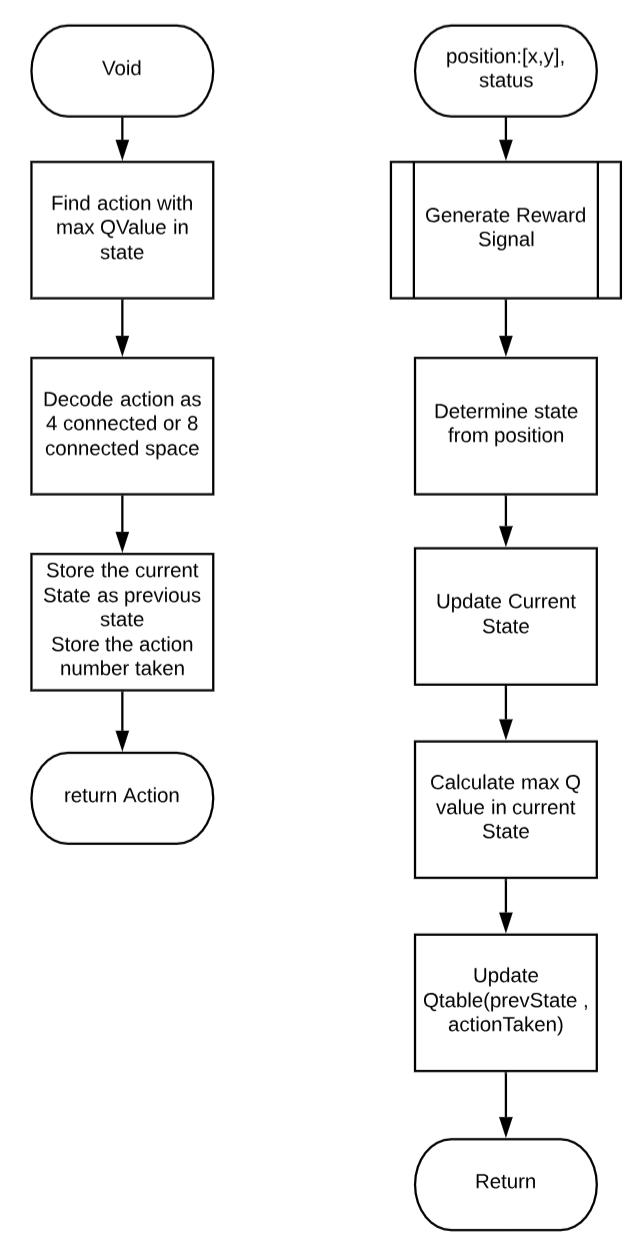


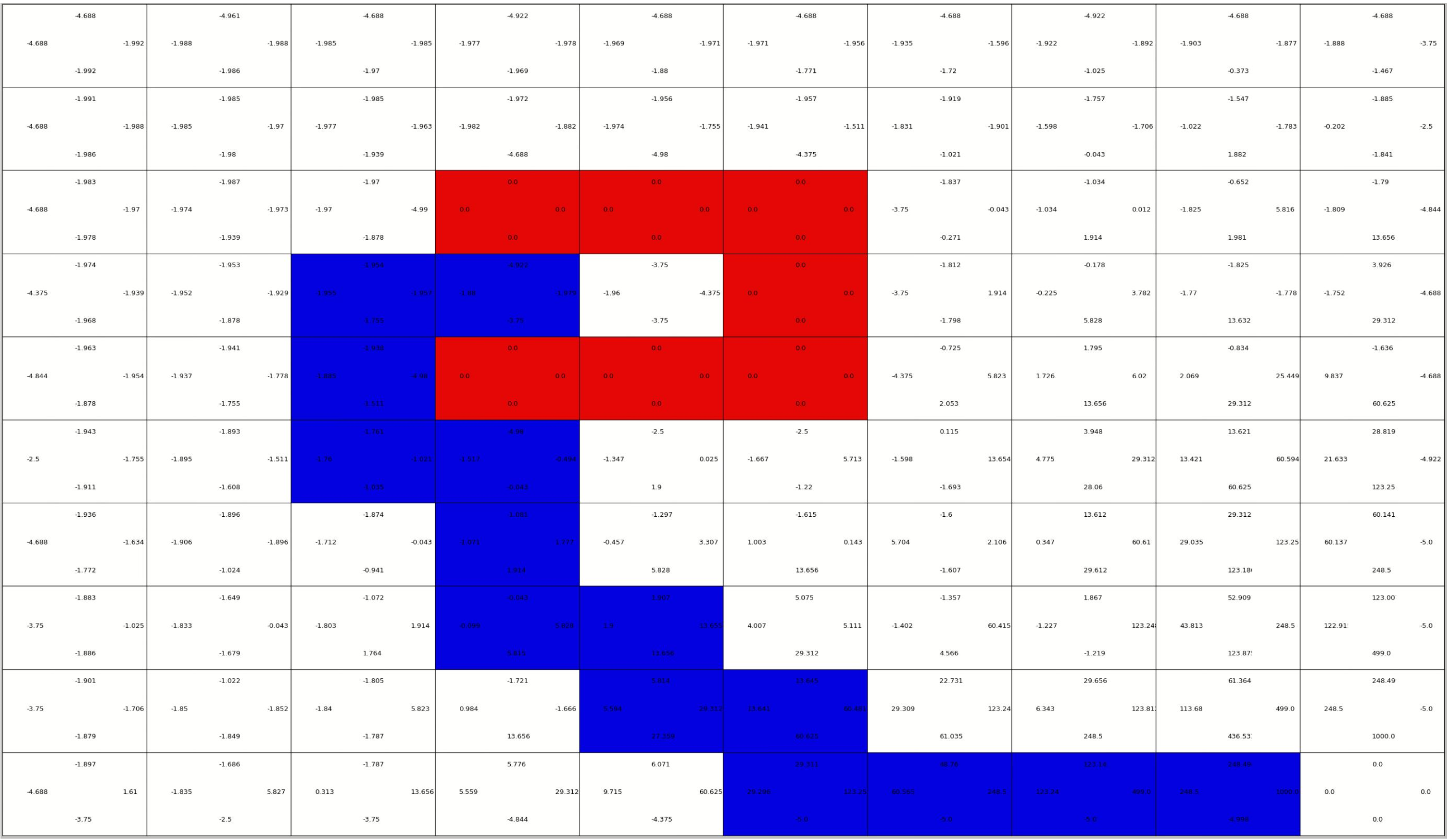
Figure 9 System Architecture/ Q Learning Module

# CHAPTER 3

# Results and Analysis

This section details the results that have been obtained for the system along with a detailed analysis

# **3.1 Trained Q Table**



The above table indicates the Q table after reaching saturation. Each grid represents a state of the system. The four Q values in each grid represents the actions for the corresponding state. Considering the grid 0,0. We observe that the top and the left Q values are -4.99. This is because of the presence of the walls of the environment detected as obstacles.

As the point robot tries to move towards the left or right direction it detects an obstacle and hence receives a negative reinforcement signal from the environment. Hence, we see a negative value.

We observe in the table that as we traverse from the grid 0,0 to 10,10. the Q-values increases. This is based on the iterative process in which the Q table gets updated. The process is explained below:

Initially the Q table is initialized with a value of 0. On each step that the robot takes it gets a negative reward of -1 from the Q-Learning module. According to the update rule the corresponding state, action entry based on calculation is -0.5

When the robot further continues its motion, the Q values are updated. On reaching the goal, the state from which the goal is reached gets a positive reinforcement of 100. For example, let us say if the goal state is 10,10 and the state previous to the goal is 10,9. Then the Q value of state 10,9 is calculated as below from the update rule:

State s1 = 10, 9, down action a 1 given by [0, 1]; reward r = 100

Let us say the initial q value of state is q(s1, a1) = 0:0, α = 0.5, γ = 0.5

q(s1, a1) = 0.0 + 0.5 \* (100 + 0.5 \* 0.0 – 0.0)

q(s1, a1) = 50

In the next episode, the state, action pair from which the updated state is achieved as a next state gets a positive q value. This is because the update rule depends on the max(q(sn, :)) which is positive. Hence it can be observed that the Q values gets propagated.

Therefore, as we move away from the goal we observe positive q values in decreasing order.

