CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Robotics is a versatile and emerging area influencing all our lives. The ability of robots to work without rest will replace human beings at work. An autonomous robot should have the capability to understand the characteristics of its surrounding and should be able to inherit necessary information from the environment which enables them to be user friendly. After gaining the essential data from the surrounding, it needs to trace its path to its destination. This is called path planning. Some of these robots are shown in fig 1.1.

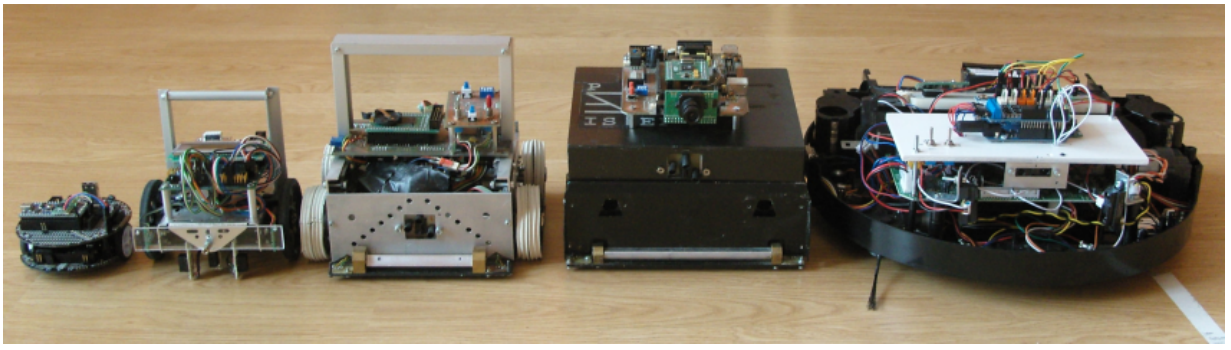


Fig 1.1: Mobile Robots

Path planning is a necessity for autonomous robots which helps these robots to track their path from the source grid to the destination grid as shown in fig 1.2. Numerous path planning algorithms have been proposed so far. The selection of path planning algorithm depends on its application. There are many factors or requirements an application needs like some might need less time of computation to complete the process where as some others focus on less memory requirement or travelling through the optimal path. According to these factors path planning algorithms are selected which satisfies these requirements.

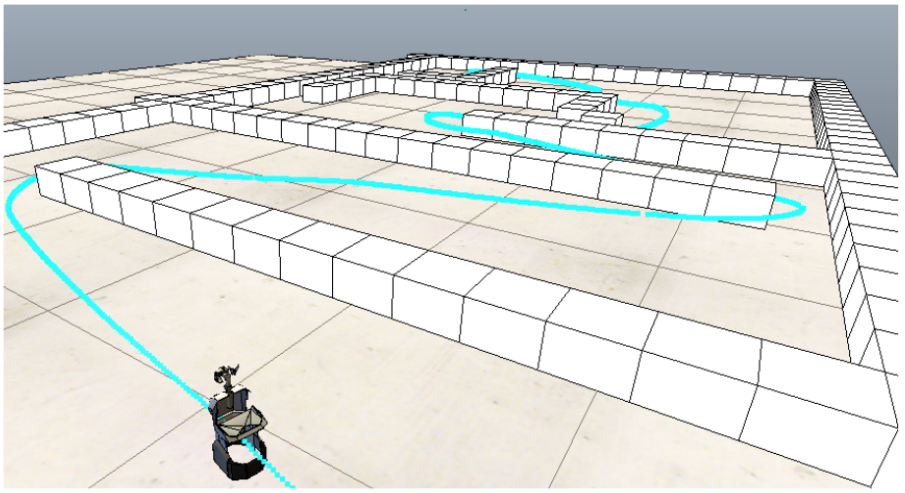


Fig 1.2: Path planning

There are a lot of path planning algorithms implemented in robotics for all kinds of obstacles in literature. Robotic path planning can be done by various techniques. It has now stirred great interest in the research community.

1.2 OBJECTIVE

The main objective of this project is to find the shortest path between source and destination with optimal path planning using SARSA which is a reinforcement learning in an indoor environment for static obstacles. This will be simulated using MATLAB. The hardware implementation of this algorithm is done using iRobot Create interfaced to NXP LPC1768 Cortex M3 controller.

1.3 MOTIVATION

Reinforcement Learning is an algorithm which gathers the required data from its surrounding according to its application and updates these data with time so that any changes in the

surrounding can be noted and taken into consideration during execution of the algorithm. This is the main advantage of using reinforcement learning. The objective of the algorithm will always be to rise up its performance rate which gives added bonus compared to the other conventional algorithms. The algorithm always learns and then executes.

1.4 ORGANIZATION OF THE REPORT

This chapter of the report gives the brief introduction to various aspects of path planning and reinforcement learning. The background study for the project is presented in Chapter 2. An introduction to Reinforcement learning is given in Chapter 3. Chapter 4 explains about the system overview and methodology. It also gives a brief idea about the different hardware components used. Chapter 5 provides the complete information about the implementation of work done and also reflects the timeline of the project. The report is then concluded in Chapter 6

1.5 CONCLUSION

This chapter gives a brief introduction to reinforcement learning based path planning. It also discusses about the main motivation which led to the choice of the work. Finally, it gives a description of how the report is organized.

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

In order to know about the different work done in the field of path planning, reinforcement learning and SARSA algorithm, a thorough background study is necessary. This chapter provides an overview about the works carried in the related field. Papers related to different algorithms for path planning and implemented applications of reinforcement learning and SARSA algorithm are mentioned in this chapter.

2.2 LITERATURE SURVEY

A robot that is fully automated should be able to extract necessary information from the environment in different ways depending on the implemented application without the intercession of humans. Designing of such robots that are self- governing in amorphous, dynamic and for onerous environments still remain a challenge [2]. All autonomous robots reach the end point through simultaneous localization, mapping and path planning taking the most advantageous path [3] [11]. Provision of topological map can be utilized to path finding [8]. A variety of path planning algorithms have been designed with time which help these robots to find the ideal path [1].

According to the type of environment in which robot is installed, path planning will be done either for static or dynamic obstacles and for known or unknown environment [9][28]. Path planning algorithm that desires lesser time with multiple robots can aim ant colony optimization algorithm [4]. Machine learning algorithms are also used for path planning like temporal

difference algorithm which designates reward according to the direction of movement [6]. Path planning algorithm with fuzzy methods are put in action where the accuracy can be increased with more inputs [5].

Real time employment application with unceasing updates from the surroundings can use reinforcement learning algorithm, which is a type of machine learning algorithm and also a branch of artificial intelligence [9][16]. Reinforcement learning can be of two types. They are model based RL and model free RL. Awareness about the environment will be provided in model based RL whereas no information will be given to model free RL [1]. The primary concept of RL algorithm is communication with environment through learning [10]. Here the robots regularly balance the actions in order to obtain the ideal movement [28] by increasing its performance [27]. It is found through trial and error method [17][26].

There are different types of RL path planning algorithm from which SARSA algorithm is studied and modified here. This learning algorithm is an on-policy algorithm. It has a wide range of application and has proved its performance in various fields. It was used in swarm RL algorithm in which numerous robots communicate with each other in order to interchange information [12]. In urban areas it is used to control traffic which is self-adaptive [13]. It is also used in power systems to determine the optimal power in these systems [15]. It is used for channel allocation in cellular networks which permits call blocking [14]. SARSA algorithm is also an approach of learning markov decision process (MDP) [16][18].

Collision free path planning has always been a constraint when both static and dynamic obstacles are present in the environment. In an environment where both unknown and moving obstacles are present fuzzy based path planning is used [19]. D* lite path planner is also used for path planning with dynamic obstacles in a haphazardly partitioned environment [20]. The

occurrences of a static obstacles are found with the help of ultrasonic sensors interfaced with a microcontroller in AGO algorithm [4] and fuzzy algorithm [5].

Designing of robots which are capable of moving from altering source and destination points is emerging with time. In [21], a prototype version of customized shopping system was built where the information about the destination point is communicated through mobile applications by the customers and the products will be distributed by the distribution centers. Android applications are used to get voice and gesture commands for smart robot assistant applications [22]. Android apps are easily created using applications like app inventor [23] and app inventor2 [24] which is an open source tool [25]. These android apps make it easy to communicates the position of obstacles to the robots.

2.3 CONCLUSION

In this chapter an overview of the background and the work done in the related field has been provided. An insight of the vast amount of work done in the areas like path planning, reinforcement learning and SARSA algorithm has been given in this chapter. Path planning, reinforcement learning and SARSA algorithm is analyzed with minimal literature.

CHAPTER 3

REINFORCEMENT LEARNING

3.1 INTRODUCTION

This chapter provides a brief introduction about Reinforcement learning and the type of RL learning algorithms used for path planning.

3.2 REINFORCEMENT LEARNING

Reinforcement Learning comes under Machine Learning and an offshoot of AI. The robot will study the environment and will obtain a reward for its action as a feedback. It obtains both positive and negative feedback. Positive feedback is obtained for taking the correct action like traversing through the optimal path and negative feedback is obtained for taking the wrong action like hitting an obstacle. The system punishes the robot for taking the wrong action by giving negative reward and rewarded with more points for taking the right action. The signal that robot obtains from the environment is called reinforcement signal.

The system should learn at least once before it starts its execution. Once it learns it can periodically update the data. If the issues involved in the system are designed with care, it can reach global optimum and can be modelled by a human expert who has a little knowledge

about the current field and can be set up with lesser time and with less complex sets of equations compared to other algorithms.

There are four divisions of reinforcement learning algorithms. They are Q Learning, SARSA, Temporal Difference Learning algorithm and Deep Q Network algorithm. All these algorithms are explained in detail in the coming sections.

3.2.1 SARSA ALGORITHM

State–action–reward–state–action (SARSA) is an algorithm which learns according to the current policy and not greedy policy like most of the other RL algorithm. It always tries to takes the safer route rather than the optimal path. It acts greedily eighty percent of the time and randomly twenty percent of the time. As the name suggests the main factors that affects the equation of SARSA algorithm for upgrading the Q-value rely on "S_1" which is the current position of the robot, "A_1" is the action taken by the robot, "R" is the reward obtained for the particular action , the state "S_2" that the agent enters after taking that action, and finally the next action "A_2" the agent choose in its new state. The acronym for the quintuple (s_t, a_t, r_t, s_{t+1}, a_{t+1}) is SARSA. It is an ON-policy algorithm. It's better to be aware of the consequences of exploration as it happens, and avoid outcomes that are too costly while acting, rather than looking for the true optimal policy. Instead of looking for the best action at every step, it evaluates the actions suggested by the current policy. It uses this information to revise the algorithm of SARSA is as shown in fig 3.1.

begin
initialize $Q[S, A]$ arbitrarily
observe current state s
select action a using a policy based on Q
repeat forever:
carry out an action a
observe reward r and state s'
select action a' using a policy based on Q
$Q[s, a] \leftarrow Q[s, a] + \alpha\,(r + \gamma Q[s', a'] - Q[s, a])$
$s \leftarrow s'$;
$a \leftarrow a'$;
end-repeat
end

Fig 3.1: SARSA Algorithm

3.2.2 TEMPORAL DIFFERENCE

Temporal difference learning is a method of Reinforcement learning algorithm. Here it takes the action according to the highest reward awarded to it. There are mainly three possible movement in path planning. They are diagonal, vertical and horizontal movement, so the maximum reward will be given for diagonal movement. Here it will always try to move diagonally if there is no obstacle in its path. It doesn't explore the environment. It works according to the reward system and the generated Q value which remains constant and will only be altered if a dynamic obstacle is encountered. There will be many exploratory restricted areas in some applications, TD algorithm will ignore all these when it comes to taking the optimal path. So, it is more concentrated on taking the best path and not on its consequences. Fig.3.2. shows the structure of TD Learning based path planning. The Q factor value is calculated for every possible direction from each state and provided with inputs like starting point, destination

point and obstacle position. Every action obtains a reward, it can be positive or negative. The action selector selects the next action as shown in fig 3.2.

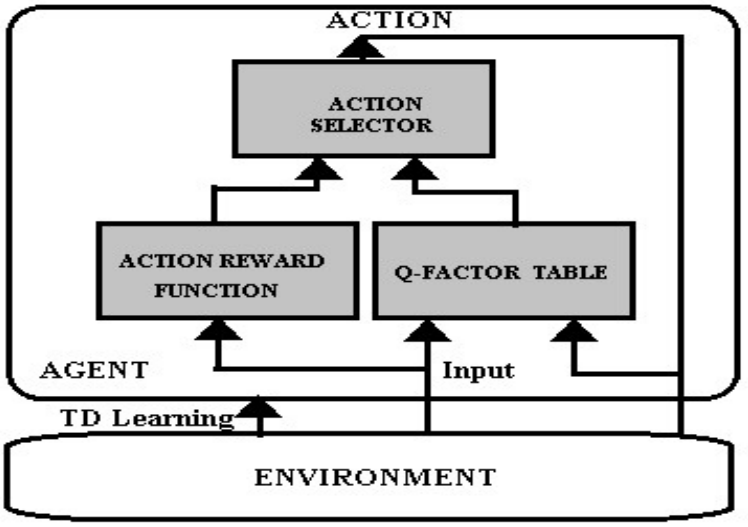


Fig 3.2: Structure of TD Learning based path planning

3.2.3 Q LEARNING ALGORITHM

Q Learning is a type of reinforcement learning algorithm where the agent takes action according to greedy policy but after training the information. It interacts with the environment to formulate the Q value through multiple episodes and then takes the action greedily. It learns about the environment through episodes and takes the optimal path even if it's an exploratory restricted area. Getting information from the environment helps the algorithm to update its data if some unmentioned changes have occurred. It's similar to TD algorithm but there is no training of data in TD algorithm which makes Q leaning a better algorithm

3.3 DEEP Q NETWORK

Deep Q Network is a combination of Q Learning algorithm and Neural Networks. It is better than the above-mentioned algorithms because all the other RL algorithm store their data in Q

table and their sizes increase rapidly as their size of grid increases. They have more memory requirement; more time consumption and larger tables which makes it complicated.

The inputs to the neural networks will be the current state and the outputs of the neural network will be the q values of possible actions from the current state. The q value obtained will be compared with target q value which is calculated and updated using the second neural network. According to the loss obtained after comparison the weights of the neural network are updated with the learning rate. This process is continued for the number of episodes defined in the algorithm. There are mainly two factors in Deep Q Network, they are fixed targets and experience relay. Experience relay is the data i.e. the Q value, which is stored to compare with the target Q value which is fixed which is called the fixed targets which will be updated with time. The algorithm of Deep Q Network is as shown in fig 3.3.

1. Initialize replay memory capacity.
2. Initialize the network with random weights.
3. *For each episode:*
    1. Initialize the starting state.
    2. *For each time step:*
        1. Select an action.
            - *Via exploration or exploitation*
        2. Execute selected action in an emulator.
        3. Observe reward and next state.
        4. Store experience in replay memory.
        5. Sample random batch from replay memory.
        6. Preprocess states from batch.
        7. Pass batch of preprocessed states to policy network.
        8. Calculate loss between output Q-values and target Q-values.
            - Requires a second pass to the network for the next state
        9. Gradient descent updates weights in the policy network to minimize loss.

Fig 3.3: Deep Q Network Algorithm

3.3 CONCLUSION

An introduction to the Reinforcement learning concept has been given in this chapter. An overview of the types of Reinforcement learning algorithms has also been provided here. Also,

the structure of modified TD Learning Learning, Deep Q Learning and SARSA algorithms-based path planning is also discussed in this section.

CHAPTER 4

SYSTEM OVERVIEW AND METHODOLOGY

4.1 INTRODUCTION

The methodology and system overview of the project is discussed in this chapter and has also explained about the hardware and software components.

4.2 SYSTEM OVERVIEW

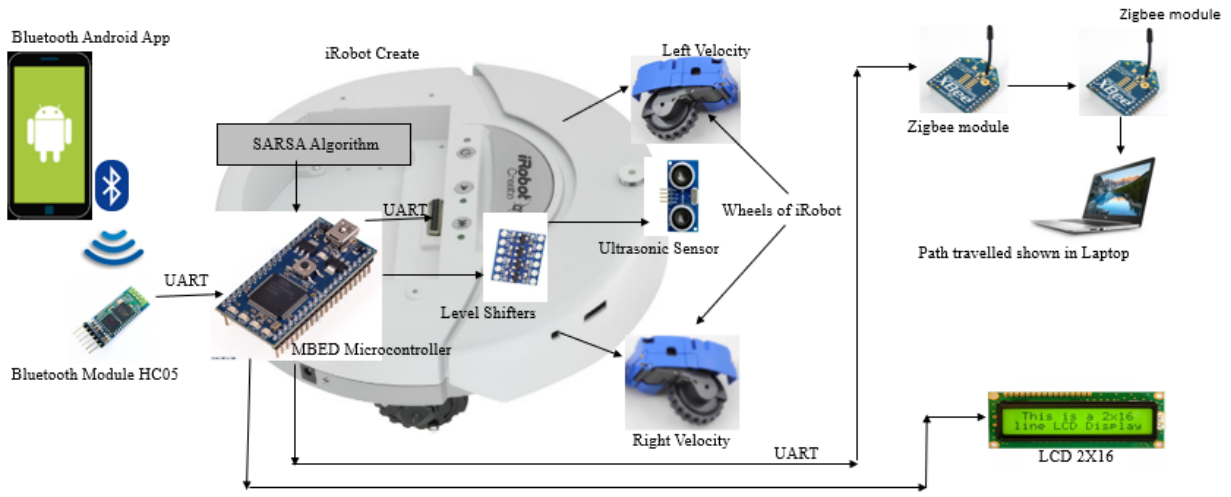


Fig 4.1: Block Diagram

Fig 4.1 shows the block diagram of the hardware part of the project. iRobot is used to implement reinforcement learning based path planning. An android based Bluetooth app is created for communicating the number of obstacles and the position of obstacles. Bluetooth app communicates to Mbed microcontroller. The reinforcement learning algorithm is implemented in the Mbed microcontroller for path planning. Then according to the algorithm, the right and left wheels of the iRobot is controlled. After the robot reaches the destination, an LCD is used to display a message stating that the robot has reached the destination. It also displays the time taken to travel and the distance travelled by the robot. After the robot reaches the destination, the path travelled by the robot is communicated to a pc through ZigBee communication to display the path travelled in a GUI using processing tool.

4.3 METHODOLOGY

The various steps involved in implementing path planning using reinforcement learning is discussed in this section

4.3.1 Implementation of A star, Dijkstra and TD algorithm in MATLAB

For the better understanding of RL algorithm, firstly A star, Dijkstra and temporal difference algorithm is studied and simulated. This algorithm was simulated in MATLAB

4.3.2 Comparison between A star, Dijkstra and TD algorithm in MATLAB

Based on the time of computation A star, Dijkstra and TD algorithms are compared and is found that TD algorithm does the computation with less time.

4.3.3 Implementation of SARSA algorithm in MATLAB

After understanding the basic algorithms, SARSA algorithm for an 8x8 grid is simulated and studied for variable sources and destination.

4.3.4 Comparison between SARSA and TD in MATLAB

SARSA algorithm has better success rate compared to that of TD. There are cases where SARSA algorithm works perfectly wherein TD fails.

4.3.5 Development of android based Bluetooth app

An android based Bluetooth app is developed for communicating the number of obstacles and the position of obstacles to the Mbed microcontroller.

4.3.6 Study of iRobot

The commands of iRobot and the basics of iRobot is studied for programming iRobot according to the algorithm.

4.3.7 Coding of SARSA algorithm in Mbed microcontroller

SARSA algorithm is coding in Mbed microcontroller and the shortest path from the source to destination is found out and then the iRobot is controlled to move from the source point to the destination point.

4.3.8 Interfacing Ultrasonic sensor to Mbed to detect dynamic obstacles

An ultrasonic sensor is interfaced with Mbed microcontroller to detect the presence of dynamic obstacles. When a dynamic obstacle is detected it should deviate from the present path and take the optimal path to its destination.

4.3.9 Interfacing LCD to Mbed microcontroller

LCD is interfaced with Mbed microcontroller to display a message after the robot reaches the destination. Also, the time taken and distance travelled is displayed on it.

4.3.10 Creating a GUI using ZigBee communication

Zigbee modules are configured and interfaced with Mbed microcontroller to communicate to a pc. In pc, a GUI is created using the processing tool. The GUI displays the path travelled by the robot from the source point to the destination point.

4.4 HARDWARE COMPONENTS

4.4.1 iRobot Create

iRobot Create is a vacuum cleaning commercial robot which can be controller using a remote, or serially or through a microcontroller. It is controlled using a microcontroller through the cargo bay connector pins (DB- 25 port) which provides serial communication, digital input & output, analog input & output, and an electric power supply. It can be

controlled to move a certain distance, turn at a particular angle, detect an obstacle and so on. These are done using iRobot commands. There is set of commands already available that needs to be sent serially through uart communication to the iRobot to control it. iRobot is as shown in fig 4.2.

Fig 4.2: iRobot Create

4.4.2 Mbed LPC1768

The Mbed Microcontroller is an ARM processor as shown in fig 4.3 is the main microcontroller used in the project. It is easy to use and program. It is a 40-pin microcontroller in a 40-pin 0.1" pitch DIP form- factor. It is used on breadboards, stripboards and PCBs. It supports lots of interfaces including USB, SPI, I2C CAN, ethernet, and serial. Here it is mainly used to get data from an app, control the iRobot and also for implementing the path planning algorithm.

Fig 4.3: Mbed LPC1768

4.4.3 Bluetooth module HC 05

HC-05 is a Bluetooth module which is used for wireless communication. It can be configured as both master and slave. It can be used to communicate between two microcontrollers, or between a pc and a microcontroller or to communicate with an app and a microcontroller and so on. It is easy to used and configure. The is interfaced through UART communication with a microcontroller. Here it is used to communicate the number of obstacles and the position of obstacles to the Mbed microcontroller through an APP. It has range up to <100m. Fig 4.4 represents the Bluetooth module HC 05.

Fig 4.4: Bluetooth module HC 05

4.4.4 Node MCU

NodeMCU is an open source IoT platform which is shown in fig 4.5. In the project it is used to upload values in thinkspeak.com. It uploads a 1 when the robot reaches the destination and a zero when the robot is not in its destination. It helps the user to update the position of the robot even when they are not physically available in the location. NodeMCU does its task after it gets an indication from Mbed microcontroller.

Fig 4.5: Node MCU

4.4.5 LCD

A liquid-crystal display (LCD) is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals shown in fig 4.6. It is mainly used to display some necessary information after the robot reaches the destination. It displays the time taken for the travel in seconds, the distance travelled by the robot in cm and a message indicating that the robot has reached its destination. The LCD is interfaced with Mbed microcontroller and is placed in iRobot.

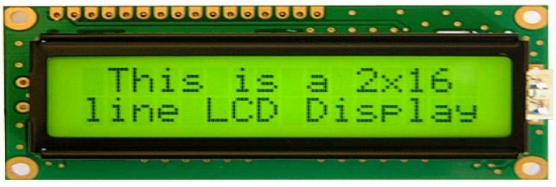


Fig 4.6: LCD

4.4.6 Ultrasonic Sensor

Ultrasonic transducers or ultrasonic sensors are a type of acoustic sensor divided into three broad categories: transmitters, receivers and transceivers shown in fig 4.7. Ultrasonic sensor is used in the project to detect dynamic obstacles that comes in its path. It is interfaced with Mbed

microcontroller and it is placed in front of the iRobot to detect the unnamed obstacles coming its way with any velocity. Here the velocity of these unnamed obstacles is random.

Fig 4 .7: Ultrasonic Sensor

4.4.7 ZigBee Module

ZigBee is a low-power, low data rate, and close proximity wireless ad hoc network as shown in fig 4.8. It is used to communicate the path travelled by the robot to pc to generate a GUI showing the path travelled by the robot. The ZigBee modules are configured as master and slave. Then one of the ZigBee modules is interfaced with Mbed microcontroller and the other ZigBee module is connected with the pc where the GUI is created. The receiver ZigBee module transmitted the data serially to a processing tool which converts the data into the path travelled and displays in GUI.

Fig 4.8: Zigbee Module

4.4.8 ESP8266 Wi-Fi Module

The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability as shown in fig 4.9. It is used to communicate with two Mbed microcontroller in robot to robot communication. The ESP modules will be set as servers with different port number so that both the modules can identify each other and communicate properly. Both the modules will be communicating to each other when it reaches near a junction and when the robot is in the junction to avoid collision of both the robots.

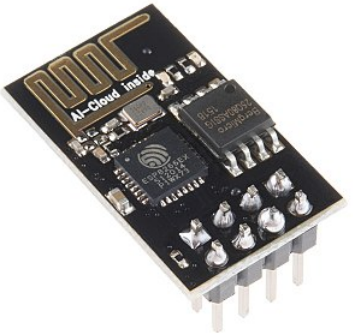


Fig 4.9: ESP8266 Wi-Fi Module

4.5 SOFTWARE COMPONENTS

4.5.1 MATLAB R2017a

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. Symbol of MATLAB is as shown in fig 4.10. It is used to simulate the various reinforcement-based path planning algorithms. Temporal Difference Learning and SARSA algorithm are implemented for different sizes of grid and also SARSA algorithm to solve complex mazes with a greater number of obstacles and they are compared with A* and Dijkstra algorithm based on the time of computation.

Fig 4.10: MATLAB R2017a

4.5.2 Mbed Online Compiler

This is the handbook for Mbed. With the Arm Mbed ecosystem, the work can be done online, in addition to offline. It is used to program the Mbed microcontroller. It generates a bin file that can be directly copied to the Mbed microcontroller. It is simple and easy to work. Mbed Online Compiler is shown in fig 4.11.

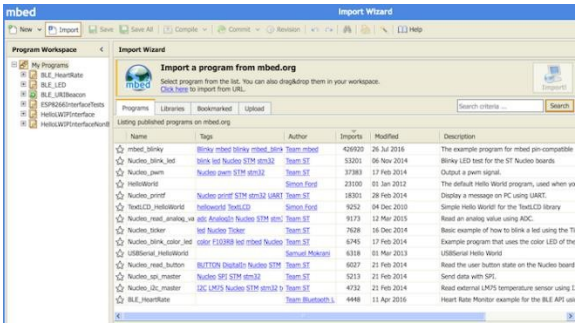


Fig 4.11: Mbed Online Compiler

4.5.3 MIT App Inventor

App Inventor for Android is an open-source web application originally provided by Google, and now maintained by the Massachusetts Institute of Technology (MIT). It is used to make android based Bluetooth apps that communicates through Bluetooth. It is easy to use and doesn't require any programming. The function of Each cell is defined using blocks. In the project, MIT is used to create an app that can communicate the number of obstacles and the position of obstacles to Mbed microcontroller through Bluetooth communication. The Mbed microcontroller receives the data using HC05 module. MIT app inventor is as shown in fig 4.12.

Fig 4.12: MIT App Inventor

4.5.4 Arduino Ide

The Arduino integrated development environment is a cross-platform application that is written in the programming language Java as shown in fig 4.13. In the project it is used to write codes for NodeMCU for incorporating IoT concept and to upload these codes to NodeMCU. It is used to upload values into thinkspeak.com.