3.2 Optimal Path being generated: The figure below indicates the optimal path for two given initial position to the goal position. The path is generated based on the optimal policy generated from the Q Table values. In the prediction phase for a given state, the action with the maximum Q value is chosen.

Observing the Q values of the path, it is observed that the path with the max value is chosen in each state. It can also be observed that even in the presence of obstacles an optimal path is generated for a given initial position. This can be inferred from the fact that it is the shortest path from the given initial position to the goal position.

3.3 Learning Rate vs Time

The graph below indicates a plot of the number of iterations vs learning rate for convergence.

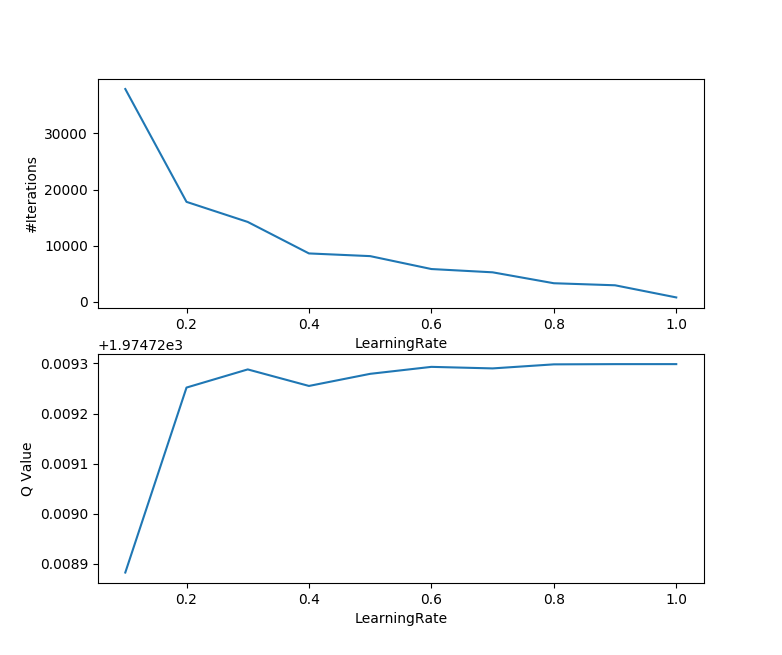


Figure 10 Learning Rate vs Number Of Iterations and Learning Rate vs Q-Values for Free space

It is observed that as the learning rate is increased, the number of iterations for convergence decreases.

The curve shows an exponential behaviour with a higher decay rate initially and slower decay at the end.

This is based on the analysis that with a lower learning rate, the update in the Q values is very small.

This can be attributed to the fact that the update rule of the q table depends on the learning rate. With

a smaller learning rate, the update is slower. Hence the number of episodes is higher for the q values to be updated continuously and for it to reach saturation.

As we increase the learning rate, the Q value update is relatively higher and therefore the decay is higher.

Increasing the learning rate results in a decreased convergence. This is because the update term contains a negative of the current Q value that decreases the update due to the reinforcement signal

3.4 Learning Rate vs Time for obstacle space

The graph below represents the learning rate vs time for obstacles in the grid.

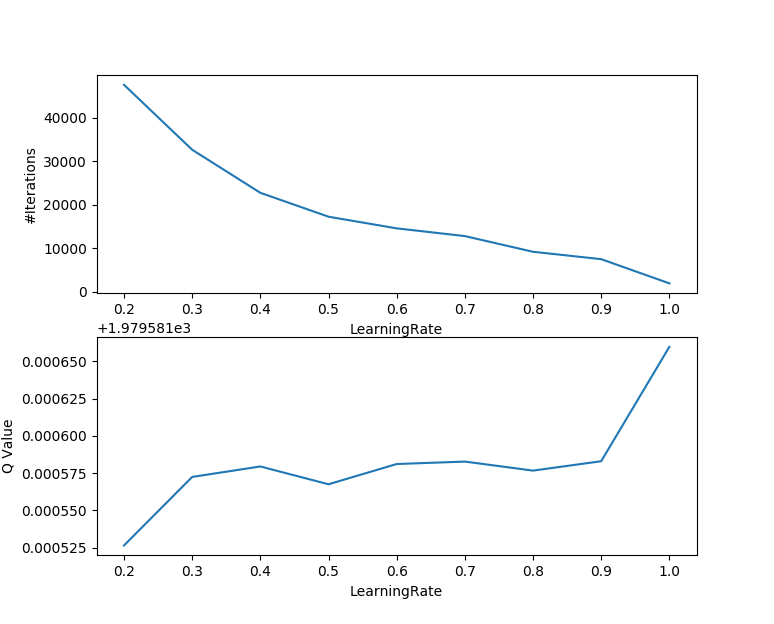


Figure 11 Learning Rate vs Number Of Iterations and Learning Rate vs Q-Values for Obstacle Space