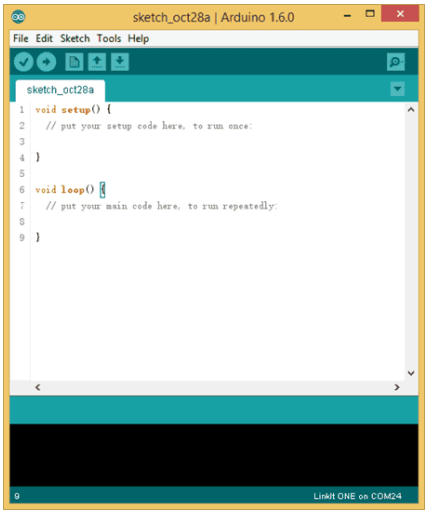


Fig 4.13: Arduino IDE

4.5.5 Processing Tool

Processing is a flexible software sketchbook and a language for learning how to code within the context of the visual arts. The tool is shown in fig 4.14. It is used to create GUI which shows the path travelled by the robot from the source point to destination point. The path is communicated through ZigBee communication from Mbed microcontroller and taken serially from receiver module to processing tool.

Fig 4.14: Processing tool

4.5.6 XCTU

XCTU is a free multi-platform application designed to enable developers to interact with Digi RF modules through a simple-to-use graphical interface as shown in fig 4.15. It is designed to configure the ZigBee modules as master and slave. Zigbee communication between the ZigBee modules can also be tested using XCTU by transmitting and receiving data from each other.

Fig 4.15: XCTU

4.5.7 PYTHON IDLE

IDLE (short for integrated development environment or integrated development and learning environment) is an integrated development environment for python, which has been bundled with the default implementation of the language and is shown in fig 4.16. It can be used to implement machine learning algorithms. In the project it is used to implement three algorithms namely Q learning, SARSA and Deep Q Network.

Fig 4.16: Python IDLE

4.6 CONCLUSION

In this chapter an overview about the system and methodology is explained. Also, the hardware and software components are mentioned. With more literature survey, methodology and system overview might be changed with time.

CHAPTER 5

IMPLEMENTATION RESULTS

5.1 INTRODUCTION

This chapter will give the information about the amount of work done till date. Also, the timeline of the project is shown in the end of this chapter.

5.2 IMPLEMENATATION OF TD ALGORITHM FOR A 4X4 GRID

In this section temporal difference learning algorithm is studied. It also contains the different assumptions made. Finally, the algorithm is simulated.

5.2.1 System layout for a 4X4 grid

The system layout and some of the obstacle occurrence scenarios are shown in fig 5.1 and fig 5.2 respectively.

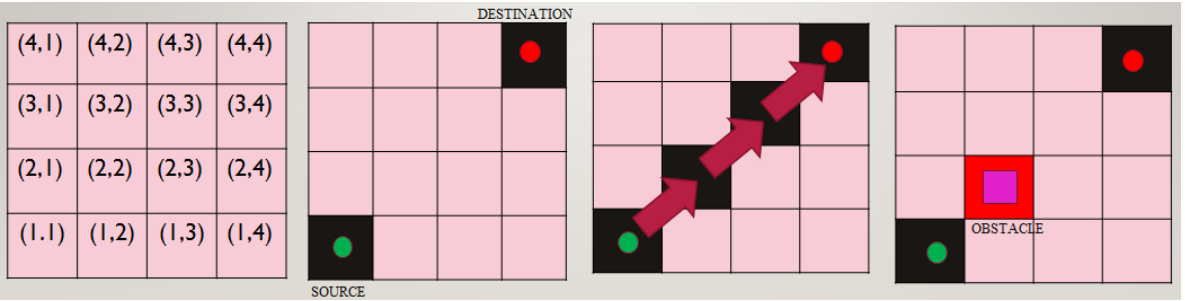


Fig 5.1: System Layout

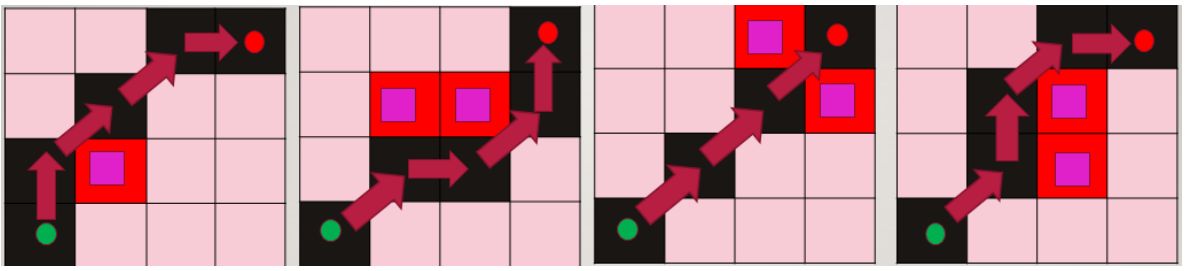


Fig 5.2: Some of the obstacle scenarios

5.2.2 Assumptions Made

There are some assumptions made before implementing TD learning algorithm and they are:

1. The size of the grid is 4x4 i.e. 16 blocks
2. Obstacles will be placed at the center of the blocks
3. Destination point can be altered
4. The robot doesn't have the ability to move backwards

5.2.3 Algorithm

TD learning algorithm is as described below:

1) The robot starts from the starting point.

2) Calculate the utility factor for each cell in a 4 x 4 grid using Q Learning equation $Q(S,a)=r(s,a)+\gamma max(Q(S',a'),$ where $Q(S, a)$ is the estimated utility function, $r(s, a)$ is the immediate

reward, ϒ is the relative value of delayed versus immediate reward, is utility factor for the resulting state *S'* after the action *a.*

3) A reward function is assigned for all possible movements. First preference assigned to diagonally forward movement, next priority given to vertically forward movement and last precedence assigned to horizontally forward.

4) Add the utility factor and corresponding reward function to evaluate the modified utility factor, which will give the path for a particular movement from the current position.

$Q(S,a)modified=R(a')+r(s,a)+Ymax(Q(S',a'))$, where $R(a')$ is the reward function assigned for action *a'*.

5) Compare the modified utility factor of all movements, which are immediately possible and calculate the difference in the utility factor for the current state and the next state. Follow the path having higher value.

6) Repeat the steps 1-5 until the target position is reached.

5.2.4 Simulation Results

TD Learning algorithm is implemented for a 4x4 for different scenarios with different source point and fixed destination point in MATLAB. Fig.5.3 show the trajectory given by the TD Learning algorithm for different number and position of static obstacles.

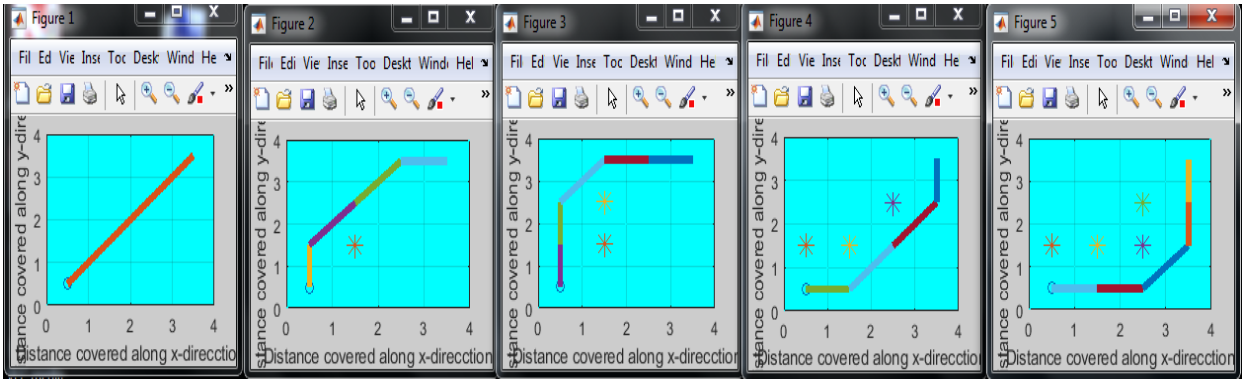


Fig 5.3: TD based Path Planning with 0,1,2,3 and 4 obstacles

It worked perfectly with different source points. The Fig.5.4 show the trajectory output given by the TD algorithm for variable starting points. The destination point is fixed at (3.5, 3.5). In Fig.5.4 (Figure 1), the starting point is at (0.5, 3.5) and it investigates one obstacle placed at (1.5, 3.5). Fig.5.4 (Figure 2) shows the case of starting point at (1.5, 0.5) with one obstacle at (2.5, 1.5). Fig.5.4 (Figure 3) shows a case with the starting point at (0.5, 1.5) and two obstacles placed at (1.5, 2.5) and (2.5, 2.5). In Fig.5.4 (Figure 4) starting point is at (1.5, 1.5) and two obstacles are considered at (1.5,2.5) and (2.5,2.5).

Fig 5.4: TD based Path Planning for different source points

There are certain cases where TD Learning algorithm fails to give output. Fig.5.5. shows two cases where TD Learning fails to find the optimum path to destination. In Fig.5.5 (Figure 1) starting point is at (0.5, 0.5) and three obstacles are placed at (0.5, 1.5), (1.5, 1.5) and (1.5, 0.5). Similar is the case of Fig.5.5 (Figure 2) where the starting point is again at (0.5, 0.5) and three obstacles are placed at (2.5, 3.5), (2.5, 2.5) and (3.5, 2.5). Since all forward movements are blocked by obstacles at starting point, the algorithm takes a backward movement initially but fails to move afterwards.

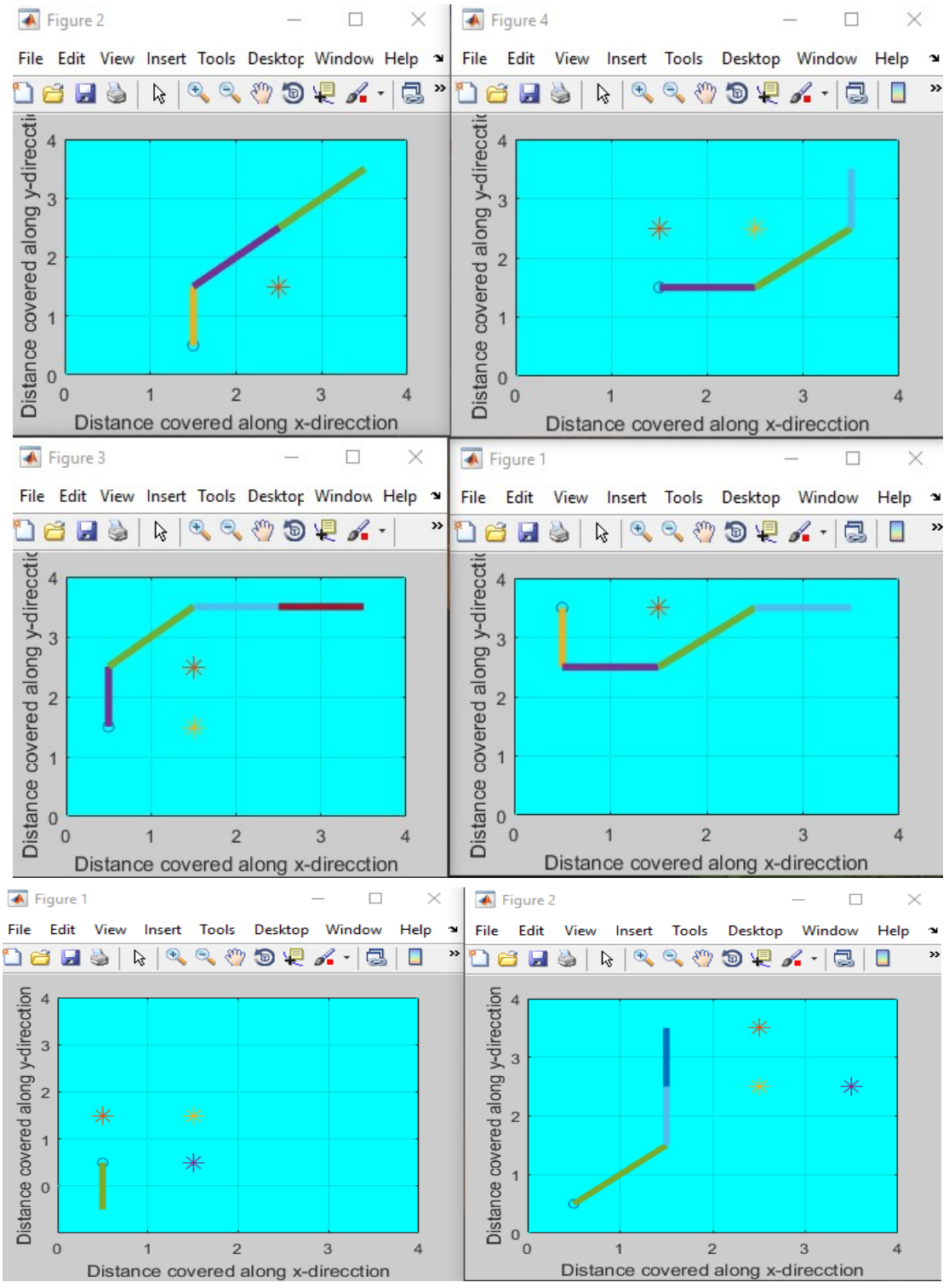


Fig 5.5: Cases where TD Learning fails

5.3 COMPARISON OF TD WITH A* AND DIJKSTRA FOR DIFFERENT SIZES OF GRID

TD algorithm was implemented for different sizes of grid and compared with A* and Dijkstra.

5.3.1 Simulation Results of A* and Dijkstra algorithm

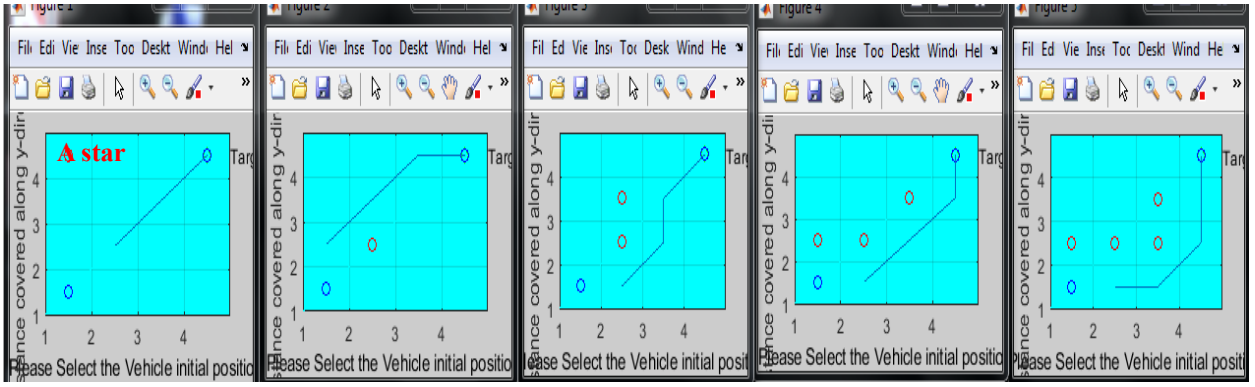


Fig 5.6: A* based path planning for 4x4 grid

A* and Dijkstra algorithm for 0,1,2,3 and 4 obstacles for a 4x4 grid is simulated as shown in the figure 5.6 and 5.7 respectively. A* and Dijkstra was successful in reaching the destination in all these cases. In both the figures 5.6 and 5.7, the obstacle occurrence is the same. For 1 obstacle the position is (1.5,1.5). For 2 obstacles, the positions are (0.5,0.5) and (1.5,2.5). For 3 obstacles, the positions are (0.5,1.5), (1.5,1.5) and (2.5,2.5). For 4 obstacles, the positions are (0.5,1.5), (1.5,1.5), (2.5,1.5) and (2.5,2.5).

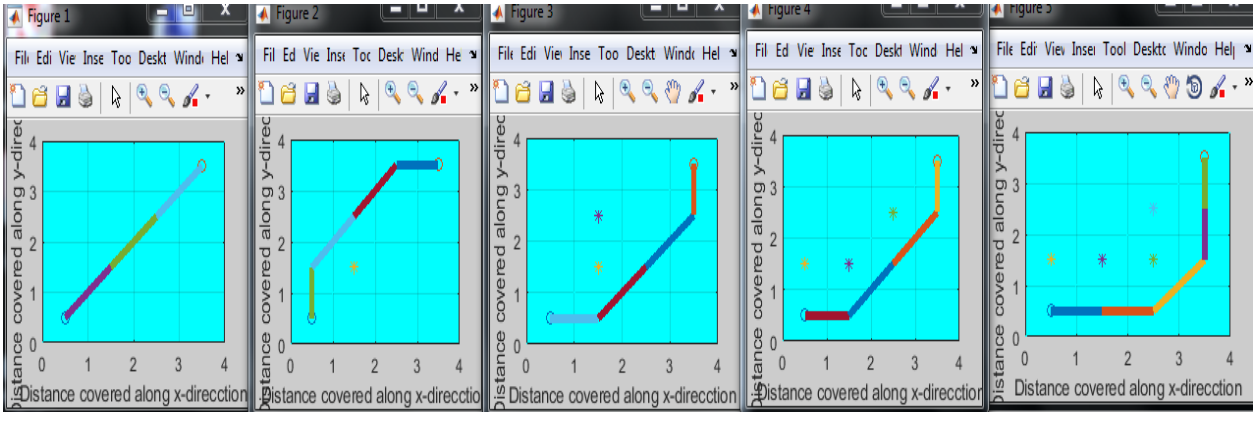


Fig 5.7: Dijkstra based path planning for 4x4 grid

5.3.2 Comparative Study between A*, Dijkstra and TD algorithm based on elapsed time for a 4X4 grid

From table 5.1 it is very evident that TD algorithm is the fastest compared to A* and Dijkstra.

Table 5.1: Elapsed time for A*, Dijkstra and TD algorithm

Number of Obstacles	Position of Obstacles	Time taken for Dijkstra algorithm	Time taken for A* algorithm	Time taken for TD algorithm
0	-	0.50636	0.15438	0.005119
1	(1.5,1.5)	0.75862	0.08102	0.005244
2	(1.5,1.5),(1.5,2.5)	0.75906	0.04286	0.007160
3	(0.5,1.5),(1.5,1.5),(2.5,2.5)	0.76045	0.08604	0.007299
4	(0.5,1.5),(1.5,1.5),(2.5,1.5),(2.5,2.5)	1.10113	0.08709	0.005827

5.3.3 Simulation Results for A* and TD algorithm for 5X5 grid

Temporal difference algorithm and A* for a 5x5 grid is simulated for 0,1,2,3,4 and 5 obstacles and the simulation results are shown in fig 5.8 and 5.9 respectively. A* and TD was successful in reaching the destination in all these cases. In both the figures 5.8 and 5.9, the obstacle occurrence is the same. For 1 obstacle the position is (1.5,1.5). For 2 obstacles, the positions are (0.5,0.5) and (0.5,2.5). For 3 obstacles, the positions are (0.5,1.5), (1.5,1.5) and (2.5,2.5). For 4 obstacles, the positions are (0.5,1.5), (1.5,1.5), (2.5,1.5) and (2.5,2.5). For 5 obstacles, the positions are (0.5,1.5), (1.5,1.5), (2.5,1.5), (2.5,2.5) and (3.5,2.5).

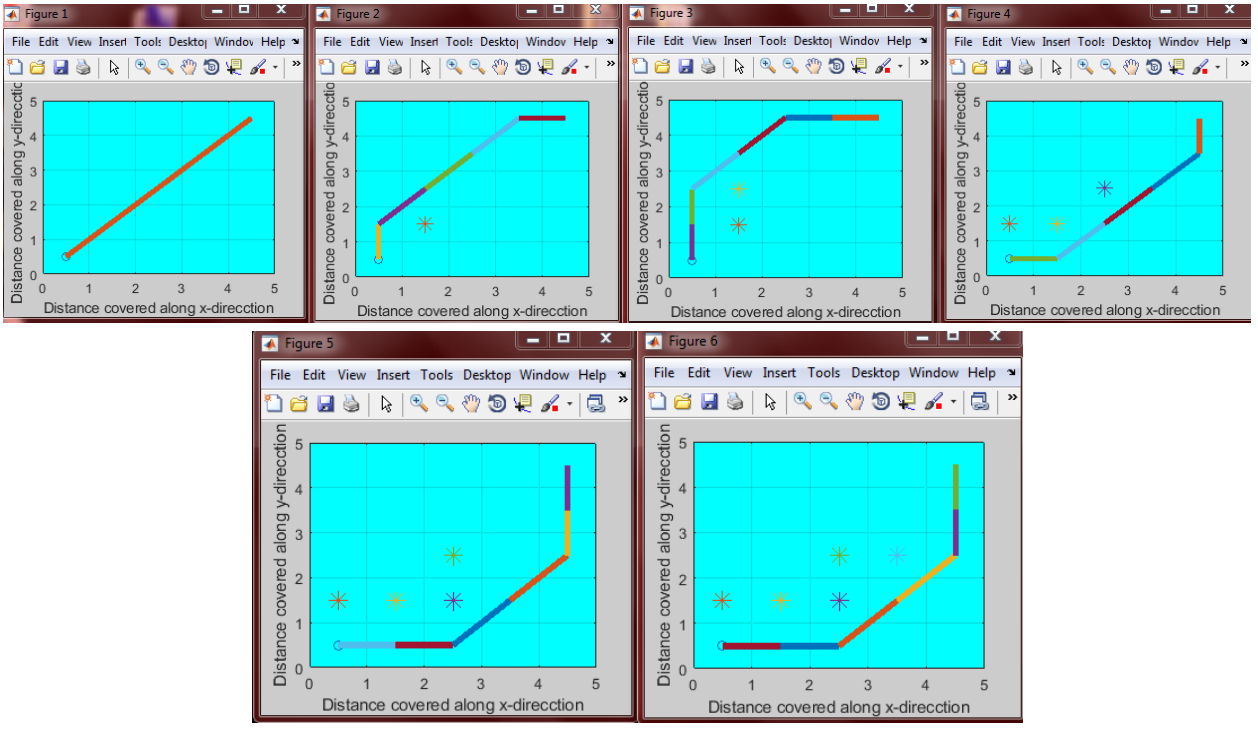


Fig 5.8: TD based path planning for 5x5 grid

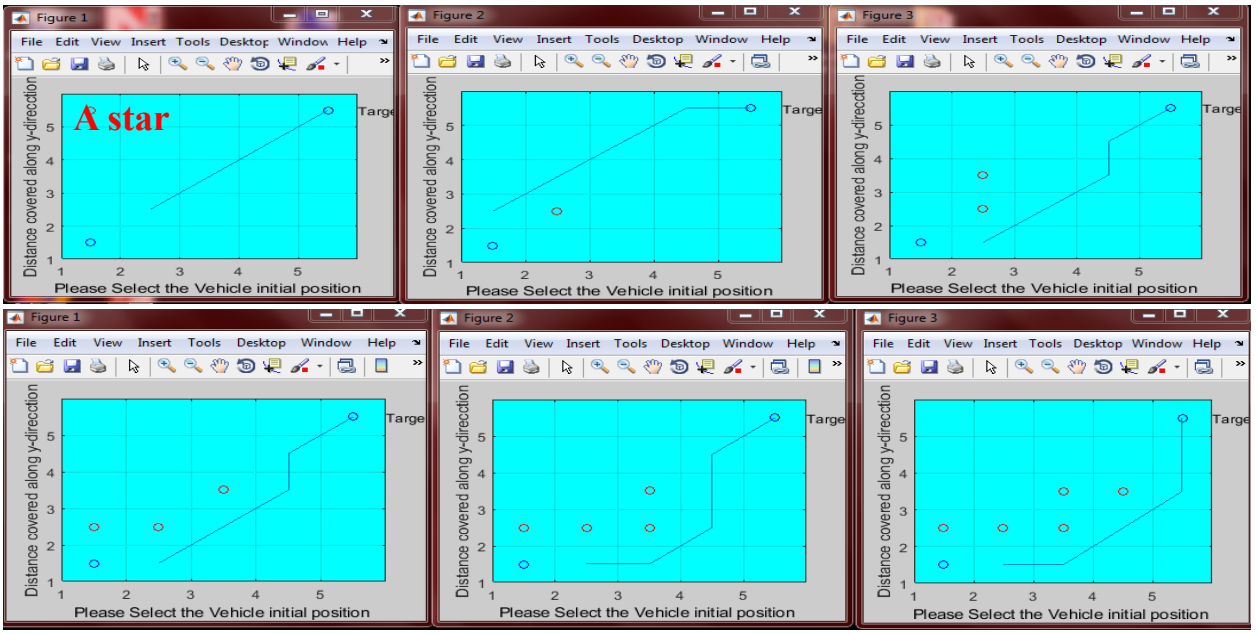


Fig 5.9: A* based path planning for 5x5 grid

The elapsed time of all these obstacle occurrence scenarios is found out and is shown in table 5.2.

All the measurements are in seconds.

Table 5.2: Elapsed time for 5x5 grid of A* and TD

Number of Obstacles	Position of Obstacles	Time taken for A* algorithm	Time taken for TD algorithm
0	-	0.76336	0.000857
1	(1.5,1.5)	1.01342	0.008074
2	(1.5,1.5),(1.5,2.5)	1.02009	0.009722
3	(0.5,1.5),(1.5,1.5),(2.5,2.5)	1.01378	0.006524
4	(0.5,1.5),(1.5,1.5),(2.5,1.5),(2.5,2.5)	1.26427	0.009797
5	(0.5,1.5),(1.5,1.5),(2.5,1.5),(2.5,2.5),(3.5,2.5)	1.26301	0.009710

5.3.4 Simulation Results for A* and TD algorithm for 6x6 grid

Temporal difference algorithm and A* for a 6x6 grid is simulated for 0,1,2,3,4,5 and 6 obstacles and the simulation results are shown in fig 5.10 and 5.11 respectively

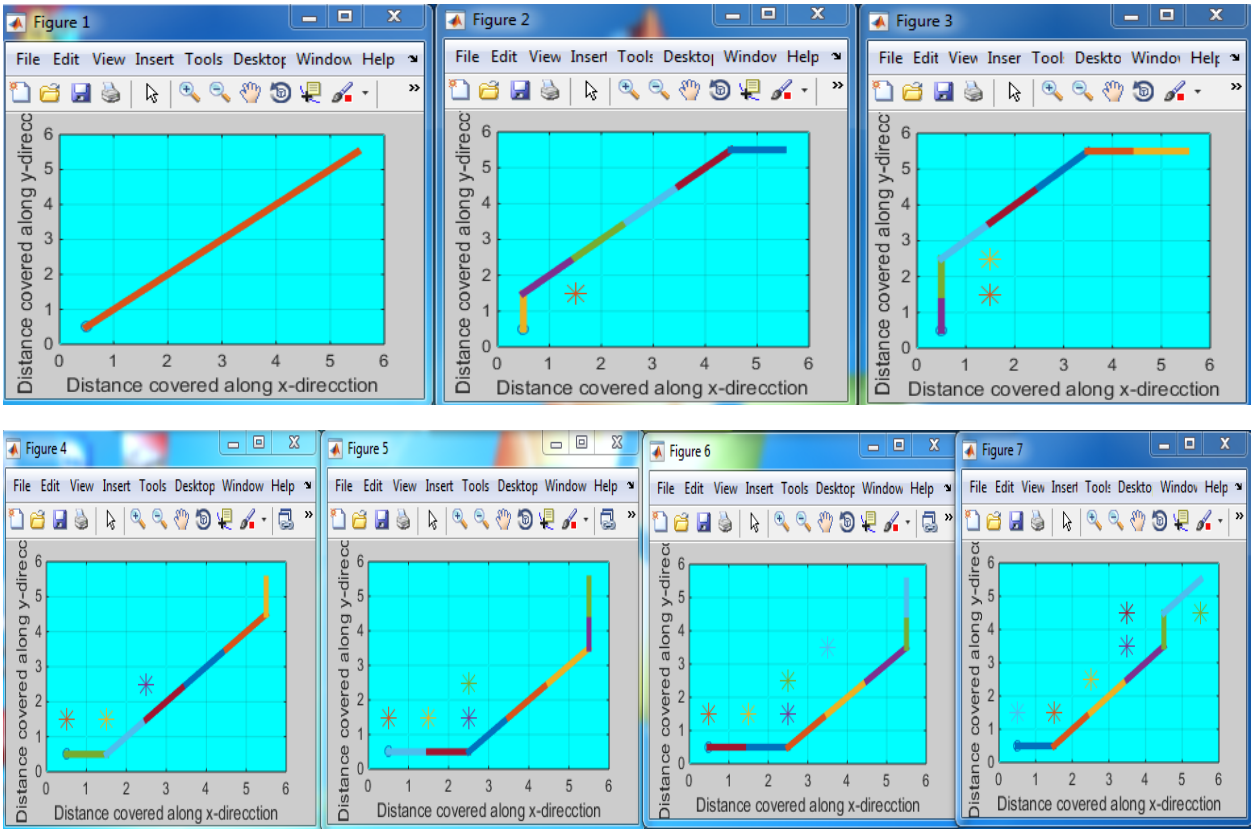


Fig 5.10: TD based path planning for 6x6 grid

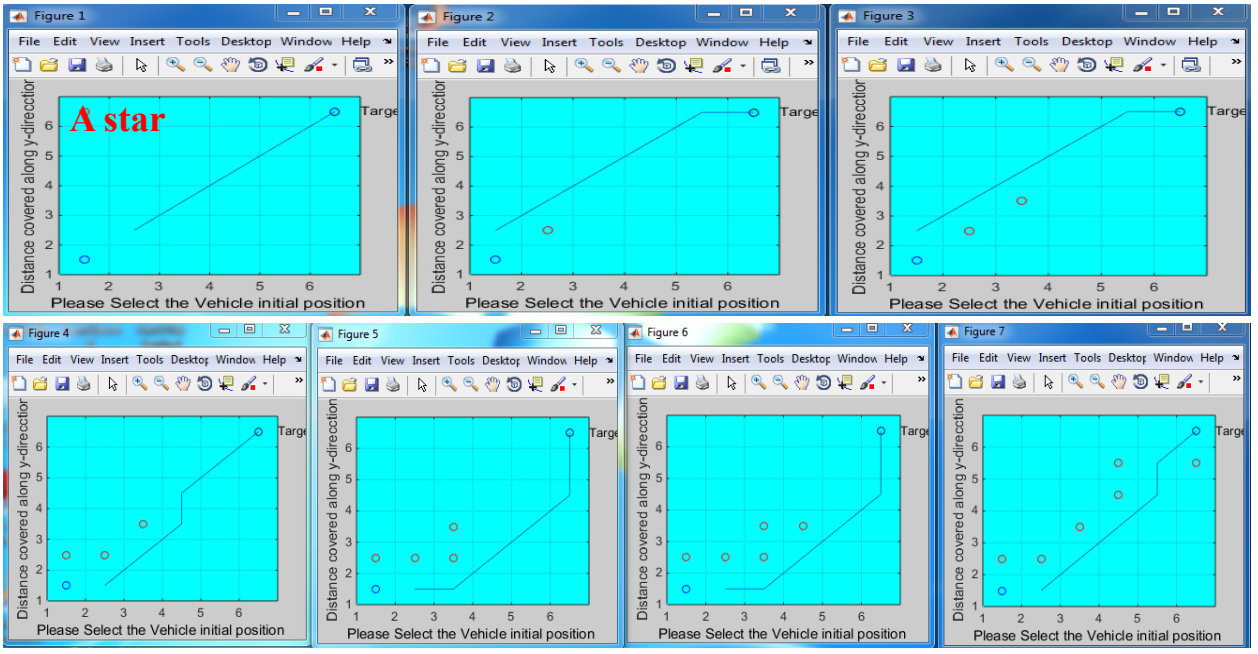


Fig 5.11: A* based path planning for 6x6 grid

The elapsed time of all these obstacle occurrence scenarios is found out and is shown in table 5.3.

All the measurements are in seconds.

Table 5.3: Elapsed time for 6x6 grid for A* and TD

Number of Obstacles	Position of Obstacles	Time taken for A* algorithm	Time taken for TD algorithm
0	-	1.052016	0.001834
1	(1.5,1.5)	1.328514	0.018820
2	(1.5,1.5),(1.5,2.5)	1.329846	0.018579
3	(0.5,1.5),(1.5,1.5),(2.5,2.5)	1.309244	0.019258
4	(0.5,1.5),(1.5,1.5),(2.5,1.5),(2.5,2.5)	1.588463	0.021461
5	(0.5,1.5),(1.5,1.5),(2.5,1.5),(2.5,2.5),(3.5,2.5)	1.566313	0.016919
6	(1.5,1.5),(2.5,2.5),(3.5,3.5),(5.5,4.5),(0.5,1.5),(3.5,4.5)	1.574019	0.015919

5.3.5 Study on the difference between A*, Dijkstra and TD algorithm

Table 5.4 shows the difference between A*, Dijkstra and TD learning algorithm.

Table 5.4: Difference between A*, Dijkstra and TD

A*	DIJKSTRA	TD
• Same as Dijkstra • A* is faster because it uses Best First Search • Heuristic Function, f(n)=g(n)+h(n), g(n) represents the cost of the path from the starting point to the vertex n. h(n) represents the heuristic estimated cost from vertex n to the g.	• It is Simple as compare to A* • Slower because it uses Greedy Best First Search • f(n)=g(n), g(n) represents the cost of the path from the starting point to the vertex n. • Dijkstra Algorithm is the worst case of A star Algorithm	• Fastest • Estimate the Q-function according to the reward function • Less stable and may converge to the wrong solution • $Q(S,a)=r(s,a)+Ymax(Q(S',a'))$

5.4 SARSA ALGORITHM

SARSA algorithm is simulated for 6x6,8x8 and 16x16. It is also simulated to solve complex mazes also.

5.4.1 Implementation of SARSA algorithm for 6x6 grid

It is implemented for 36 grids with different source and destination. Up to 30 obstacles is possible. The reward function for the source point is 0, for destination it is 1 and finally for the obstacles it is -1. For obstacles we are giving negative value for that the grid having obstacles is not chosen for path planning. The reward points are as shown in table 5.5

Table 5.5: Reward function for SARSA

SL	POSITION IN THE GRID	REWARD POINTS
1	Source Point	0
2	Destination Point	1
3	Obstacles	-1

There are two cases where SARSA algorithm fails. The first case is when the source is covered with obstacles so it won't be able to move from the source point and path planning is blocked. The second is when the destination point is blocked. These cases are shown in figure 5.12.

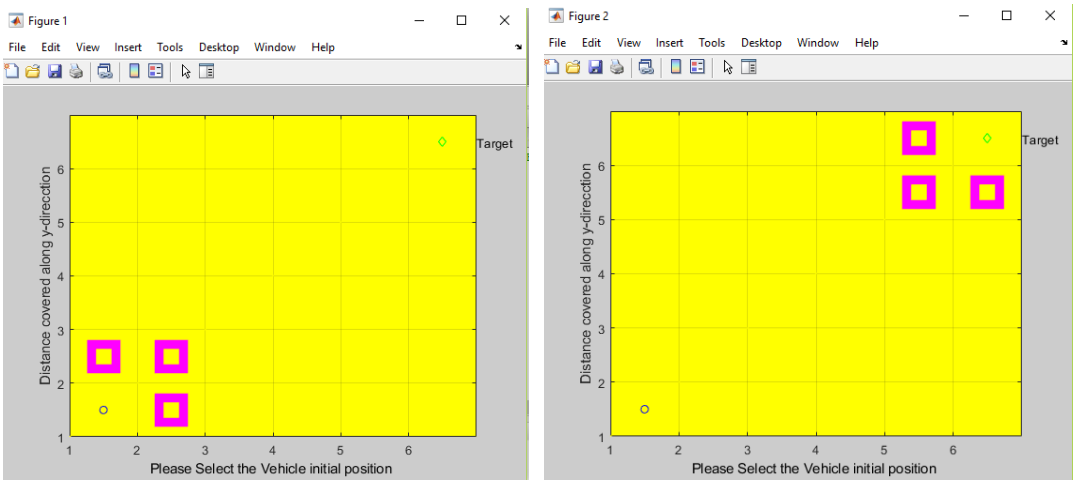


Figure 5.12: SARSA based path planning for 6x6 grid where it fails

Path planning for different cases for different number of obstacles is carried out from 0 to 7 obstacles and is shown in figure 5.13. The position of obstacles for 1 obstacle is (1.5,1.5). The position of obstacle for 2 obstacles is (2.5,1.5) and (2.5,2.5). The position of obstacle for 3 obstacles is (1.5,2.5) ,(2.5,1.5) and (2.5,2.5). The position of obstacle for 4 obstacles is (1.5,3.5), (2.5,2.5), (3.5,1.5) and (3.5,2.5). The position of obstacle for 5 obstacles is (1.5,1.5), (1.5,2.5), (2.5,3.5), (3.5,1.5) and (3.5,2.5). The position of obstacle for 6 obstacles is (1.5,1.5), (2.5,2.5), (2.5,3.5) (1.5,4.5), (3.5,2.5) and (4.5,1.5). The position of obstacle for 7 obstacles is (0.5,1.5), (1.5,1.5), (2.5,1.5), (2.5,2.5), (3.5,2.5), (3.5,3.5) and (3.5,2.5).

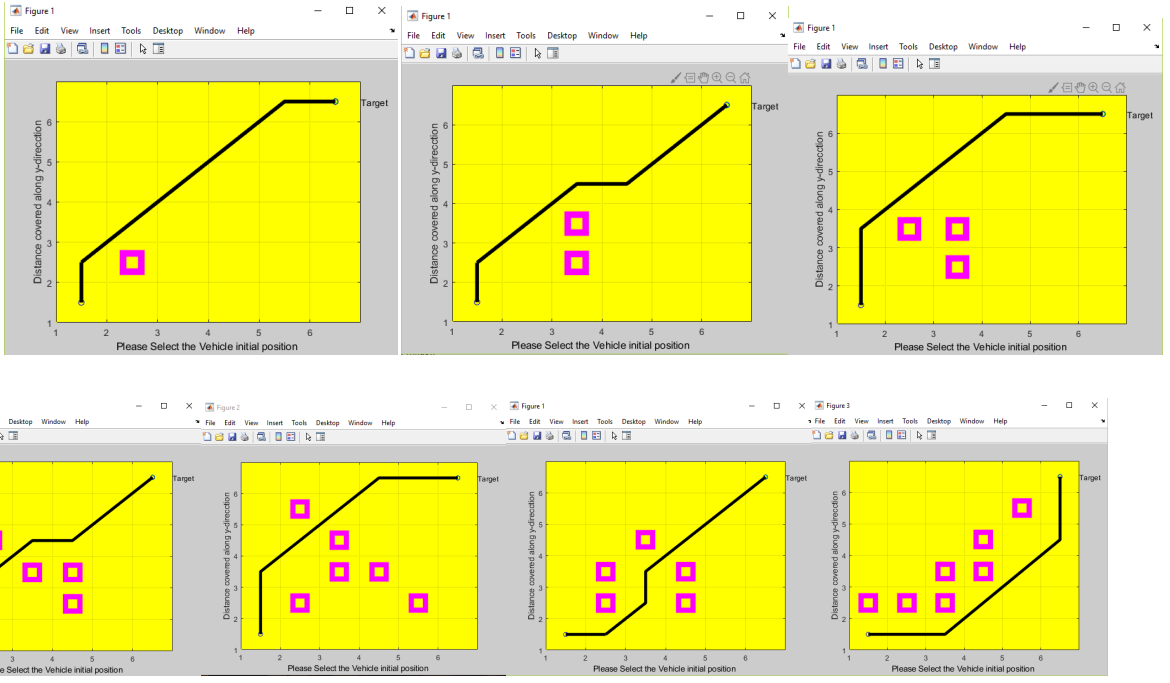


Fig 5.13: SARSA based path planning for 6x6 grid

5.4.2 Implementation of SARSA algorithm for 6x6 grid to solve complex mazes more than 15 obstacles

In this section, SARSA is used to solve complex mazes, with more than 15 obstacles and some of the cases are shown in fig 5.14 where there are 20 obstacles in the first figure, 25 in the second figure and 17 obstacles in the third figure.

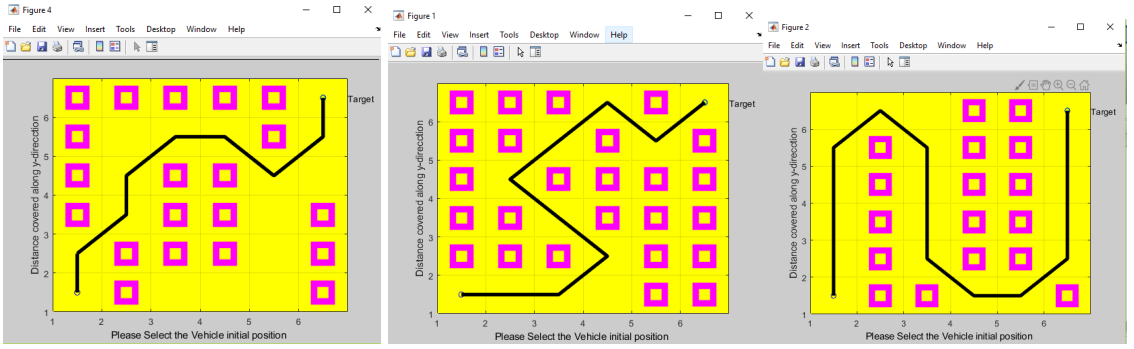


Fig 5.14: SARSA based path planning to solve complex mazes

5.4.3 Implementation of SARSA algorithm for 16x16 grid to solve complex mazes more than 15 obstacles

A 16x16 grid is considered with 256 grids. SARSA algorithm works perfectly for even more than 200 obstacles. Many cases are tested and some of the cases are shown in fig 5.15. There are 31 obstacles in the first figure, 27 in the second figure and finally 34 obstacles in the third image.

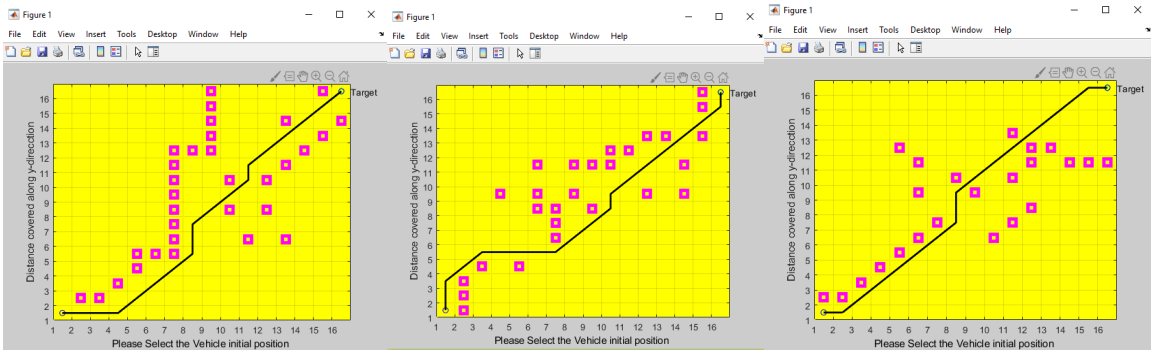


Fig 5.15: SARSA based path planning for 16x16 grid

5.4.4 Implementation of SARSA algorithm for 16x16 grid to solve complex mazes more than 15 obstacles

In this section, SARSA is used to solve complex mazes, with more than 100 obstacles and some of the cases are shown in fig 5.16 where there are 123 obstacles in the first figure, 185 in the second figure and 148 obstacles in the third figure.

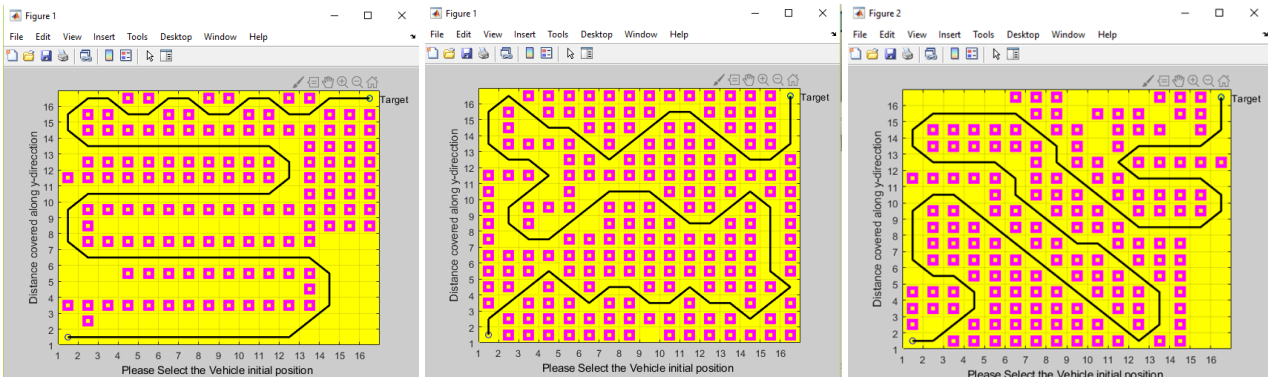


Fig 5.16: SARSA based path planning to solve complex mazes for 16x16 grid

5.4.5 Implementation of SARSA algorithm for 8x8 grid to solve complex mazes more than 25 obstacles

SARSA algorithm for a complex grid with 35 obstacles is simulated. The total number of iterations chosen for exploring the environment is 100.The source and destination of the grid can be varied. The reward function of the system is as given in table 5.6.

Table 5.6: Reward Function for SARSA for 8x8 grid

SL NO	POSITION IN THE GRID	REWARD POINTS
1	Source Point	0
2	Destination Point	10
3	Available Path	0
4	Obstacles	-50

Simulation results of SARSA algorithm for an 8x8 grid is as shown in the coming figures. In fig 5.17 the reward function for each function is according to the table 5.6. The dark yellow color is for the path available. The dark blue color is for the obstacles.

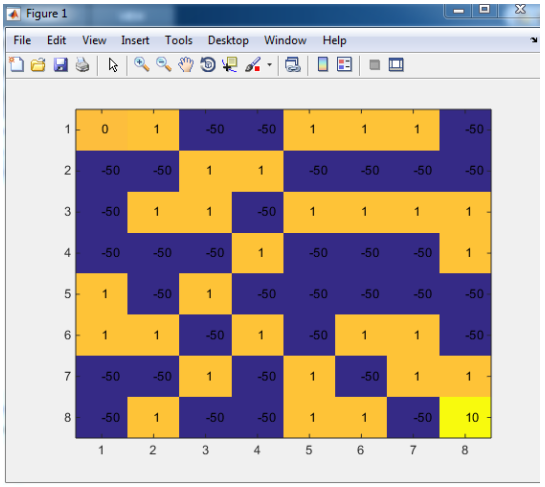


Fig 5.17: System layout of SARSA algorithm for 8x8 grid with reward function

To understand the obstacle position properly a system layout highlighting the obstacles is shown in the figure 5.18. The obstacles are shown as -X-. The start and goal points are also marked. The green color is for obstacles and the yellow color is for the path available.

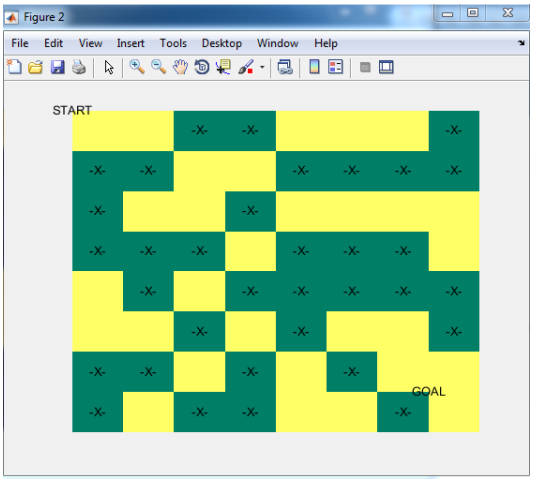


Fig 5.18: System layout of SARSA algorithm highlighting the obstacles.

The source and destination grid number should be given as input in the command window as shown in fig 5.19

```
enter the start point:1
enter the destination point64
```

Fig: 5.19 Data given through command window

After path planning is shortest path to the destination is found out and it is shown in figure 5.20.

The green color is the path taken from source to destination which has red dots in the center.

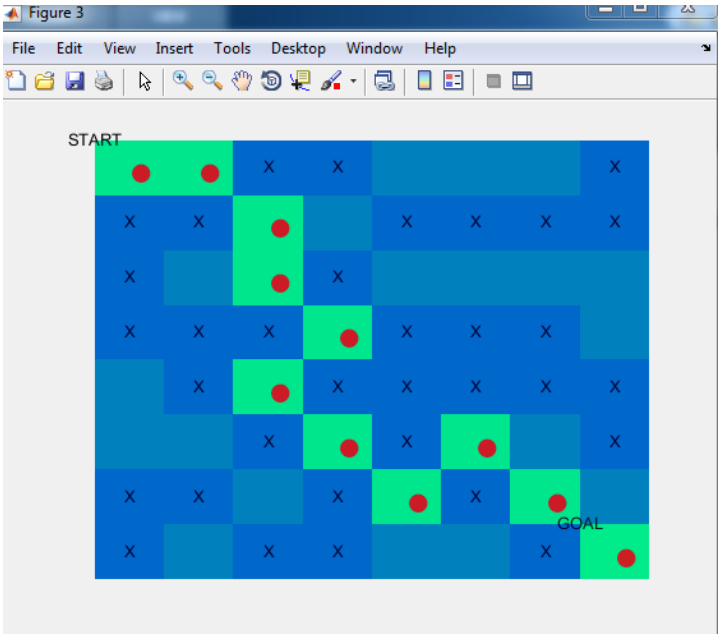


Fig 5.20: SARSA based path planning to solve complex mazes for 8x8 grid

```
path =
     1     2

path =
     1     2    11

path =
     1     2    11    19

path =
     1     2    11    19    28

path =
     1     2    11    19    28    35

path =
     1     2    11    19    28    35    44

path =
     1     2    11    19    28    35    44    53

path =
     1     2    11    19    28    35    44    53    62

path =
     1     2    11    19    28    35    44    53    62    55

path =
     1     2    11    19    28    35    44    53    62    55    64
Final Path: 1   2  11  19  28  35  44  53  62  55  64>>
```

Fig 5.21: Path taken shown in Command Window

The path taken after entering in each grid is shown in command window and it is shown in the figure 5.21. In the end the final path will also be shown after the path planning is successful. The source point of the 8x8 grid can be changed and is shown in fig 5.22. In the first figure the start point is 21. In the second figure the start point is 18. Finally, in the third image the start point is 41. In all these cases the destination point is fixed which is 64

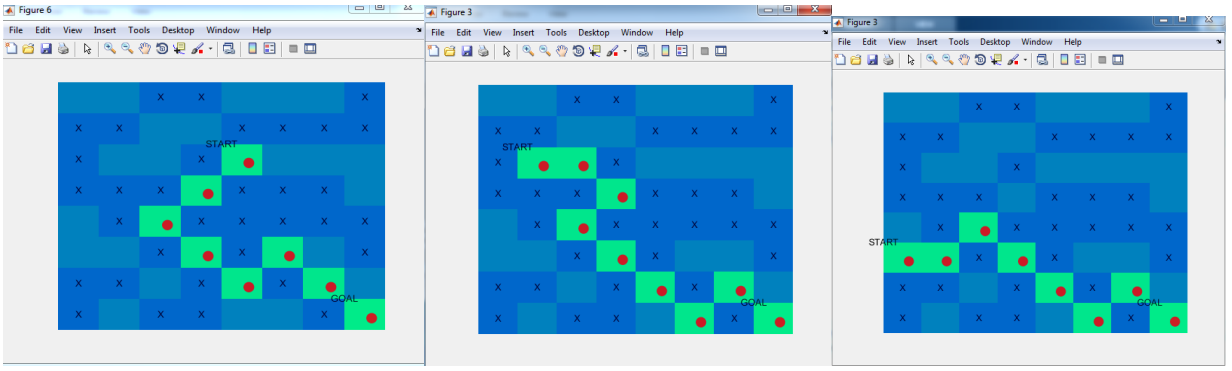


Fig 5.22: SARSA based path planning to solve complex mazes for 8x8 grid with different source points

The destination point of the 8x8 grid can be changed and is shown in fig 5.23. In the first figure the destination point is 46. In the second figure the destination point is 35. Finally, in the third image the destination point is 53. In all these cases the start point is fixed which is 1.

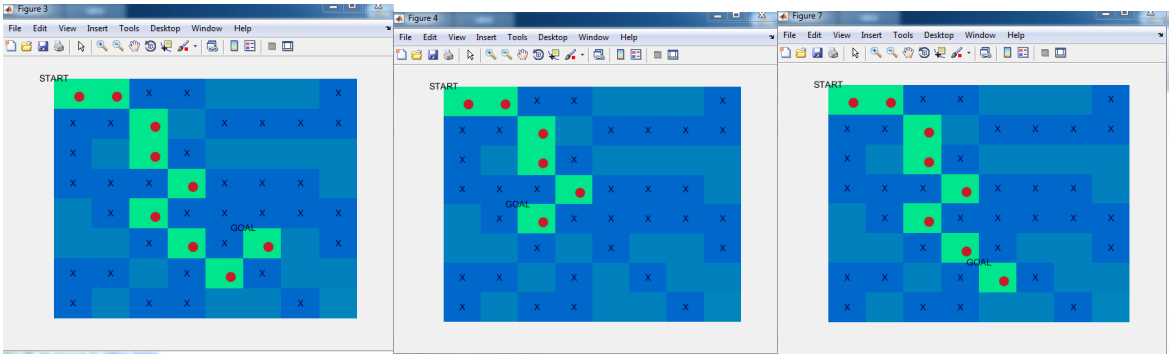


Fig 5.23: SARSA based path planning to solve complex mazes for 8x8 grid with different destination points

5.4.6 Implementation of SARSA algorithm with altering source points

SARSA algorithm is implemented not only for a fixed source point but also for altering source points. In fig 5.24 SARSA algorithm is implemented for different source points. Eight such cases are shown below. Here, in picture 1 the source point is (2.5,1.5), in picture 2 the source point is (1.5,2.5), in picture 3 the source point is (1.5,3.5), in picture 4 the source point is (1.5,4.5), in picture 5 the source point is (1.5,3.5), in picture 6 the source point is (3.5,2.5), in picture 7 the source point is (4.5,2.5) and in picture 8 the source point is (4.5,1.5).