It can be observed that the convergence is slower in case of adding obstacles as compared to the learning rate vs time for free space. This is because the presence of obstacle results in the interference of the Q values being propagated. Hence the values around the obstacle results in a negative reinforcement signal of -5 resulting in a greater number of iterations to update the Q table in the states around the obstacles.

## 3.5 Time vs number of obstacles

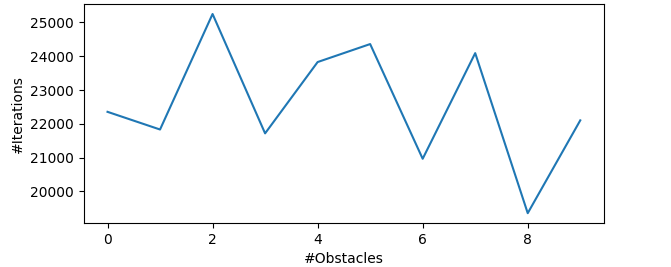


Figure 12 Number of Iterations vs Number of Obstacles

It is observed that the number of iterations for convergence increases with the number of obstacles. This is because as the obstacle space is increased, the number of states count around obstacles increases and needs a larger time for updating. Hence the number of iterations for convergence increases with number of obstacles.

# CHAPTER 4

# Conclusion and Future Scope

## 4.1 Conclusion

A given indoor environment is divided into grids and the optimal shortest path from any given position to a goal position for a point robot with 4 connected actions is determined. The system performs well with static obstacles in the environment.

Further, the behaviour of the learning rate vs time for free space and obstacle space is analysed and

verified.

## 4.2 Future Work

With the following results, the following work is to be planned to be done in the future:

1. Q learning applied to a system with dynamic obstacle: Most environments are dynamic in nature with moving objects. Much of the research is ongoing in this field. For example, in the paper on socially aware motion planning published by MIT, a robot moves in the environment based on observation of human motion patterns. These research studies can be further extended to apply navigation in environments that are dynamic and follows pattern.
2. Deep Q learning Network: Finding the shortest path with deep Q learning is another field of interest. This is because the system will have a more flexibility with the number of actions and states thereby increasing the performance of the system
3. Application of path planning using reinforcement learning for differential drive robot. Using a differential drive robot helps in terms of performance and reduction in time.

# Bibliography

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* D. Tamilselvi, Dr. S. Mercy Shalinie and G. Nirmala, “*Q Learning for Mobile Robot Navigation in Indoor Environment”*. IEEE Conference, 3-5June 2011
* Sven Koenig and Reid G. Simmons, *“The Effect of Representation and Knowledge on Goal-Directed Exploration with Reinforcement-Learning Algorithms\*”,* 1 May 1991
* <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=5972477tag=1>
* Sebastian Thrun, *“A Lifelong Learning Perspective for Mobile Robot Control”* Proceedings of the IEEE/RSJ/GI Conference on Intelligent Robots and Systems, 1994
* James L Crowley *“Navigation for an Intelligent Mobile Robot”*, August 1984
* Leslie Pack Kaelbling, Michael L Littman and Andrew W. Moore, *“Reinforcement Learning: A Survey”*, Journal of Artificial Intelligence Research, 1996
* Richard S. Sutton and Andrew G. Barto, *“Reinforcement Learning: An Introduction”*

Second edition, MIT Press Cambridge, Massachusetts London, England, 2017

# Appendix

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Project Code

<https://github.com/jkvasan7692/qlearning_turtlebot_indoor.git>

* Results, Videos, Presentation <https://drive.google.com/file/d/1D3KduBAaf2A91FTo4dkIjdEZhyq9VS_f/view?usp=sharing>