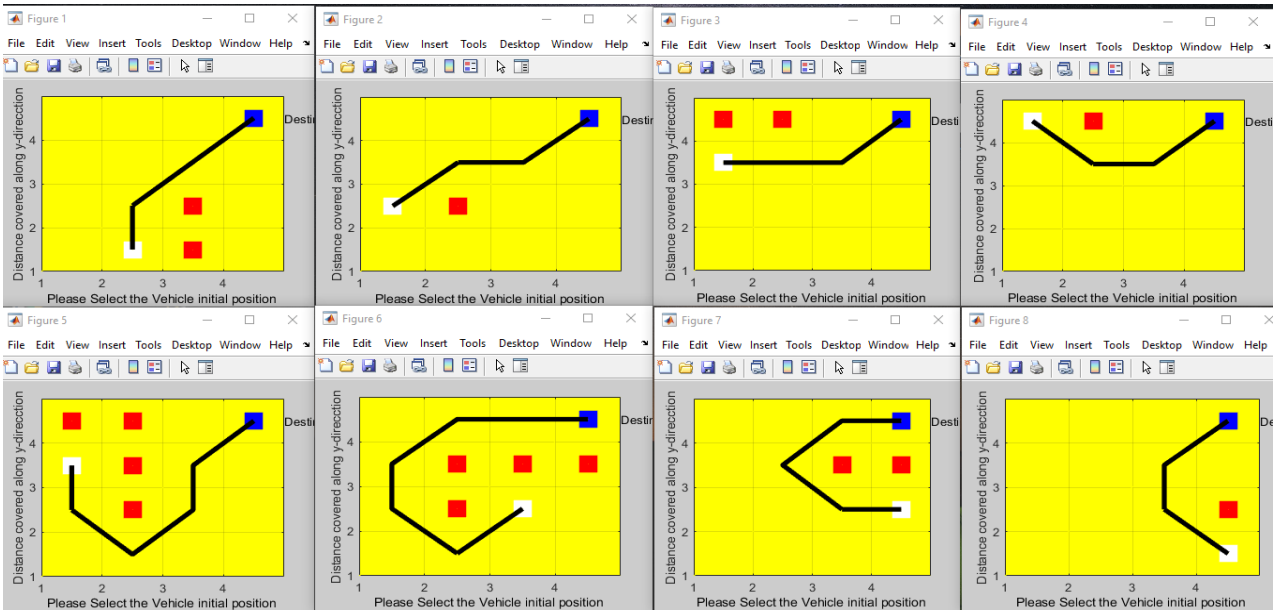


Fig 5.24: SARSA algorithm with altering source points

5.4.7 Implementation of SARSA algorithm with altering destination points

SARSA algorithm is implemented not only for a fixed destination point but also for altering destination points. In fig 5.25 SARSA algorithm is implemented for different destination points. Eight such cases are shown below. Here, in picture 1 the source point is (3.5,1.5), in picture 2 the source point is (4.5,3.5), in picture 3 the source point is (1.5,4.5), in picture 4 the source point is

(2.5,2.5), in picture 5 the source point is (4.5,1.5), in picture 6 the source point is (2.5,3.5), in picture 7 the source point is (4.5,2.5) and in picture 8 the source point is (3.5,1.5).

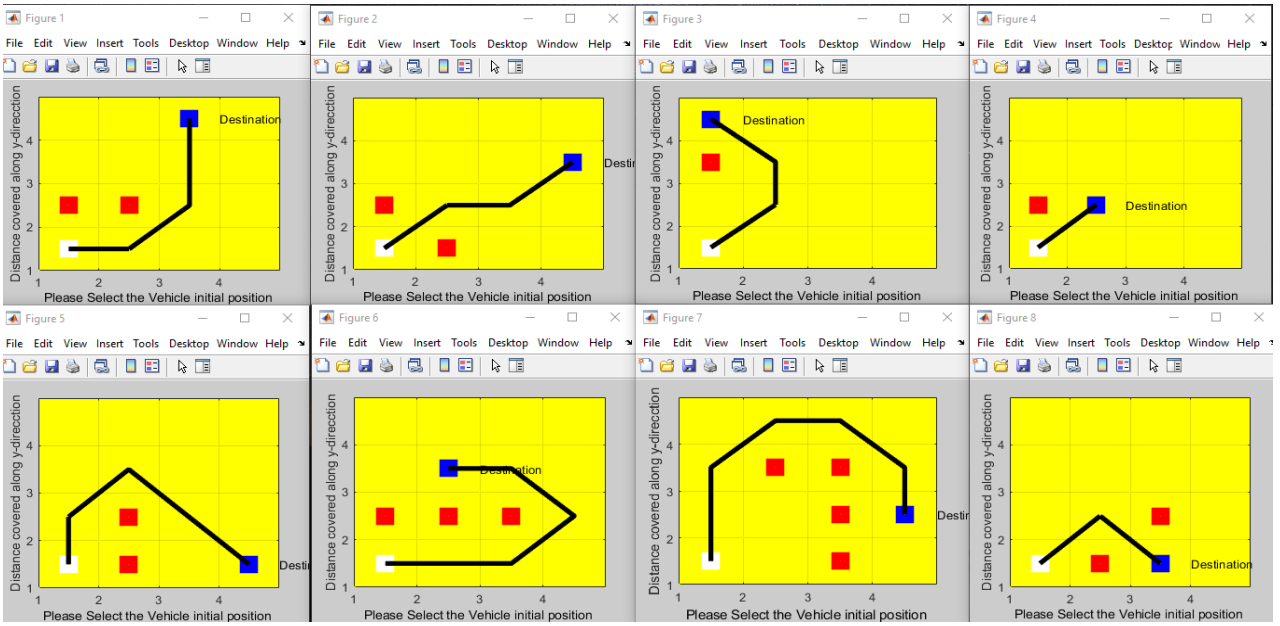


Fig 5.25: SARSA algorithm with altering destination points

5.4.8 Implementation of SARSA algorithm with altering source and destination points

SARSA algorithm is implemented for altering source and destination points. In fig 5.26 SARSA algorithm is implemented for different source and destination points. Eight such cases are shown below. Here, in picture 1 the source point is (4.5,1.5) and destination point is (1.5,2.5), in picture 2 the source point is (2.5,2.5) and destination point is (3.5,4.5), in picture 3 the source point is (1.5,2.5) and destination point is (4.5,3.5), in picture 4 the source point is (4.5,1.5) and destination point is (1.5,4.5), in picture 5 the source point is (1.5,4.5) and destination point is (3.5,4.5), in picture 6 the source point is (2.5,1.5) and destination point is (4.5,2.5), in picture 7 the source point is (3.5,1.5) and destination point is (4.5,2.5) and in picture 8 the source point is (2.5,1.5) and destination point is (2.5,3.5).

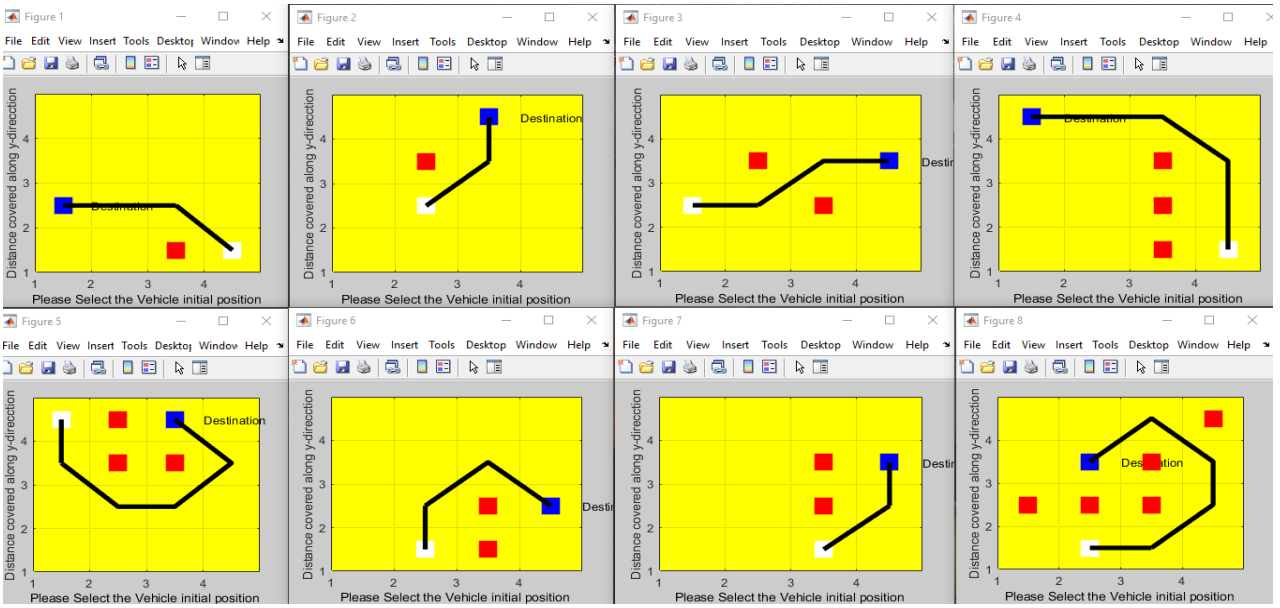


Fig 5.26: SARSA algorithm with altering source and destination points

5.4.8 Implementation of SARSA algorithm where source point is at the top of the grid and destination point is at the bottom of the grid

SARSA algorithm is implemented where the source point is at the top of the grid and destination point is at the bottom of the grid and fig 5.27 shows the same. Eight such cases are shown below. Here, in picture 1 the source point is (4.5,4.5) and destination point is (1.5,1.5), in picture 2 the source point is (3.5,4.5) and destination point is (1.5,2.5), in picture 3 the source point is (1.5,4.5) and destination point is (4.5,1.5), in picture 4 the source point is (1.5,4.5) and destination point is (4.5,2.5), in picture 5 the source point is (1.5,2.5) and destination point is (4.5,1.5), in picture 6 the source point is (1.5,2.5) and destination point is (3.5,2.5), in picture 7 the source point is (4.5,4.5) and destination point is (1.5,1.5) and in picture 8 the source point is (4.5,4.5) and destination point is (4.5,1.5).

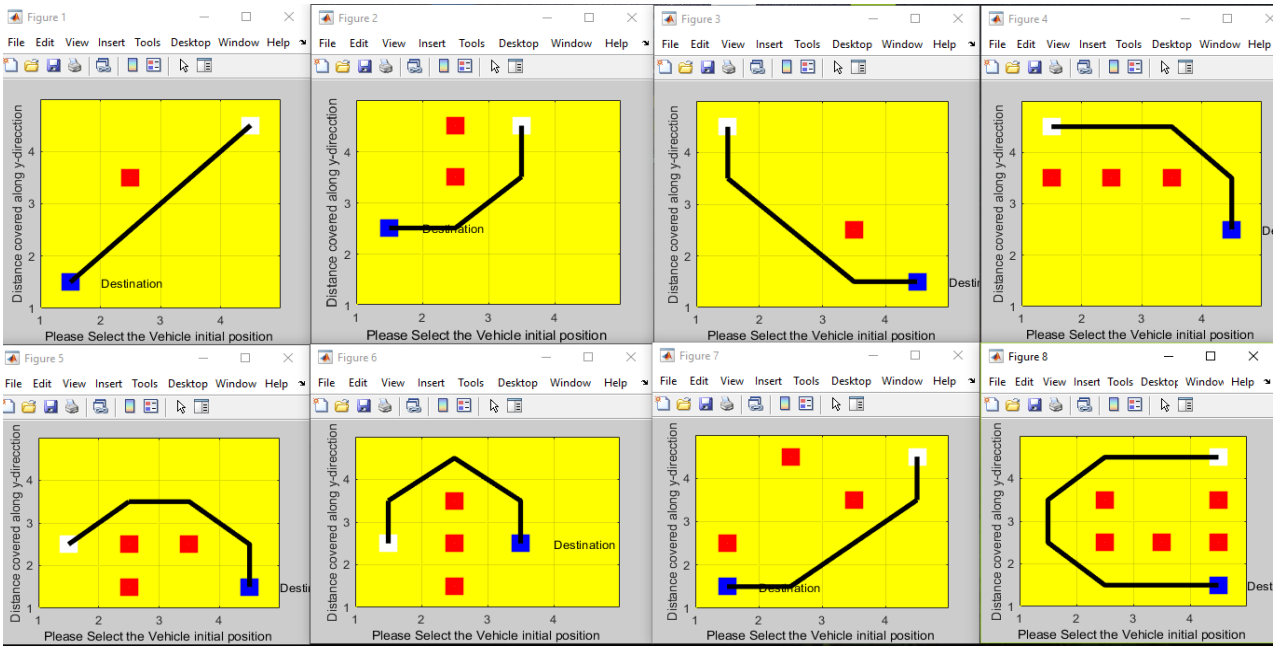


Fig 5.27: SARSA algorithm with source point at the top of the grid and destination at the bottom of the grid

5.5 COMPARISON BETWEEN TD AND SARSA ALGORITHM

SARSA and TD algorithm is compared in terms of their success rates and also on the basis of their working algorithm

5.5.1 Study on the difference between SARSA and TD algorithm

Table 5.7: Difference between SARSA and TD learning algorithm

SARSA	TD LEARNING
• ON policy • $Q(st,at) \leftarrow Q(st,at) + \alpha[rt+1+\gamma Q(st+1,at+1) - Q(st,at)]$ • Take actual action • Explores the environment • Allows possible penalties from exploratory restricted areas • SARSA will tend to avoid a dangerous optimal path and only slowly learn to use it when the exploration parameters are reduced.	• OFF policy • $Q(st,at) \leftarrow Q(st,at) + \alpha[rt+1+\gamma \max a' Q(st+1,a') - Q(st,at)]$ • Take the action with highest reward. • Doesn't explore the environment • Q-learning will ignore them. • More concentrated on the optimal path and not on the consequences

5.5.2 Comparison of TD and SARSA based on their success rate

Compared to TD algorithm, SARSA algorithm has got more success rate. There are cases where TD algorithm fails and SARSA works perfectly. Three such cases are highlighted here. In fig 5.28, TD fails for four obstacles and their positions are (3.5,5.5), (3.5,4.5), (3.5,3.5) and (3.5,2.5) which is represented as pink square boxes. It fails to reach the destination point (5.5, 5.5). According to TD algorithm it gives maximum priority for diagonal movement and moves forward, Due to the presence of obstacles in the positions (3.5,2.5) and (3.5, 3.5), it takes a vertical up movement and reaches the top and cannot proceed further. Hence TD failed in this condition. However, SARSA works perfectly for this condition which is shown in fig5.29. The specialty of SARSA algorithm is the interaction with its environment. Here it randomly selects grids and find out the possible routes from the grid till it reaches its destination. This helps it to tackle situations where TD fails but SARSA works properly.

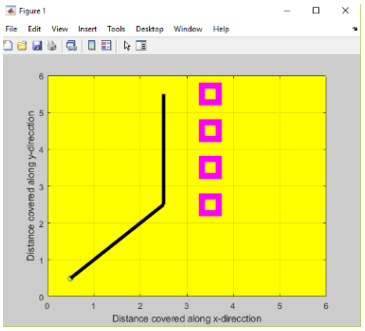


Fig 5.28: Case 1 where TD fails

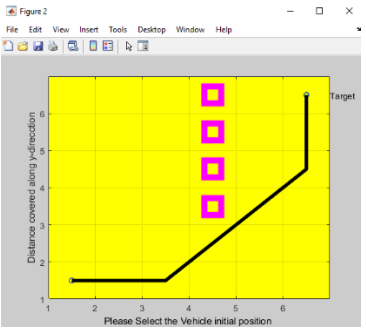
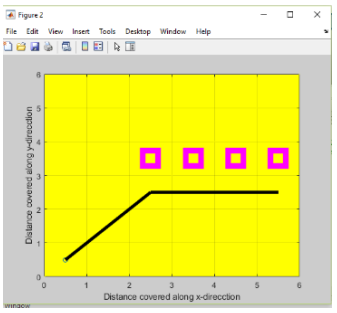


Fig 5.29: Case 1 where SARSA works

In fig 5.30, case 2 where TD fails is shown for four obstacles with their positions (2.5,3.5), (3.5,3.5), (4.5,3.5) and (5.5,3.5). Here also it fails to reach its destination grid. The path cannot proceed more when continuous horizontal right movements are taken after diagonal movements. For this particular case SARSA works and reaches the destination without any collision with the obstacles which is illustrated in fig 5.31.



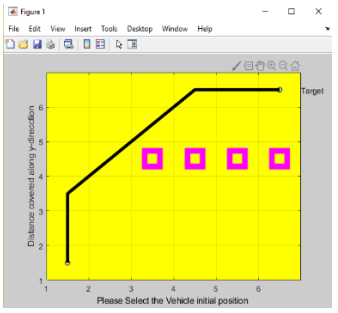
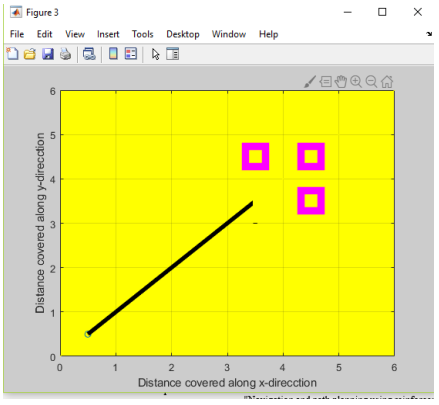


Fig 5.30: Case 2 where TD fails Fig 5.31: Case 2 where SARSA works

Fig 5.32, illustrates the third case where TD fails for three obstacles and their positions are (3.5,4.5), (4.5,4.5) and (4.5,3.5). Here the path cannot proceed because all the direction of movement i.e. diagonal, horizontal and vertical movements is obstructed. The path taken by SARSA is shown for the above case in fig 5.33 which successfully reaches the end point.



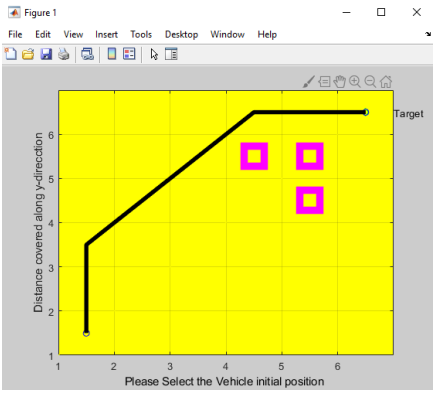


Fig 5.32: Case 3 where TD fails Fig 5.33: Case 3 where SARSA works

Several cases of 3,4,5 and 6 obstacles in a 6×6 grid are analyzed for both the TD and SARSA algorithm. A few such case are listed in Table 5.8, where TD fails and SARSA works. From these cases its evident that SARSA is better than TD when it comes to better path planning. Its success rate is much higher than TD. TD fails in all the above cases due to greedy policy where priority is given for each direction of movement and not for the optimal path planning. SARSA learns about the environment first and then does the path planning which helps the algorithm to determine the optimal path.

Table 5.8: Cases where TD fails and SARSA works for different number of obstacles

SL No	No. of Obstacles	Special Cases	TD	SARSA
1	3	a. (4.5,5.5),(4.5, 4.5),(4.5 ,3.5) b. (3.5,4.5),(4.5, 4.5), (4.5,3.5)	✗ ✗	✓ ✓
2	4	a. (3.5,5.5),(3.5 ,4.5), (3.5,3.5), (3.5,2.5) b. (2.5,3.5),(3.5, 3.5), (4.5,3.5), (5.5,3.5)	✗ ✗	✓ ✓
3	5	a. (2.5,4.5),(3.5, 4.5), (4.5,4.5), (4.5,3.5),(4.5,2.5) b. (0.5,5.5),(1.5, 4.5), (2.5,3.5), (3.5,2.5),(4.5,1.5)	✗ ✗	✓ ✓
4	6	a. (2.5,4.5),(3.5, 4.5),(3.5,3.5), (3.5,2.5),(3.5,1.5),(3.5,0.5) b.(0.5,3.5),(1.5,3.5),(2.5,3.5),(3.5,3.5), (3.5,2.5), (3.5,1.5)	✗ ✗	✓ ✓

5.6 IMPLEMENTATION OF TD ALGORITHM IN IROBOT FOR A 4X4 GRID

TD algorithm was implemented for a 4X4 grid with different number of obstacles in hardware with Mbed microcontroller interfaced with iRobot create. The various steps involved are described below.

5.6.1 Development of android based Bluetooth app

Android based Bluetooth app is created using MIT app inventor. The layout of how this app is shown in fig 5.34. In this figure the right side contains the elements that needs to be added to the app like button, checkboxes, date picker and son on. The middle portion of this app is the screen which is how the app will be in the phone after keeping all the elements in the screen. All the numbered blocks in the screen are buttons and they are placed by dragging the elements from the left side. The first horizontal row from the top numbered from 1 to 5 is for communicating the number of obstacles in the grid. Maximum of only 5 obstacles is only possible. The rest of the rows from 2 to 5 is for identifying the position of obstacles after the number of obstacles is selected. The 4x4 grid numbered from 1 to 16 is represented here. Finally, the Bluetooth symbol at the end is for enabling the Bluetooth connection with the Bluetooth module interfaced with a microcontroller. There are two columns in the right side of the app. The first one from the left is the elements that are present in the screen and the second column is for the properties of each of the element added in the screen. By this the app design part is completed. Now we need to specify what values will be send to the microcontroller when these buttons and for doing so we go to the blocks section.

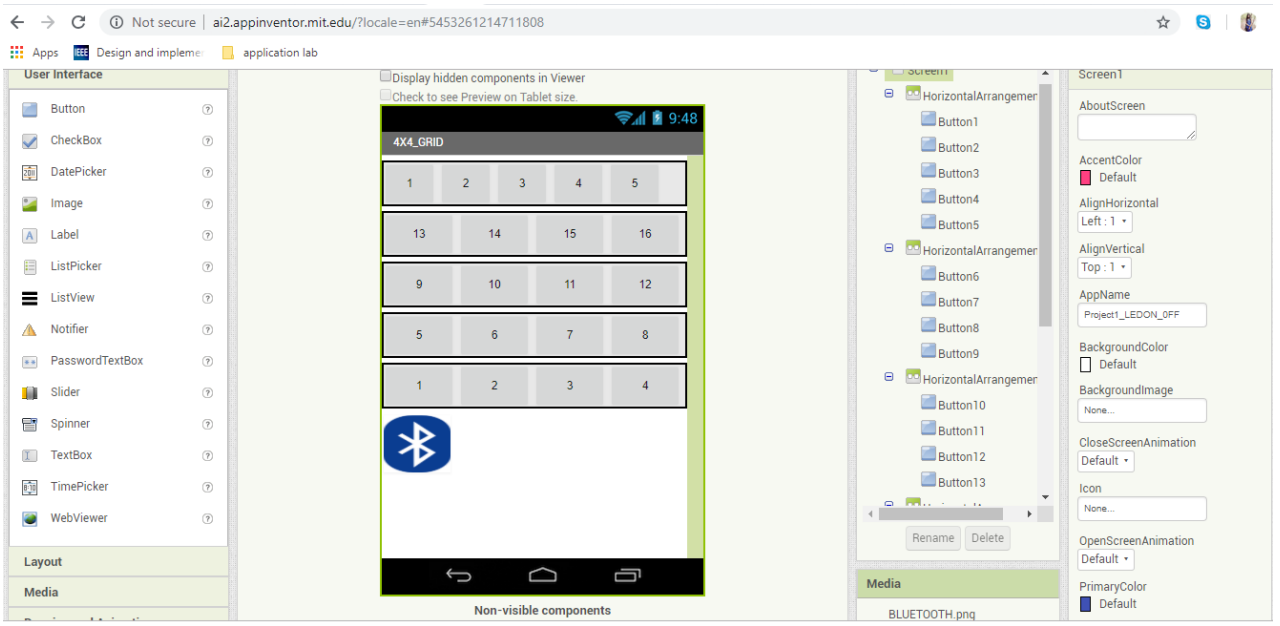


Fig 5.34: MIT App Inventor

In the blocks section, we define what value needs to be assigned for each button to send. For doing so there are pre-defined blocks available. All we need to do is place the blocks for each function correctly. This is shown in fig 5.35. The left side column contains all the elements added to the app and by clicking them blocks applicable to that particular element will be displayed and according to our requirement blocks should be selected. First task is to enable Bluetooth communication. Before picking the Bluetooth clients, we can see all the linked clients available and after picking we select the client which is chosen. This is shown in the first horizontal row of the blocks section. Rest all the blocks are for buttons and in this we specify what a particular button of sending uniquely to the Bluetooth module interfaced with a microcontroller. By this the complete design part is completed.

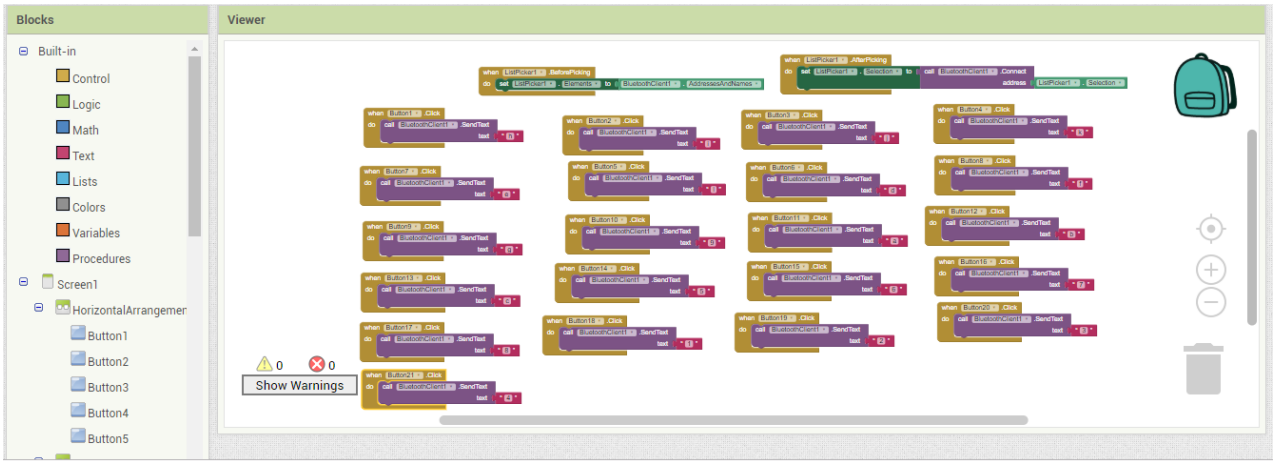


Fig 5.35: Blocks Section of MIT App Inventor

After the design part of the app is completed, build the program and save it to the computer. Then an apk file will be created. Copy this file to the phone. This is an installation file. Open it and install the file. Then the app is created with the project name and with MIT app inventor app icon. All these procedures are shown in fig 5.36.

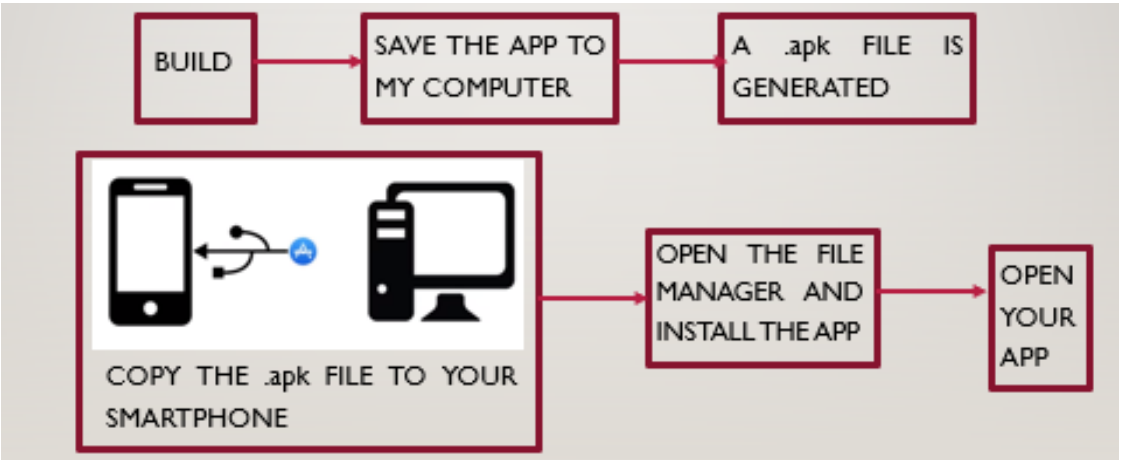


Fig 5.36: Steps involved after app design

Some of the screenshot after the app is installed below. Fig 5.37 shows how the app icon looks like after its installed. The name of the app is the project name and the icon is that of MIT App

Inventor and it is highlighted with a blue box. Fig 5.38 shows how the app looks after the app is opened. Finally, fig 5. 39 shows the linked clients after the Bluetooth icon is selected.

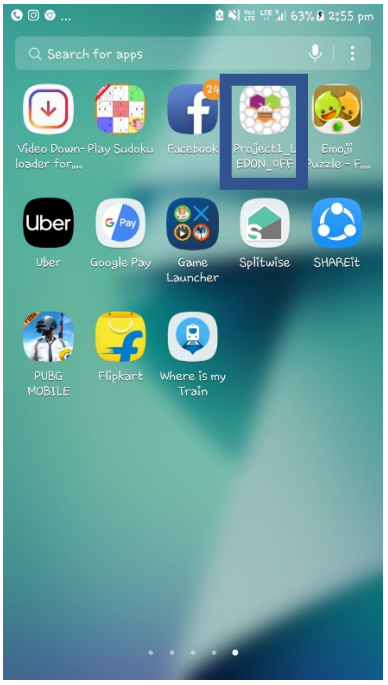


Fig 5.37: Screenshot of the app icon

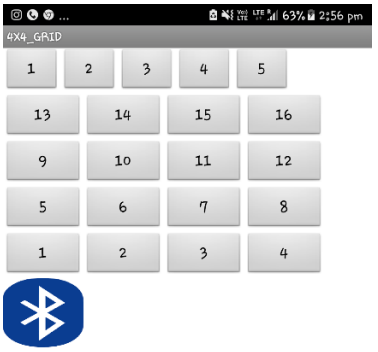


Fig 5.38: Screenshot of how the app looks when its opened

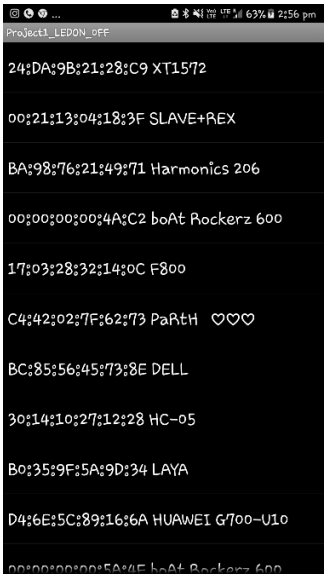


Fig 5.39: Screenshot of how the app looks after the Bluetooth icon is clicked

5.6.2 Communication between android app and Mbed microcontroller

After the app development part, the app needs to communicate with a microcontroller. Here the microcontroller chosen is Mbed microcontroller. Communication between Android app and Mbed is shown in fig 5.40.

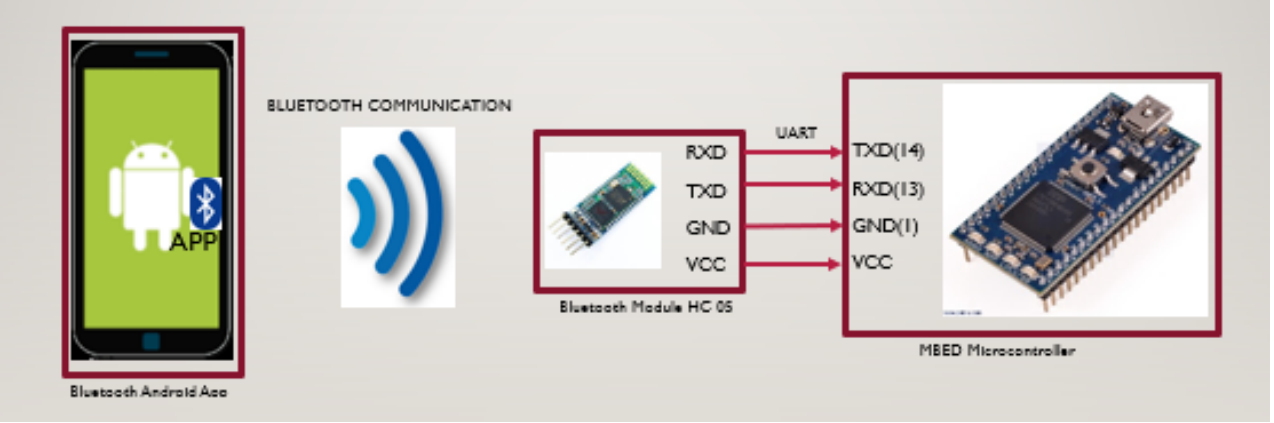


Fig 5.40: Block Diagram of Communication between Android app and mbed microcontroller

Here for Bluetooth communication, the client is HC-05. After the client is selected in the app the buttons can be selected. The Bluetooth module HC-05 is interfaced with mbed microcontroller through UART. To uniquely find the which button is pressed, we send unique characters while each button is pressed. This is shown in the table 5.9 and table 5.10. Table 5.9 represents the characters send while selecting the number of obstacles. Table 5.10 represents the characters send while selecting the position of obstacles.

Table 5.9: Characters send while selecting the number of obstacles

Buttons for choosing the number of the obstacles	Ascii value of the below given number will be send after selecting the button
1	H
2	I

3	J
4	K
5	L

Table 5.10: Characters send while selecting the position of obstacles

Buttons for choosing the position of the obstacles	Ascii value of the below given number will be send after selecting the button
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	G

The screenshots of the hardware are as shown below. Fig 5.41 shows the app in the phone and the client which is HC-05. When this client is selected the connection is established. Fig 5.42 shows the hardware setup. Fig 5.43 shows the output in the serial terminal for 1- 5 obstacles.

Fig 5.41: Linked clients- HC-05

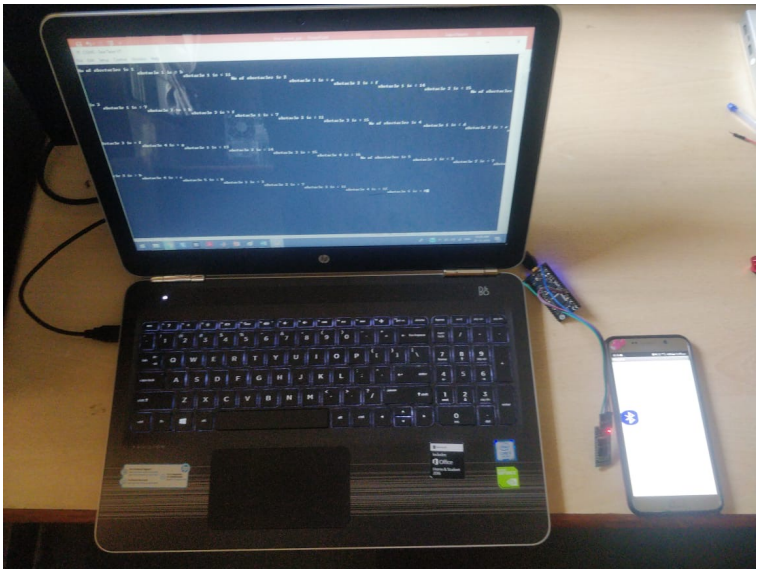


Fig 5.42: Hardware Setup

Fig 5.43: Serial Terminal Output for 1-5 Obstacles

5.6.3 Study of iRobot

A robotic vacuum cleaner is an autonomous robotic vacuum cleaner as shown in fig 5.44.

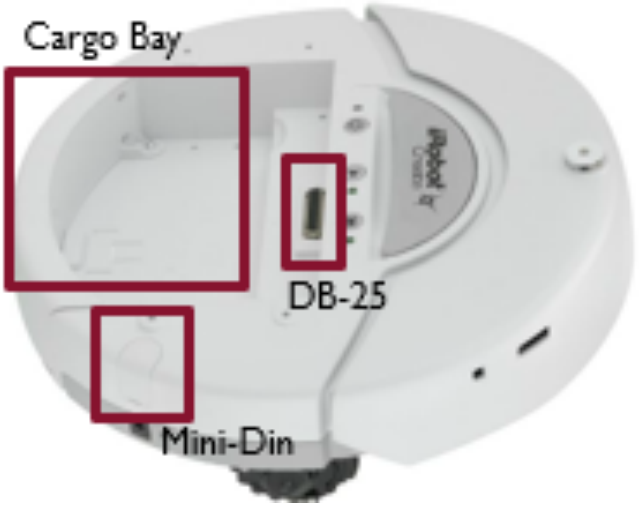


Fig 5.44: iRobot

DB-25 is mainly used to control the robot. It is controlled using a microcontroller. It is interfaced with it through UART communication. It sends the necessary commands to control the robot according to the requirement. DB-25 is shown in fig 5.45.

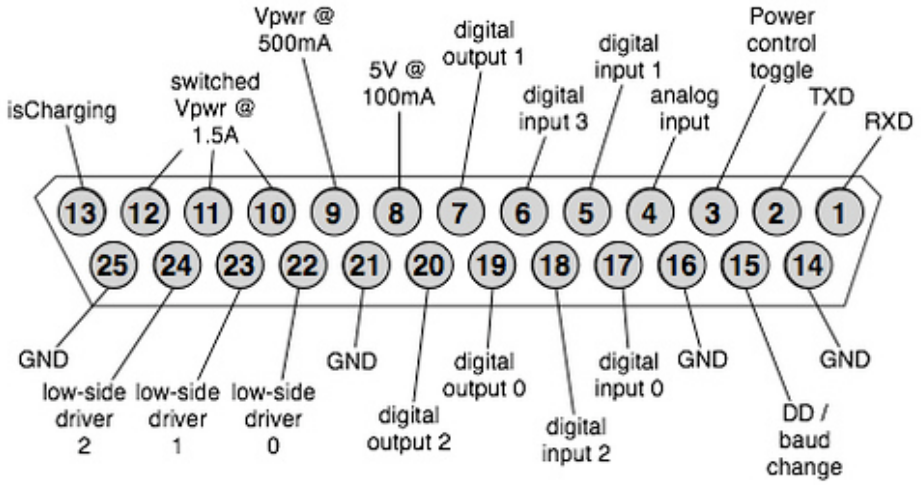


Fig 5.45: DB-25

There are many open interface commands of iRobot that the robot understands and according to these commands the robot can be programmed according to the application. Few of the open interface commands of iRobot are given below:

1) 128: Start command

2) 131: Safe

3) 132: Full

4) 137: Drive

5)145: Drive Direct

One example on how these commands are send into iRobot is shown below. This example is to drive the iRobot in a square.

152 17 137 1 44 128 0 156 1 144 137 1 44 0 1 157 0 90 153 (Drive in a Square)

- Script 152
- Number of bytes 17
- Drive 137
- 300 mm/s 1 44
- Straight 128 0

- Wait for Distance 156
- 400 mm 1 144
- Drive 137
- 300 mm/s 1 44
- Spinning counter clockwise 0 1
- Wait for Angle 157
- 90 degrees 0 90
- Restart Script 153

5.6.4 Controlling iRobot in different directions

iRobot can be controlled in different directions. Rotations are also possible. This is done by sending the above mentioned above to the robot. The possible rotations possible are the following:

1) Rotating 45 degrees in clockwise and anticlockwise direction

2) Rotating 90 degrees in clockwise and anticlockwise direction

3) Rotating 135 degrees in clockwise and anticlockwise direction

4) Rotating 180 degrees in clockwise and anticlockwise direction

5) Rotating 270 degrees in clockwise and anticlockwise direction

6) Rotating 360 degrees in clockwise and anticlockwise direction

The possible directions the iRobot can be controlled is the following:

1) Diagonal right, Diagonal left movement

2)Forward movement

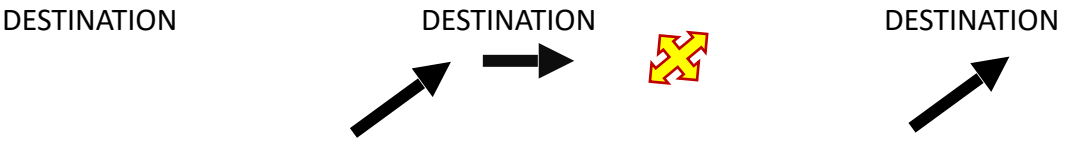
3)Backward movement

4)Right movement

5)Left movement

5.6.5 Implementation of TD in Mbed microcontroller for a 4X4 grid

With the help of app, the number of obstacles and position of obstacles is communicated to Mbed microcontroller. TD algorithm is coded in Mbed microcontroller for 1 to 5 obstacles. TD algorithm works according to the priority of movements. Some of the obstacle occurrence scenarios tested in hardware with one obstacle is shown in fig 5.46, with two obstacles is shown in fig 5.47, with three obstacles is shown in fig 5.48, with four obstacles is shown in fig 5.49 and with five obstacles is shown in fig 5.50. The black lines indicates the path travelled and the yellow cross indicates the static obstacles. In fig 5.46, picture 1 has obstacle in (2.5,2.5), picture 2 has obstacle in (3.5,3.5) and picture 3 has obstacle in (1.5,4.5). In fig 5.47, picture 1 has two obstacles in (1.5,2.5) and (2.5,2.5), picture 2 has two obstacles in (2.5,2.5) and (3.5,3.5) and picture 3 has two obstacles in (2.5,1.5) and (2.5,2.5). In fig 5.48, picture 1 has three obstacles in (3.5,1.5), (3.5,2.5) and (3.5,3.5), picture 2 has three obstacles in (1.5,3.5), (2.5,3.5) and (3.5,3.5) and picture 3 has three obstacles in (2.5,3.5), (3.5,3.5) and (3.5,4.5). In fig 5.48, picture 1 has three obstacles in (3.5,1.5), (3.5,2.5) and (3.5,3.5), picture 2 has three obstacles in (1.5,3.5), (2.5,3.5) and (3.5,3.5) and picture 3 has three obstacles in (2.5,3.5), (3.5,3.5) and (3.5,4.5). In fig 5.49, picture 2 has four obstacles in (2.5,1.5), (2.5,2.5), (3.5,3.5) and (2.5,3.5), picture 3 has four obstacles in (1.5,2.5), (2.5,2.5), (3.5,3.5) and (3.5,2.5) and picture 1 has four obstacles in (1.5,4.5), (2.5,1.5), (2.5,2.5) and (3.5,3.5). In fig 5.49, picture 1 has five obstacles in (2.5,1.5), (2.5,2.5), (3.5,3.5), (4.5,3.5) and (2.5,3.5), picture 2 has five obstacles in (1.5,2.5), (2.5,2.5), (3.5,3.5), (2.5,3.5) and (3.5,2.5) and picture 3 has five obstacles in (1.5,2.5), (2.5,1.5), (2.5,2.5), (2.5,3.5) and (3.5,3.5).



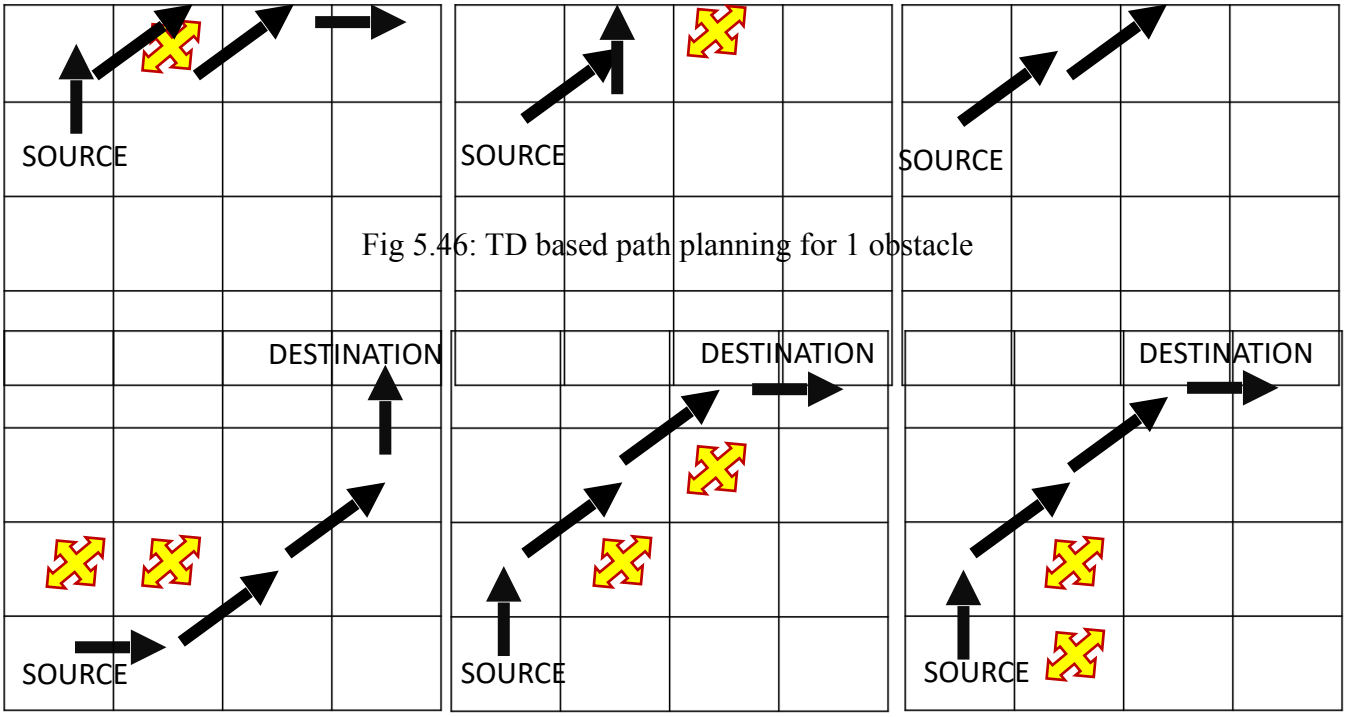


Fig 5.46: TD based path planning for 1 obstacle

Fig 5.47: TD based path planning for 2 obstacles

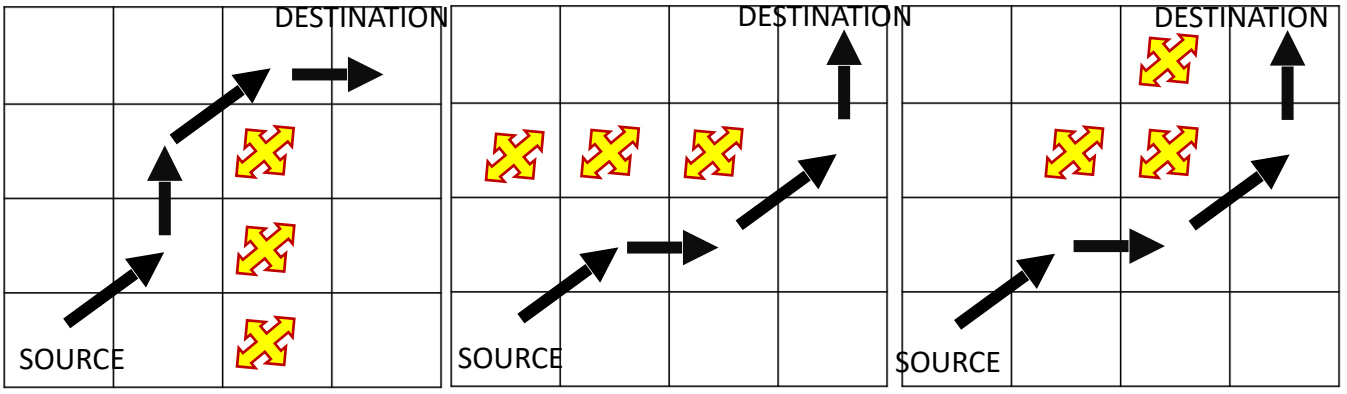


Fig 5.48: TD based path planning for 3 obstacles

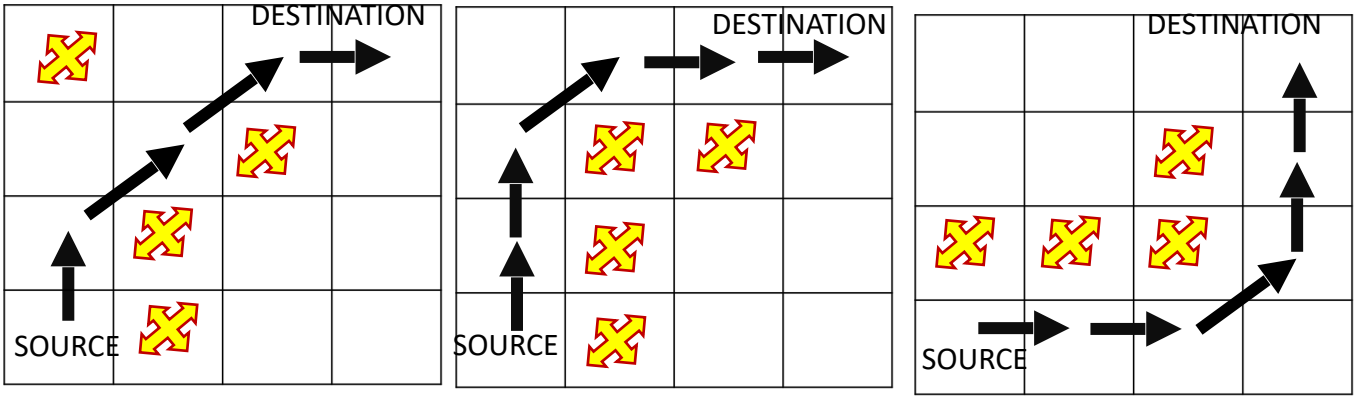


Fig 5.49: TD based path planning for 4 obstacles

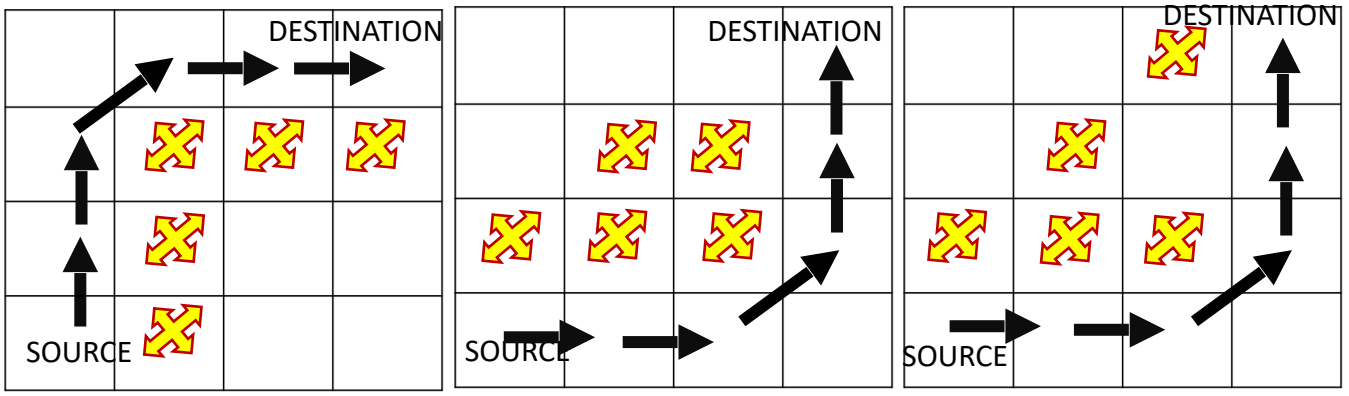


Fig 5.50: TD based path planning for 5 obstacles

5.6.6 Tried TD algorithm for dynamic obstacles

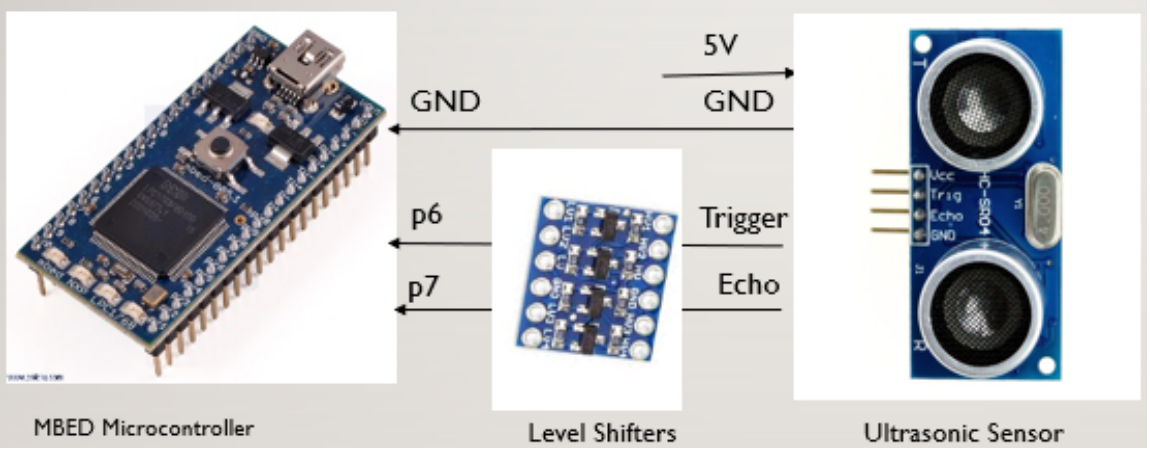


Fig 5.51: TD algorithm with dynamic obstacles

In order to detect dynamic obstacles, an ultrasonic sensor is interfaced with Mbed microcontroller as shown in fig 5.51. Level shifters are used between Mbed and ultrasonic sensor because 5V required for ultrasonic sensors is levelled down as 3.3V for Mbed microcontroller. When a dynamic obstacle is detected in the path, its deviates from the previously obtained optimal path and sets the grid at which dynamic obstacle is found as source point. Then TD algorithm is done once more to

find the optimal path from the new source point and destination point. Some of the dynamic obstacle scenarios are tested and are shown in fig 5.52. The black arrows indicate first path taken

before encountering a dynamic obstacle. The green arrows indicate the path taken after a dynamic obstacle is seen. The dark brown cross symbols indicate static obstacle whereas the blue ones indicate dynamic obstacles. The source and destination points are fixed in all the scenarios. The source point is at (1.5,1.5) and the destination point is at (4.5,4.5).

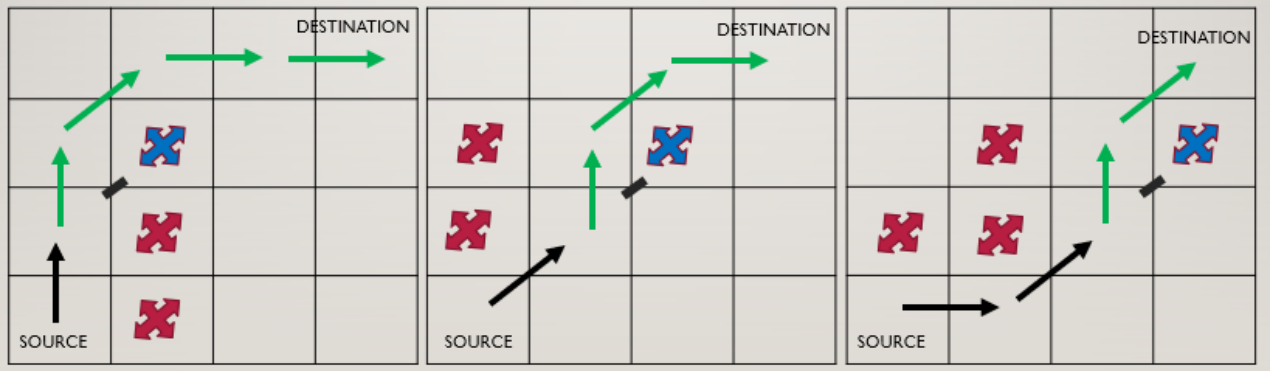


Fig 5.52: Path planning with TD algorithm for dynamic obstacles

5.6.7 Interfaced LCD with Mbed microcontroller

In order to display a message after the robot has reached the destination, an LCD is interfaced with Mbed microcontroller. Also, the time taken and distance travelled by the robot is displayed in the LCD after the message is displayed. The connection diagram is shown in fig 5.53. The pictures of the hardware are shown in fig 5.54, fig 5.55 and fig 5.56. Fig 5.54 shows the message displayed after the robot reaches the destination. Fig 5.55 shows the time taken by the robot to reach the destination. Fig 5.56 shows the distance travelled by the robot from the source point to the destination point.

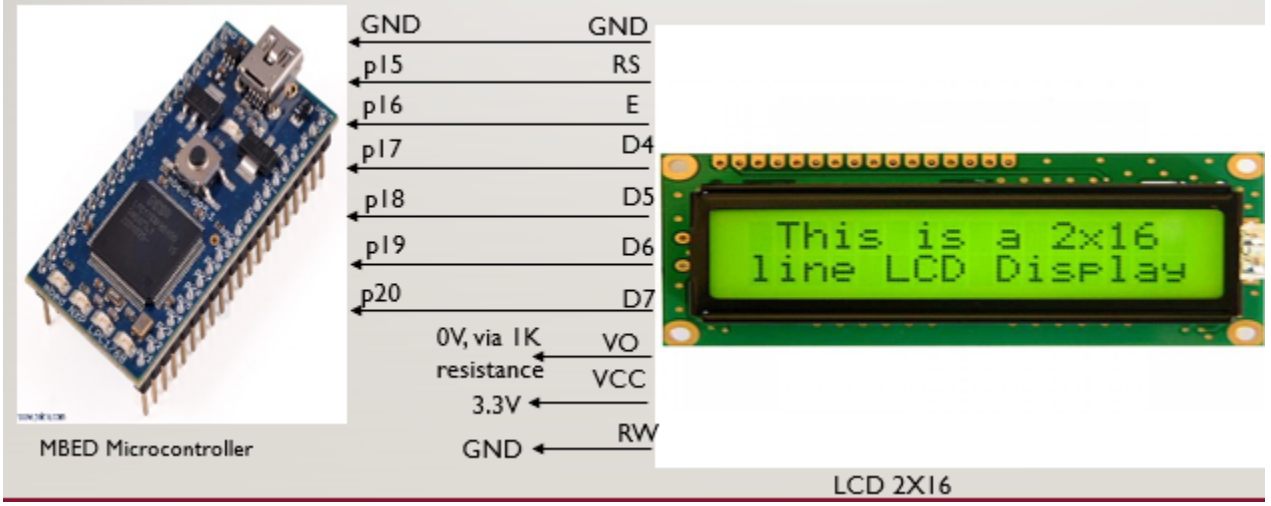


Fig 5.53: LCD interfaced with Mbed microcontroller

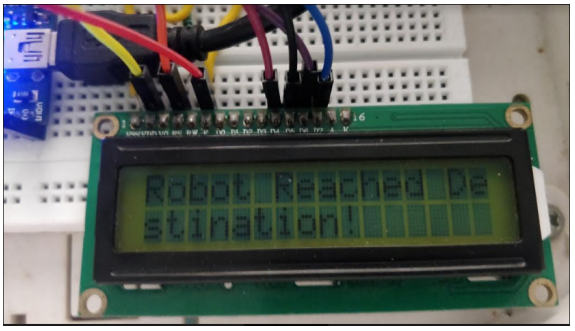


Fig 5.54: Message displayed after robot reaches the destination

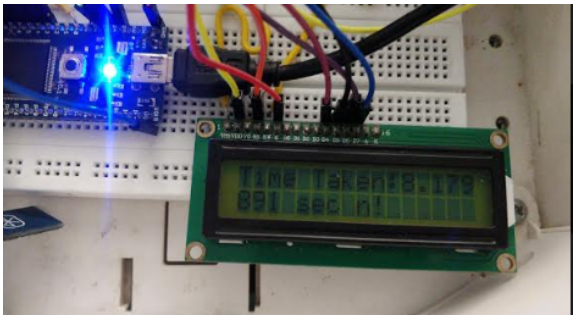


Fig 5.55: Time taken by the robot to reach the destination

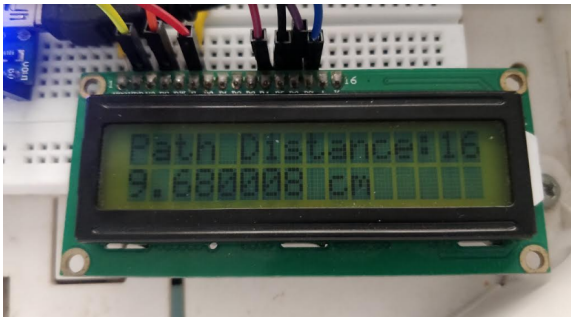


Fig 5.56: Distance travelled by the robot

5.6.8 Incorporated IoT concepts into project

After the robot reaches the destination, Mbed microcontroller communicates to Node MCU. After that Node MCU updates to thingspeak.com that the robot has reached the destination. A high

signal i.e. 1 will be send when the robot reaches destination or else a low signal i.e. 0 will be send to thingspeak.com. The system layout is shown in fig 5.57. After the robot reaches the destination, the output can be viewed in the laptop connected to the same hotspot that of Node MCU. The screenshot of output of thingspeak.com is shown in fig 5.58. Here, the high signal indicates that the robot has reached the destination and the low signal indicated that the robot is travelling to reach the destination.

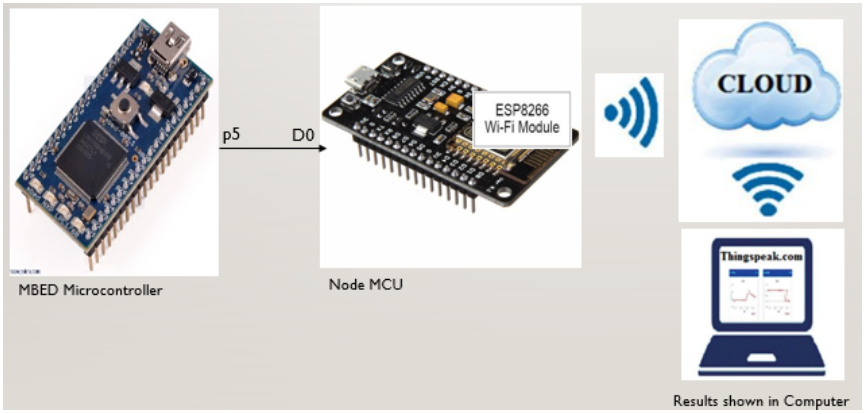


Fig 5.57: System layout of how IoT concepts were incorporated

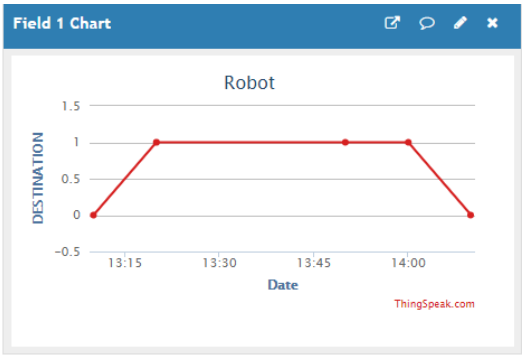


Fig 5.58: Screenshot of thingspeak.com

The final hardware setup after integrating the whole code which includes the app, TD algorithm, iRobot control, dynamic obstacles, LCD and IoT concepts is shown in fig 5.59.

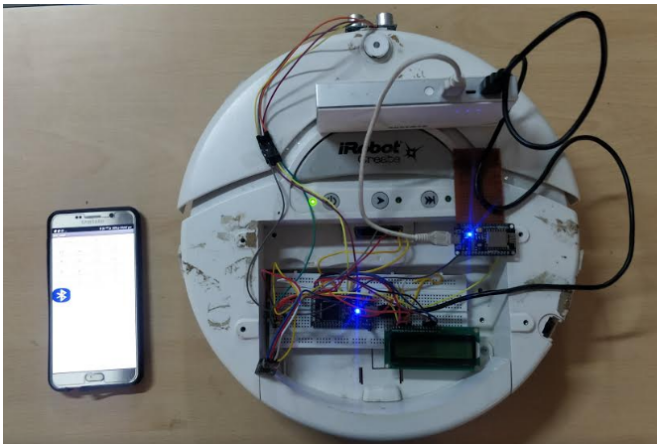


Fig 5.59: Final Hardware Setup of TD algorithm

5.7 IMPLEMENTATION OF SARSA ALGORITHM IN IROBOT FOR A 4X4 GRID

After implementing TD algorithm in Mbed, SARSA algorithm is implemented. The system layout is similar of that of before, the only difference is the algorithm coded within. SARSA algorithm was implemented for 1 to 5 obstacles. Fig 5.60 shows SARSA based path planning for 1 obstacle. Fig 5.61 shows SARSA based path planning for 2 obstacles. Fig 5.62 shows SARSA based path planning for 3 obstacles. Fig 5.63 shows SARSA based path planning for 4 obstacles. Fig 5.64 shows SARSA based path planning for 5 obstacles. In fig 5.60, picture 1 has one obstacle at (2.5,2.5), picture 2 has one obstacle at (2.5,3.5) and picture 3 has one obstacle at (3.5,3.5). In fig 5.61, picture 1 has two obstacles at (2.5,2.5) and (1.5,2.5), picture 2 has two obstacles at (1.5,3.5) and (3.5,3.5) and picture 3 has two obstacles at (2.5,1.5) and (2.5,2.5). In fig 5.62, picture 1 has three obstacles at (3.5,1.5), (3.5,2.5) and (3.5,3.5), picture 2 has three obstacles at (1.5,3.5), (2.5,3.5) and (3.5,3.5) and picture 3 has three obstacles at (2.5,3.5), (3.5,3.5) and (3.5,2.5). In fig 5.63, picture 1 has four obstacles at (1.5,2.5), (2.5,2.5), (3.5,3.5) and (3.5,2.5), picture 2 has four

obstacles at (2.5,1.5), (2.5,2.5), (1.5,4.5) and (3.5,3.5) and picture 3 has four obstacles at (2.5,1.5), (2.5,2.5), (2.5,3.5) and (3.5,3.5). In fig 5.64, picture 1 has five obstacles at (2.5,1.5), (2.5,2.5), (2.5,3.5), (3.5,3.5) and (4.5,3.5), picture 2 has five obstacles at (1.5,2.5), (2.5,2.5), (3.5,2.5), (3.5,4.5) and (3.5,3.5) and picture 3 has five obstacles at (2.5,1.5), (2.5,2.5), (1.5,3.5), (1.5,4.5) and (3.5,3.5).

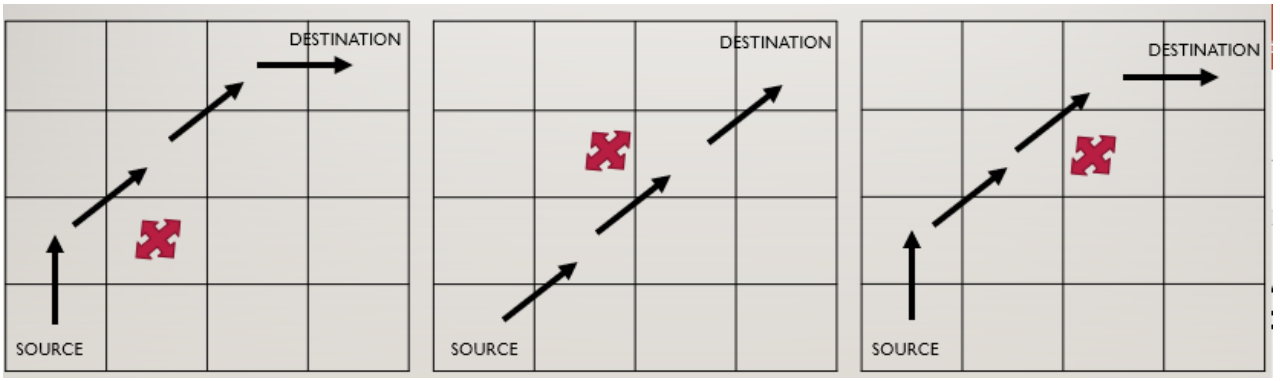


Fig 5.60: SARSA based path planning for 1 obstacle

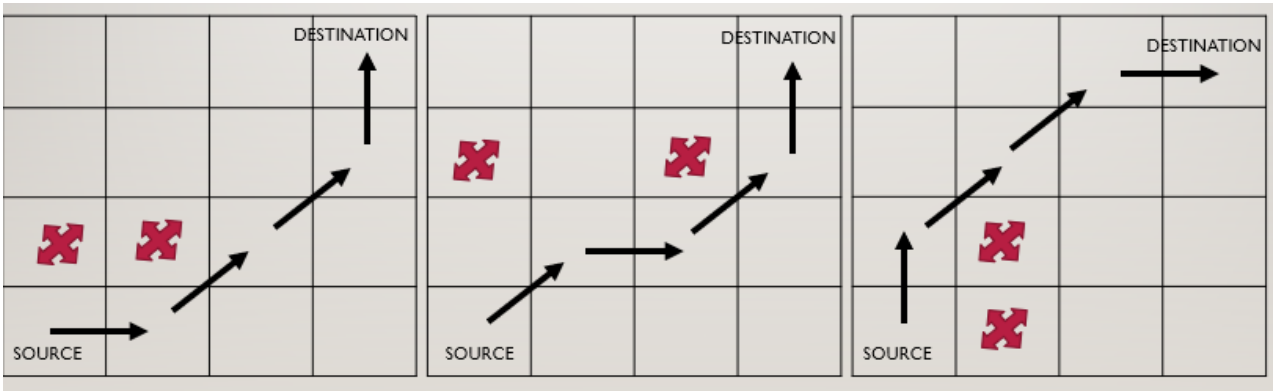


Fig 5.61: SARSA based path planning for two obstacles

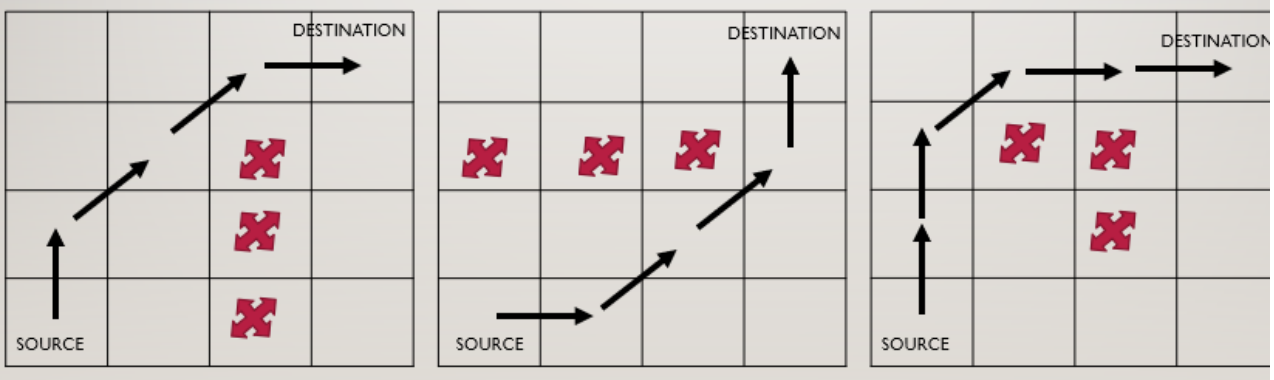


Fig 5.62: SARSA based path planning for three obstacles

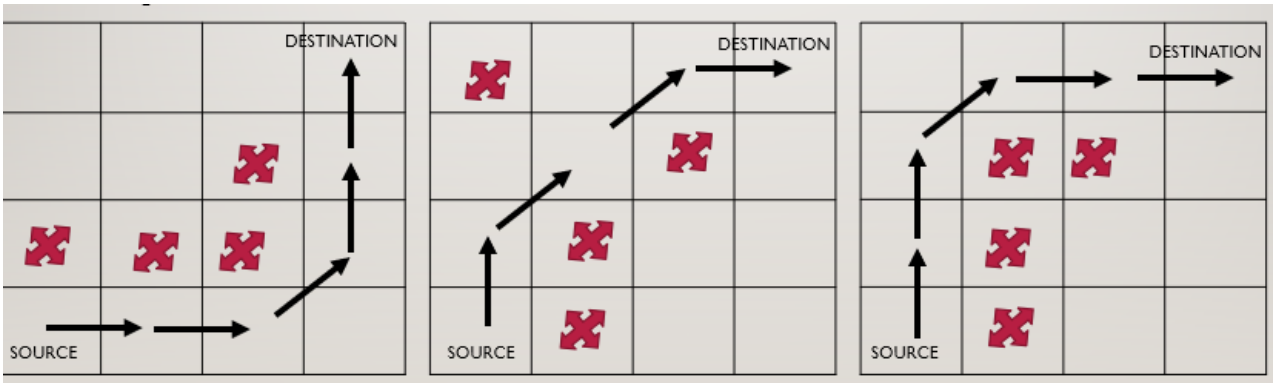


Fig 5.63: SARSA based path planning for four obstacles

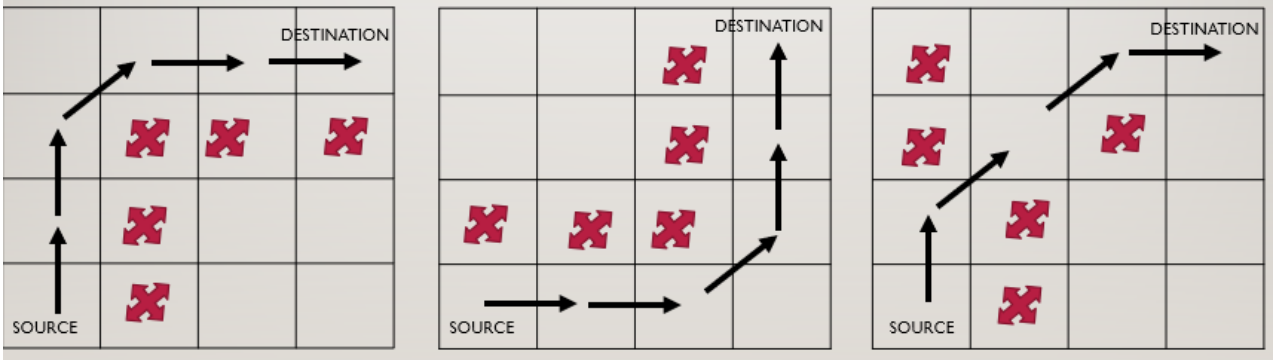


Fig 5.64: SARSA based path planning for five obstacles

5.7.1 SARSA algorithm with dynamic obstacles

SARSA algorithm was also tried for dynamic obstacles by interfacing ultrasonic sensor to Mbed microcontroller. When a dynamic obstacle is detected, the robot goes back to the initial position and update the grid where the dynamic obstacle was found and then does the path again in such a

way that it takes the optimal path. This is how the interaction with the environment factor helps to find the optimal path from the source point to destination point. SARSA based path planning for dynamic obstacles are shown in fig 5.65, fig 5.66 and fig 5.67. Fig 5.65 and fig 5.66 shows SARSA based path planning for dynamic obstacles with one or two static obstacles and one dynamic obstacles. Fig 5.67 shows SARSA based path planning for dynamic obstacles with two or three static obstacle and one dynamic obstacle. The black arrows indicate the path taken before encountering a dynamic obstacle. After encountering a dynamic obstacle, the robot goes back to its initial point and then takes the correct optimal path which is shown in green arrows. Static obstacles are represented as dark brown cross and dynamic obstacles are represented as blue cross.

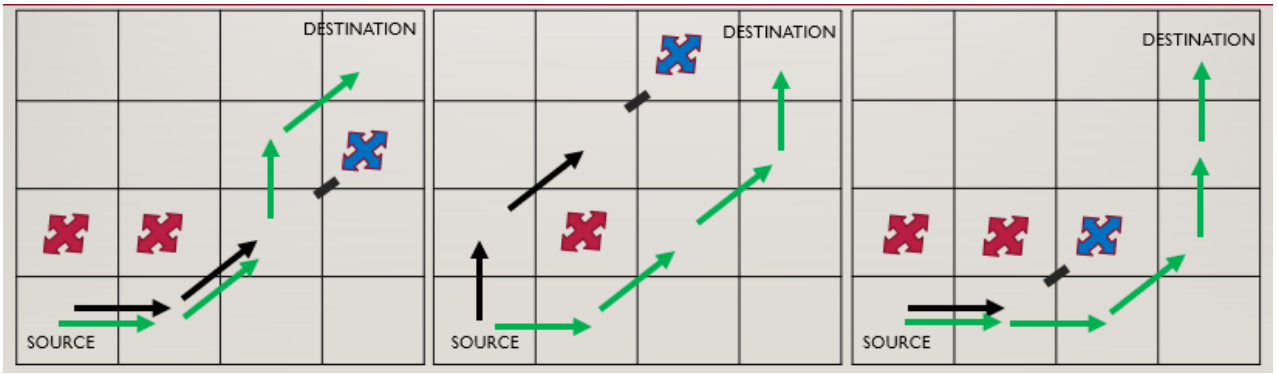


Fig 5.65: SARSA based path planning with dynamic obstacles with one or two static obstacles

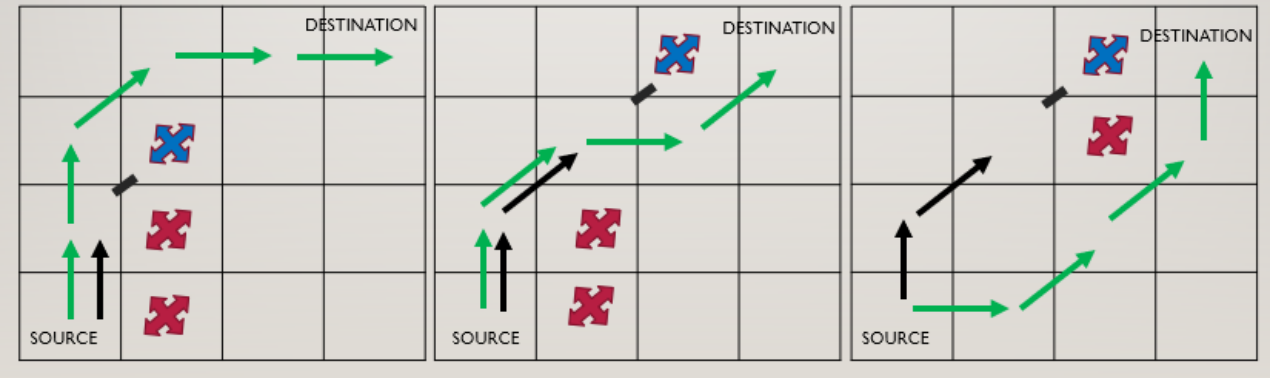


Fig 5.66: SARSA based path planning with dynamic obstacles with one or two static obstacles

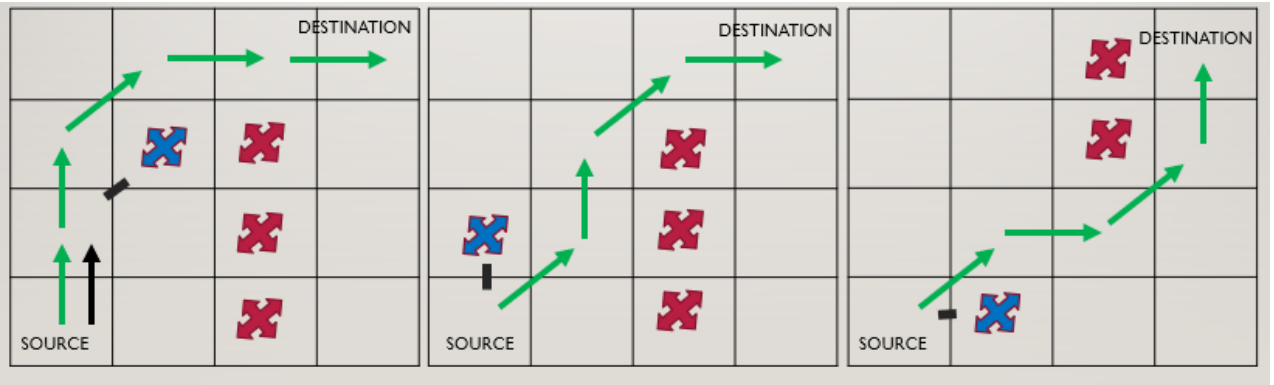


Fig 5.67: SARSA based path planning with dynamic obstacles with three or two static obstacles

5.8 COMPARISON BETWEEN SARSA AND TD ALGORITHM BASED ON SUCCESS RATE AND DISTANCE TRAVELLED IN HARDWARE

After the implementation of TD and SARSA algorithm, both the algorithm is compared based on the distance travelled in different scenarios and also based on the success rates.

5.8.1 Comparison between SARSA and TD algorithm based on success rate

After implementing SARSA and TD algorithm in hardware, both the algorithms are compared based on their success rates. There are cases where SARSA works properly and TD fails which were tested in software and now these cases are tested in hardware as well. Three such cases are shown in fig 5.68, fig 5.69 and fig 5.70. In fig 5.65, case 1 is shown. Here, the position of obstacles are (3.5,2.5), (3.5,3.5) and (3.5,4.5). Picture 1 indicates that TD fails in this case and picture 2 indicates that SARSA works in this case. The dark brown square box indicates that, in TD algorithm, after the robot gets stucked, it keeps on moving in that particular area and finally hits the obstacles. That is how TD fails in this condition. In fig 5.69, case 2 is shown. Here, the position of obstacles is (2.5,3.5), (3.5,3.5) and (4.5,3.5). Picture 1 indicates that TD fails in this case and picture 2 indicates that SARSA works in this case. In fig 5.70, case 3 is shown. Here, the

position of obstacles is (2.5,3.5), (3.5,3.5) and (3.5,2.5). Picture 1 indicates that TD fails in this case and picture 2 indicates that SARSA works in this case.

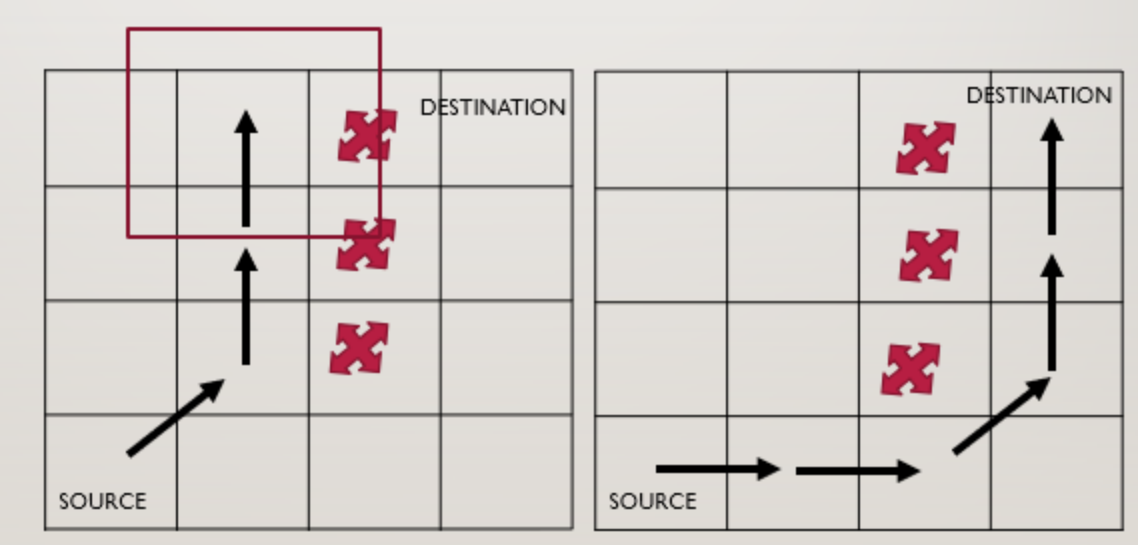


Fig 5.68: Case 1 where SARSA works and TD fails

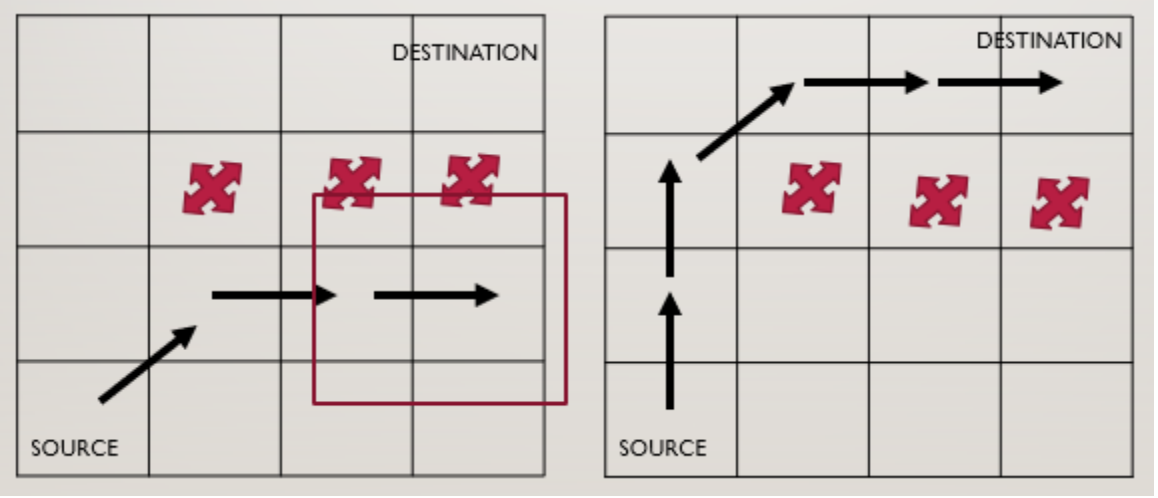


Fig 5.69: Case 2 where SARSA works and TD fails

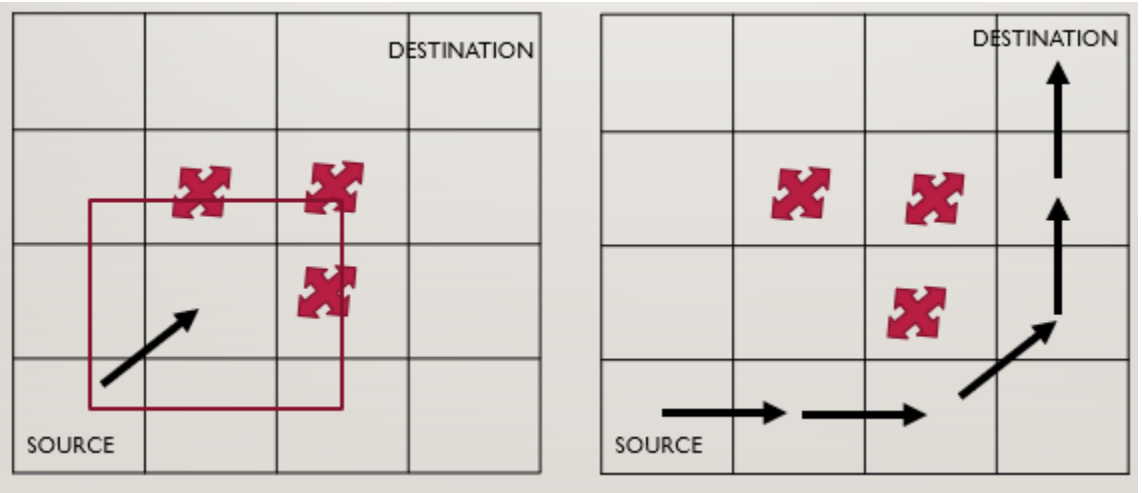


Fig 5.70: Case 3 where SARSA works and TD fails

5.8.2 Comparison between SARSA and TD based on the distance travelled

For SARSA algorithm, after it detects a dynamic obstacle it comes to initial position and then does path planning ensuring that it always takes the optimal path but in case of TD, it does not come to be initial position, it stops the algorithm where it encounters a dynamic obstacle and runs the algorithm again considering the present grid as the source point and finds the optimal path to the destination. In this way, SARSA always takes the optimal path and TD might not always take the optimal path. One such example is shown in fig 5.71. Here, picture 1 represents how TD algorithm does path planning. The number of steps taken in this case is 5 and the distance travelled is 216.559 cm. Picture 2 represents how SARSA algorithm does path planning. The number of steps taken in this case is 4 and the distance travelled is 193.119 cm. The black arrows indicate the path taken before encountering a dynamic obstacle. After encountering a dynamic obstacle, the robot goes back to its initial point and then takes the correct optimal path which is shown in green arrows. Static obstacles are represented as dark brown cross and dynamic obstacles are represented as blue cross.

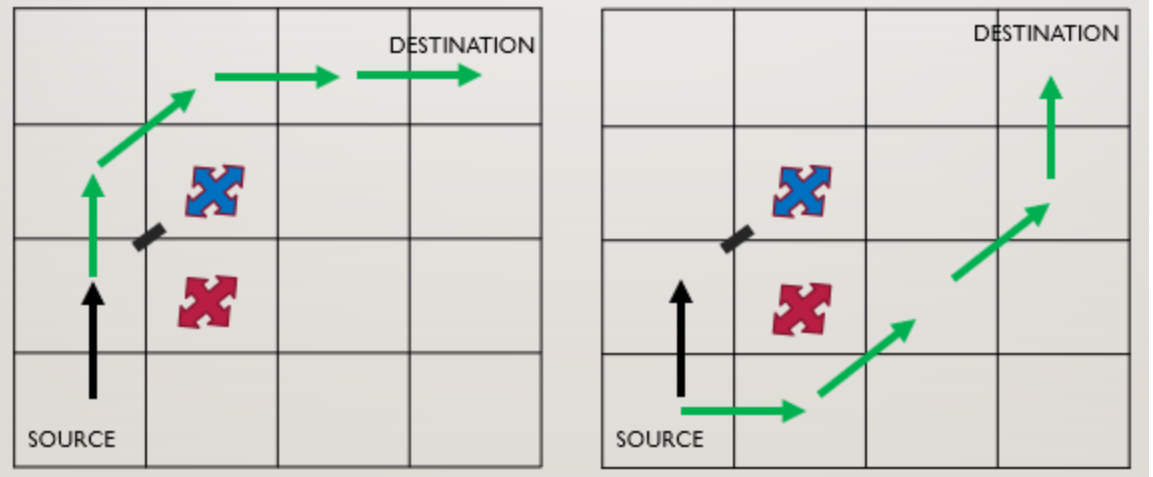


Fig 5.71: Comparison based on distance travelled between SARSA and TD algorithm

5.9 IMPLEMENTATION OF SARSA ALGORITHM IN IROBOT FOR AN 8X8 GRID

After implementing SARSA algorithm for a 4X4 grid, it was extended to 8X8 grid in iRobot.

5.9.1 Created an app for 8X8 grid using MIT APP Inventor

Similar to an app created for 4x4, an app for 8x8 was created using MIT app inventor as shown in fig 5.72. Up to 10 obstacles can be chosen and can be selected from the top section of the app numbered from 1 to 10. To select the position of the obstacles, the system layout of an 8x8 grid is shown below.

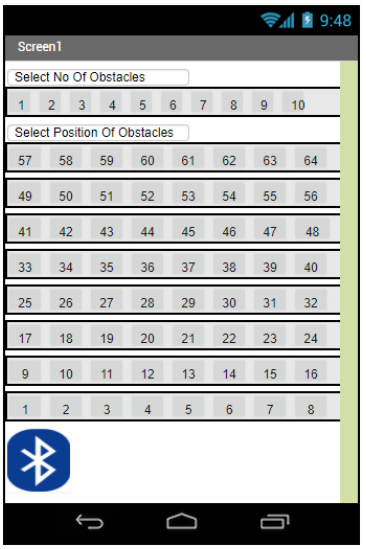


Fig 5.72: Screenshot of the app for 8x8 grid

5.9.2 Cases tested with 1 to 10 obstacles for SARSA algorithm

Different obstacle occurrence scenarios were tested for an 8X8 grid. Fig 5.73 shows the path planning for an 8X8 grid with one and two obstacles. The green box indicates the static obstacles. The arrows indicate the path travelled. SP is the source point. DP is the destination point. The number of steps travelled is also shown along with the arrows. The destination grid contains the total number of steps taken to reach there. Fig 5.74 shows the path planning for an 8X8 grid with three obstacles. Fig 5.75 shows the path planning for an 8X8 grid with four obstacles. Fig 5.76 shows the path planning for an 8X8 grid with five and six obstacles. Fig 5.77 shows the path planning for an 8X8 grid with seven obstacles. Fig 5.78 shows the path planning for an 8X8 grid with eight obstacles. Fig 5.79 shows the path planning for an 8X8 grid with nine obstacles. Fig 5.80 shows the path planning for an 8X8 grid with ten obstacles.

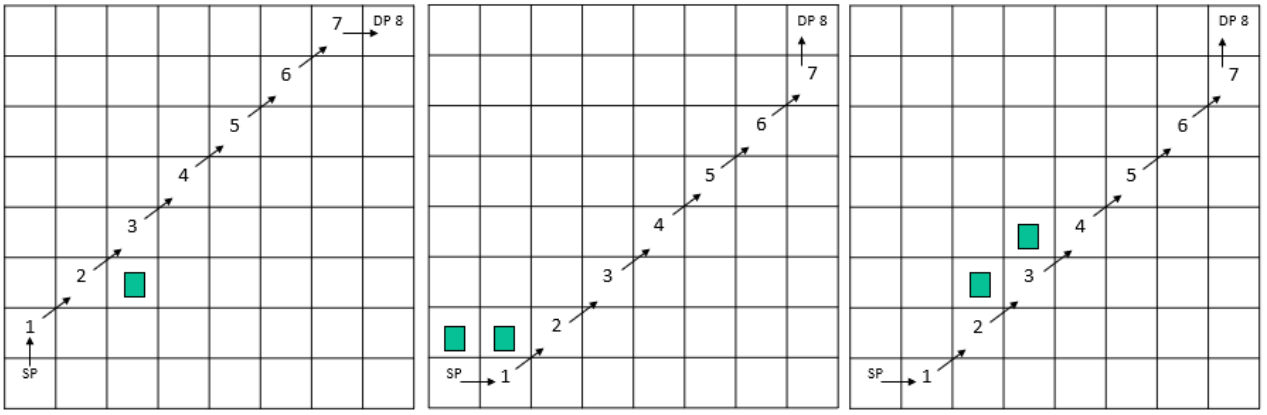


Fig 5.73: Path planning with one and two obstacles

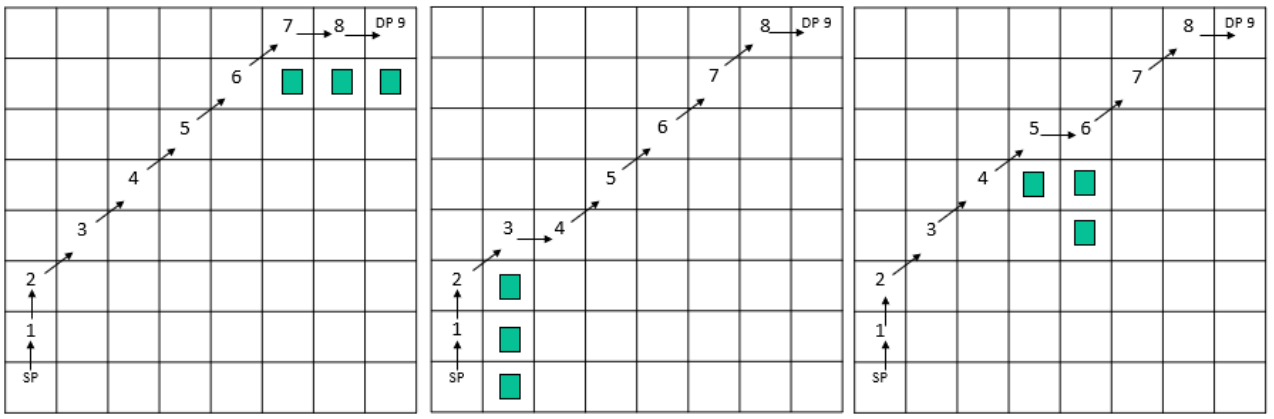


Fig 5.74: Path planning with three obstacles

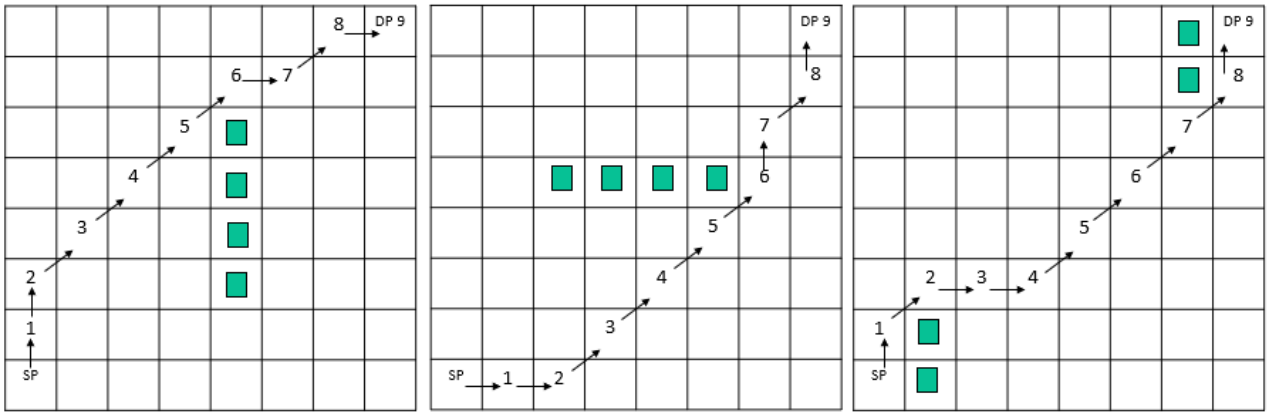


Fig 5.75: Path planning with four obstacles

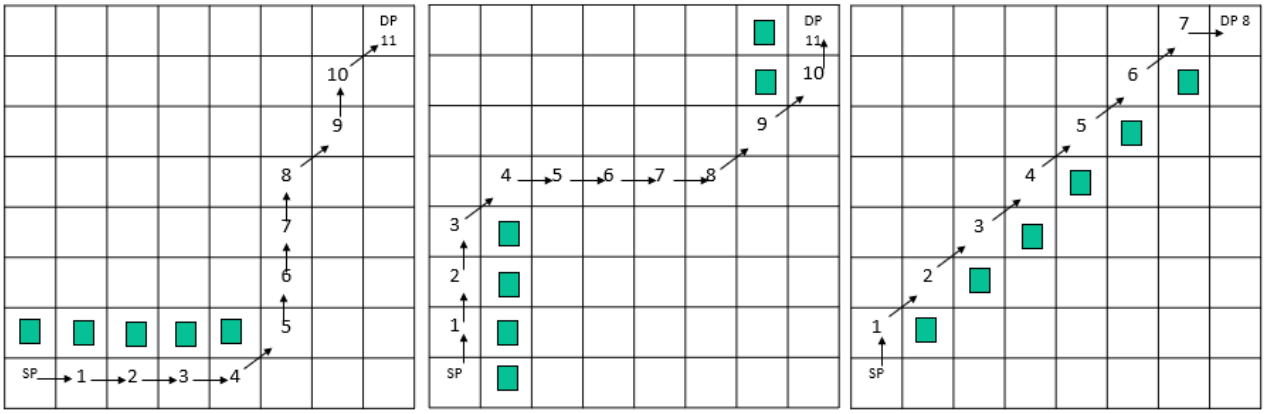


Fig 5.76: Path planning with five and six obstacles

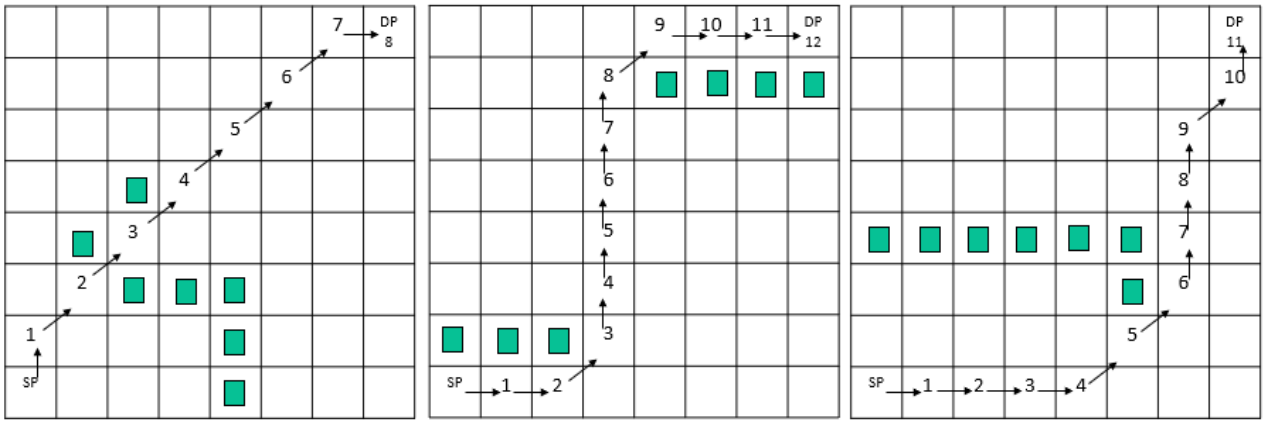


Fig 5.77: Path planning with seven obstacles

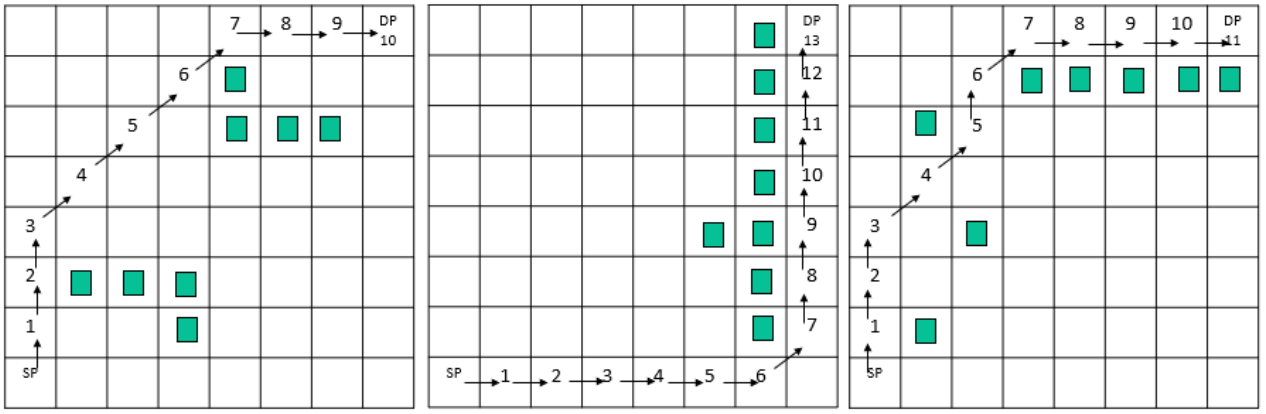


Fig 5.78: Path planning with eight obstacles 5.

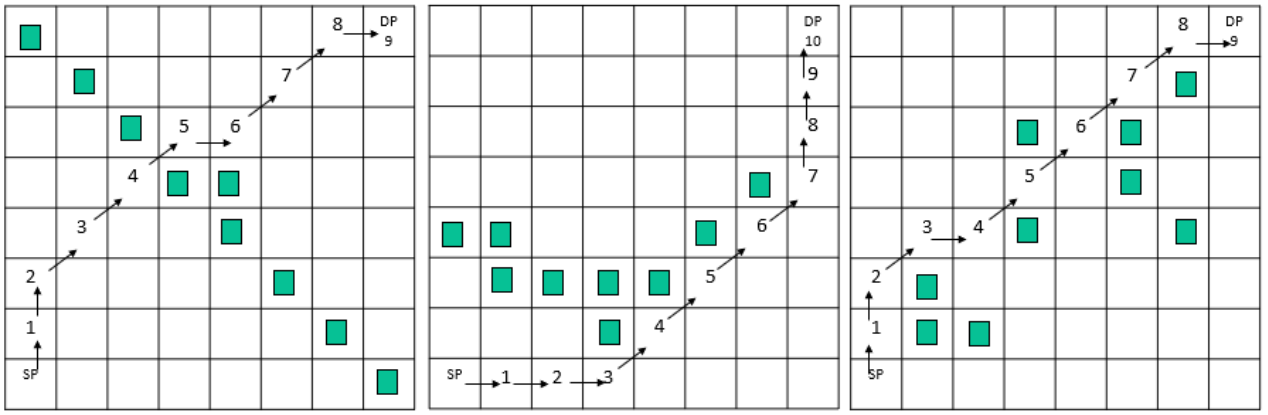


Fig 5.79: Path planning with nine obstacles

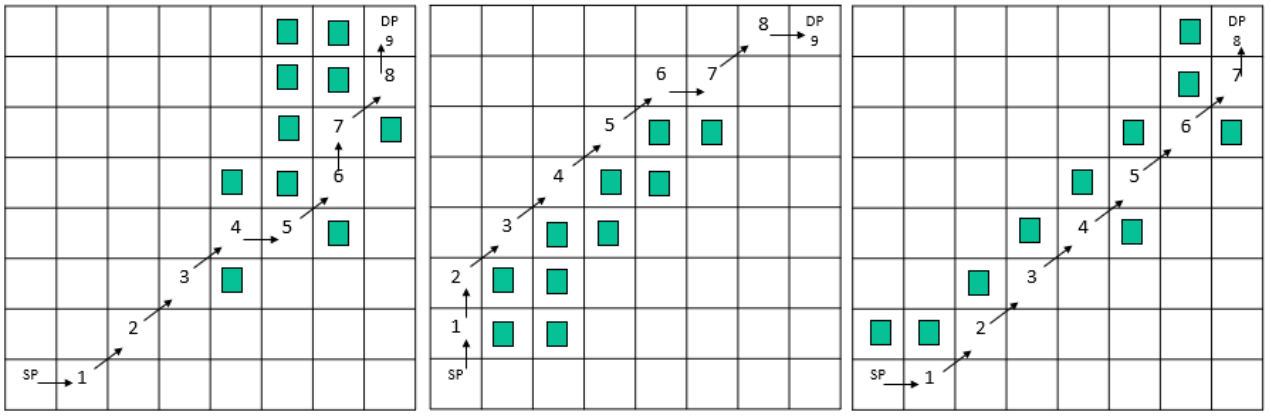


Fig 5.80: Path planning with ten obstacles

5.9.3 Cases tested with backward movement for an 8X8 grid

Different obstacle occurrence scenarios were tested for an 8X8 grid with backward movement. Fig 5.81 shows the path planning for an 8X8 grid with ten static obstacles. The green box indicates the static obstacles. The arrows indicate the path travelled. SP is the source point. DP is the destination point. The number of steps travelled is also shown along with the arrows. The destination grid contains the total number of steps taken to reach there. Fig 5.82 shows the path planning for an 8X8 grid with ten static obstacles and two dynamic obstacles. The yellow boxes indicate the dynamic obstacles. The red arrows indicate the path planning after encountering a dynamic obstacle. The first picture indicates the path planning with just static obstacle. The second picture indicates the path planning after encountering one dynamic obstacle and the third picture indicates the path panning after the encountering the second dynamic obstacle.

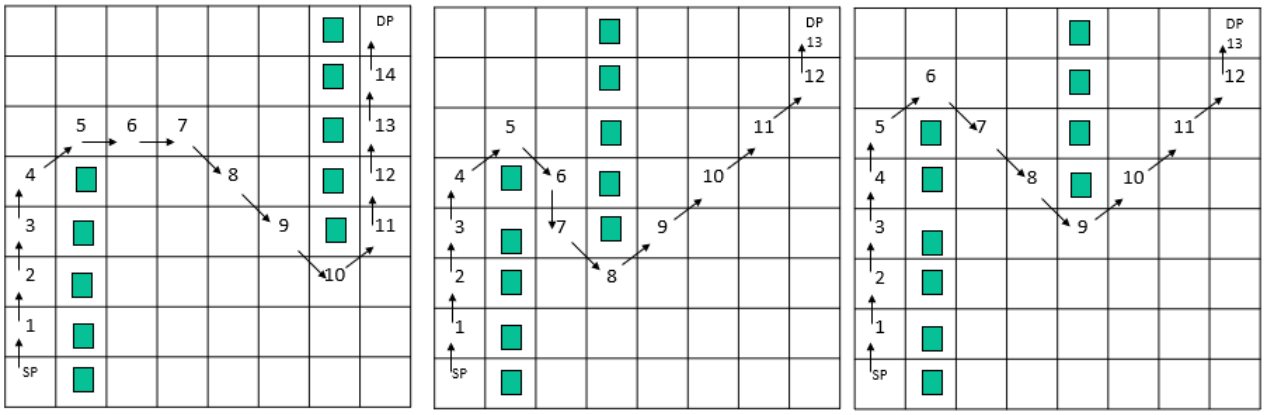


Fig 5.81: Path planning with backward movement for static obstacles

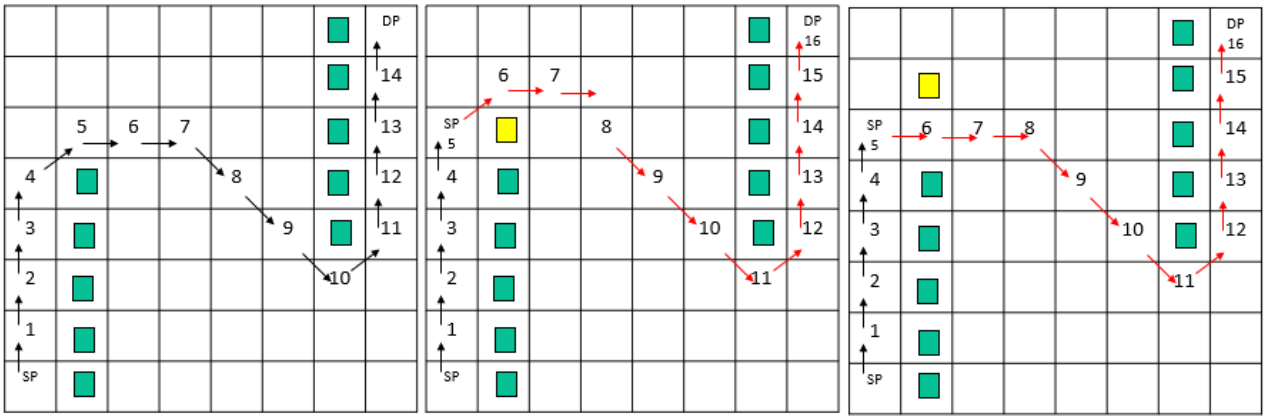


Fig 5.82: Path planning with backward movement for static and dynamic obstacles

5.9.4 Cases tested for more than one dynamic obstacles for an 8X8 grid

Different obstacle occurrence scenarios were tested for an 8X8 grid with backward movement. Fig 5.83 and fig 5.84 shows the path planning for an 8X8 grid with two static obstacles and three dynamic obstacles. The green box indicates the static obstacles. The arrows indicate the path travelled. SP is the source point. DP is the destination point. The number of steps travelled is also shown along with the arrows. The destination grid contains the total number of steps taken to reach there. The yellow boxes indicate the dynamic obstacles. In Fig 5.83, the first picture indicates the path planning with just static obstacle. The second picture indicates the path planning after encountering one dynamic obstacle and the third picture indicates the path panning after the encountering the second dynamic obstacle. Fig 5.84 indicates the path panning after the

encountering the third dynamic obstacle. Fig 5.85 and fig 5.86 shows the path planning for an 8X8 grid with two static obstacles and five dynamic obstacles. In fig 5.85, the first picture indicates the path planning with just static obstacle. The second picture indicates the path planning after encountering one dynamic obstacle and the third picture indicates the path panning after the encountering the second dynamic obstacle. In fig 5.86, the first picture indicates the path planning with the third dynamic obstacle. The second picture indicates the path planning after encountering fourth dynamic obstacle and the third picture indicates the path panning after the encountering the fifth dynamic obstacle.

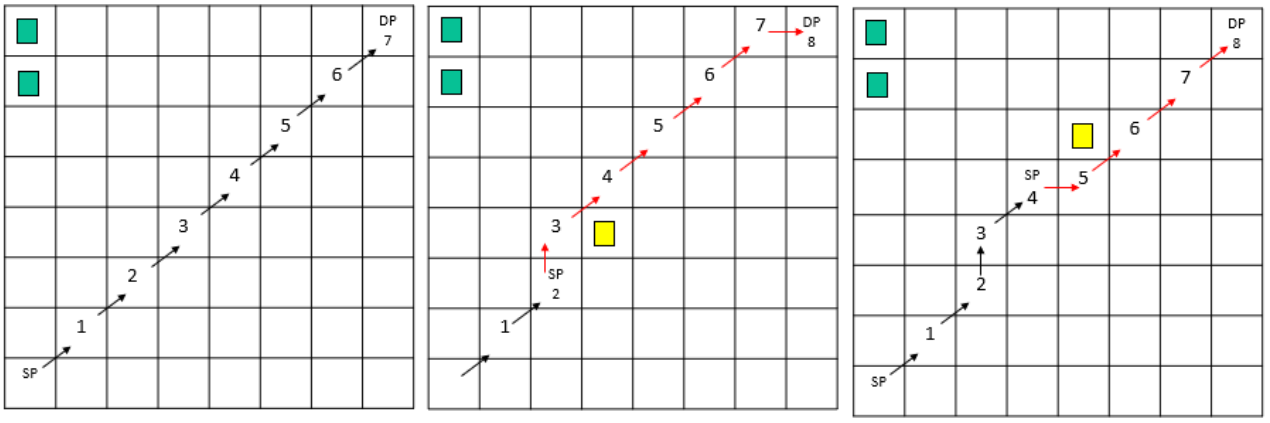


Fig 5.83: Case 1 with two static obstacles and first and second dynamic obstacles

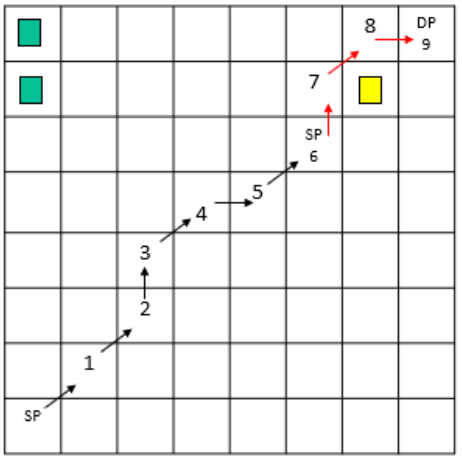


Fig 5.84: Case 1 with two static obstacles and third dynamic obstacles

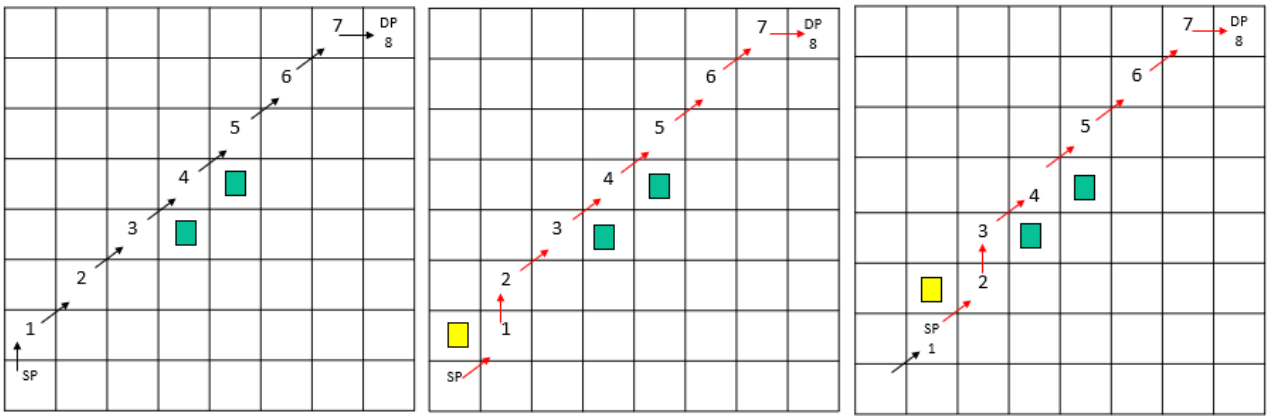


Fig 5.85: Case 2 with two static obstacles and first and second dynamic obstacles

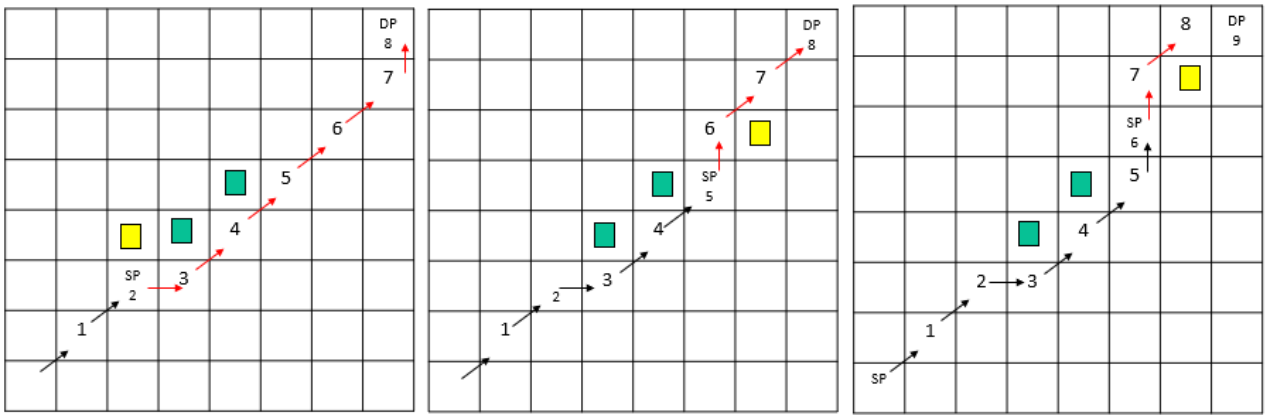


Fig 5.86: Case 2 with two static obstacles and third, fourth and fifth dynamic obstacles

5.9.5 Creating a GUI

For creating a GUI, ZigBee communication is used. There will be two ZigBee modules, one acting as a transmitter and another as receiver. For configuring these modules XCTU tool is used. After configuration one will act as a master and the another as slave. Then ZigBee communication is tried with one module interfaced with Mbed microcontroller and the other module serially connected with a pc as shown in fig 5.87.

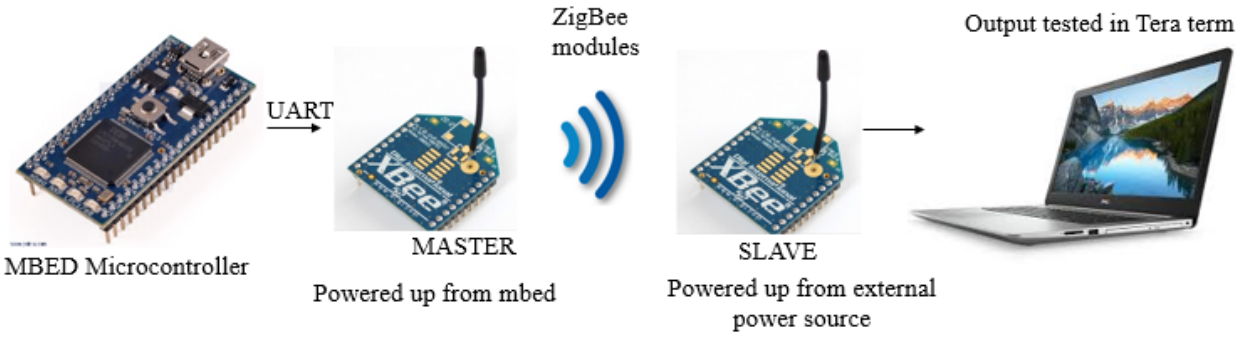


Fig 5.87: Communication between Mbed and pc

After obtaining the values from Mbed to pc, processing tool is used to create a GUI. Before ZigBee communication the GUI is as shown in fig 5.88. The green box indicates the source point and the blue box indicates the destination point. Once the ZigBee communication starts the GUI is as shown as in fig 5.89. The red boxes indicate the path taken from the source point to the destination point. The transmitter ZigBee module is as shown in fig 5.90 and the receiver ZigBee module is as shown in fig 5.91. The final iRobot model is as shown in fig 5.92.

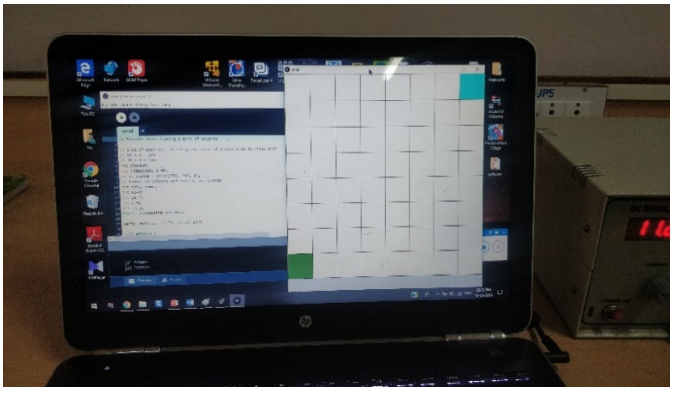


Fig 5.88: GUI results before communication

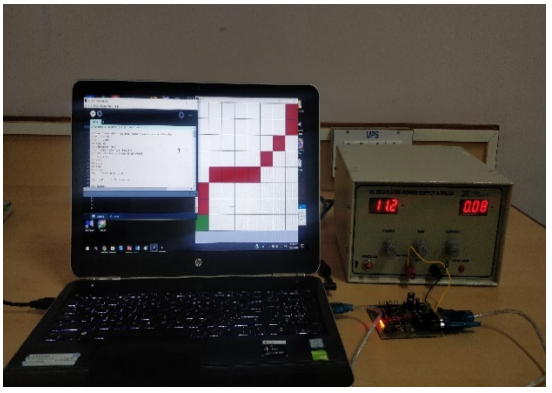


Fig 5.89: GUI results after communication

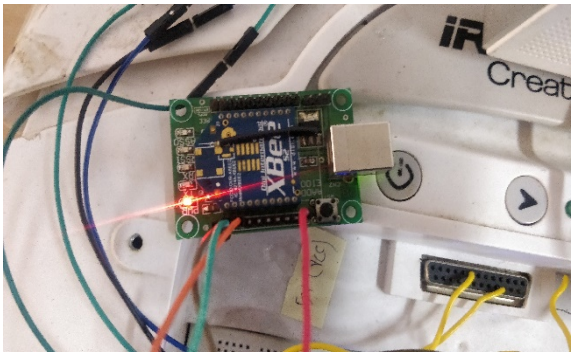


Fig 5.90: Transmitter Zigbee Module

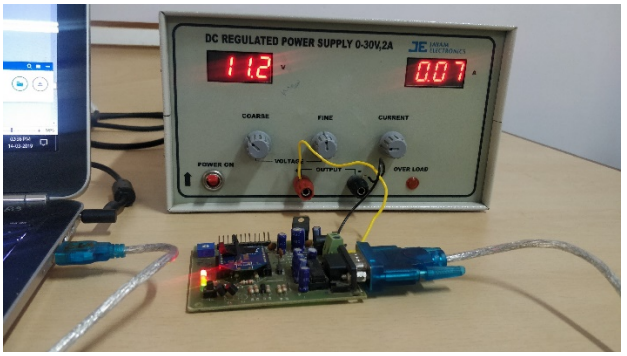


Fig 5.91: Receiver Zigbee Module

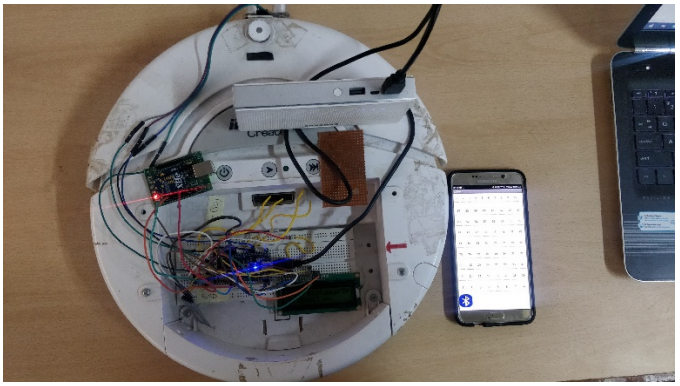


Fig 5.92: Hardware model of SARSA algorithm for an 8X8 grid

5.9.6 Interfaced LCD to Mbed microcontroller

To display the necessary details like distance travelled, time taken and a message indicating that the robot has reached the destination, an LCD is interfaced. Fig 5.93 shows the message displayed after the robot reaches the destination. Fig 5.94 and fig 5.95 shows the distance travelled and time taken for travel respectively.

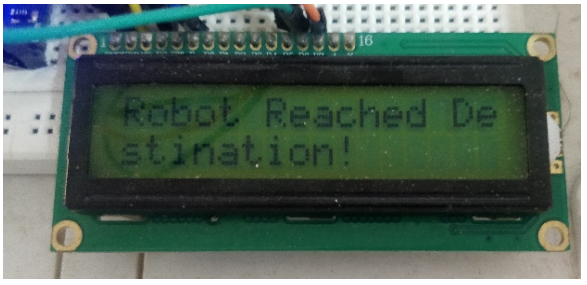


Fig 5.93: Message displayed in LCD

Fig 5.94: Distance travelled displayed in LCD Fig 5.95: Time taken displayed in LCD

An analysis of the time of computation and the time to reach the destination was done as shown in table 5.11. From the table its evident that the time increases as the number of obstacles, complexity of path planning and number of steps taken to reach the destination increases.

Table 5.11: Time of computation for different number of static obstacles

SL NO	No of Obstacles	Position of Obstacles	Time Taken (in sec)
1	0	-	38.95
2	1	**(1.5,1.5)**	45.31
3	2	**(1.5,1.5),(6.5,6.5)**	45.61
4	3	**(1.5,1.5),(1.5,2.5),(1.5,3.5)**	50.58
5	4	**(4.5,2.5),(4.5,3.5),(4.5,4.5),(4.5,5.5)**	50.98
6	5	**(1.5,0.5),(1.5,1.5),(1.5,2.5),(1.5,3.5),(1.5,4.5)**	55.09
7	6	**(1.5,0.5),(1.5,1.5),(1.5,2.5),(1.5,3.5),(6.5,7.5), (6.5,6.5)**	55.06
8	7	**(0.5,1.5),(1.5,1.5),(2.5,1.5),(7.5,6.5),(6.5,6.5), (5.5,6.5),(4.5,6.5)**	57.66
9	8	**(1.5,2.5),(2.5,2.5),(3.5,2.5),(3.5,1.5),(6.5,5.5), (5.5,5.5),(4.5,5.5),(4.5,6.5)**	58.22

10	9	**(0.5,3.5),(1.5,3.5),(1.5,2.5),(2.5,2.5),(3.5,2.5), (3.5,1.5),(4.5,2.5),(5.5,3.5),(6.5,4.5)**	56.39
11	10	**(1.5,0.5),(1.5,1.5),(1.5,2.5),(1.5,3.5),(1.5,4.5), (1.5,5.5),(1.5,6.5),(2.5,6.5),(3.5,6.5),(4.5,6.5)**	62.57

Also, an analysis of the time of computation with dynamic obstacles was carried out and is shown in table 5.12. It is understood that as the number of dynamic obstacles increase, the time of computation also increases.

Table 5.12: Time of computation for different number of dynamic obstacles

SL NO	No of Static Obstacles	Position Of Static Obstacles	No of Dynamic Obstacles	Position of Dynamic Obstacles	Time Taken (in sec)
1	0	(1.5,1.5)	0	-	38.95
2	1	(1.5,1.5)	1	**(0.5,1.5)**	45.31
3	2	(1.5,1.5)	**2**	**(0.5,1.5), (3.5,2.5)**	45.61
4	3	(1.5,1.5)	**3**	**(0.5,1.5), (3.5,2.5), (2.5,2.5)**	50.58
5	4	(1.5,1.5)	**4**	**(0.5,1.5), (3.5,2.5), (2.5,2.5), (4.5,3.5)**	50.98
6	5	(1.5,1.5)	**5**	**(0.5,1.5), (3.5,2.5), (2.5,2.5), (4.5,3.5), (4.5,4.5)**	55.09
7	6	(1.5,1.5)	**6**	**(0.5,1.5), (3.5,2.5), (2.5,2.5), (4.5,3.5), (4.5,4.5), (3.5,4.5)**	55.06
8	7	(1.5,1.5)	**7**	**(0.5,1.5), (3.5,2.5), (2.5,2.5), (4.5,3.5), (4.5,4.5), (3.5,4.5), (5.5,5.5)**	57.66

9	8	(1.5,1.5)	**8**	**(0.5,1.5), (3.5,2.5), (2.5,2.5), (4.5,3.5), (4.5,4.5), (3.5,4.5), (5.5,5.5), (5.5,6.5)**	58.22

5.10 IMPLEMENTATION OF ROBOT TO ROBOT COMMUNICATION FOR AN 7X7 GRID

Robot to robot communication is implemented using two iRobot named as robot 1 and robot2. The hardware setup of this application is a 7x7 grid as shown in fig 5.96. There will be three junctions named as J1, J2 and J3. There will be six destination points named as D1, D2, D3, D4, D5 and D6. The blue boxes indicate the path that cannot be taken. An app is created using MIT App Inventor to communicate the destination points to both the robots as shown in fig 5.97. The initial points of robot 1 and robot 2 is fixed. There can be cases where the robots might clash with each other, so robot to robot communication is enabled. Mbed modules are integrated with iRobot for controlling the robots. Communication between the Mbed modules is enabled with the help of WI-FI modules. For this, the Wi-Fi modules should be configured as servers with different port number and their IP address is found which is used communicate with each other. The screenshots of the AT commands are given in fig 5.98. Then Mbed to Mbed communication is done, then iRobot is controlled. The system layout of robot to robot communication is shown in fig 5.99.

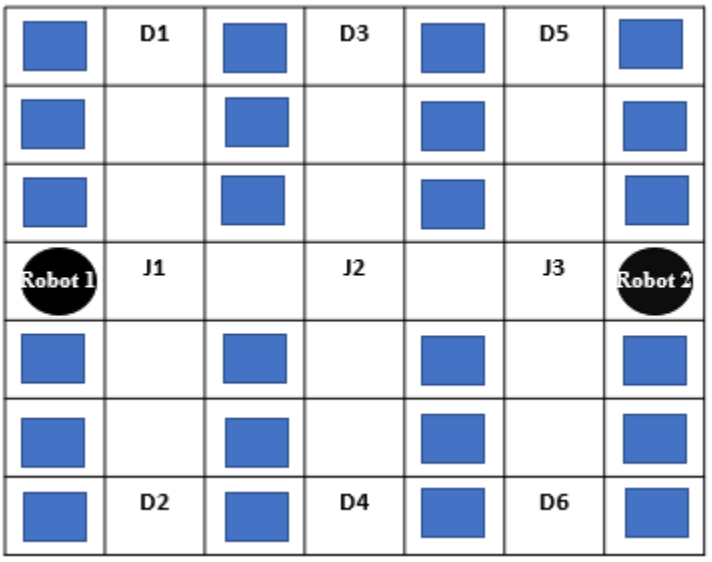


Fig 5.96: Layout of robot to robot communication

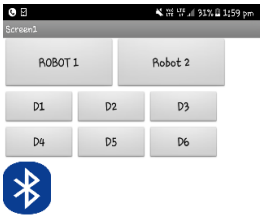


Fig 5.97: APP for robot to robot communication

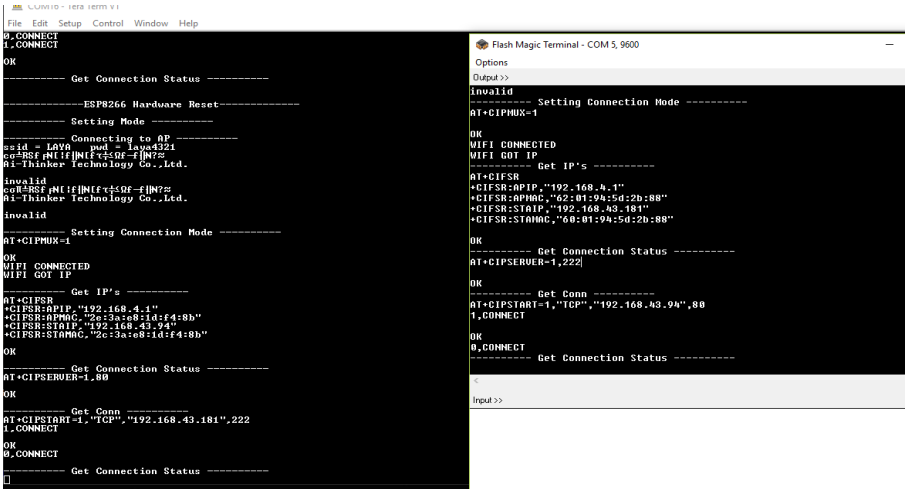


Fig 5.98: AT Commands

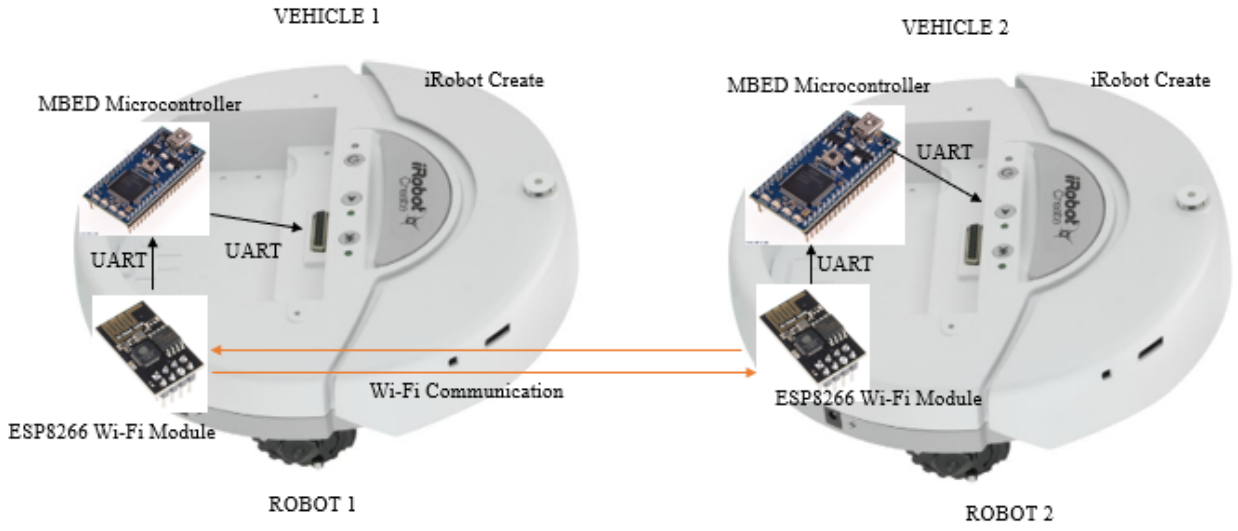


Fig 5.99: System Layout of robot to robot communication

In order to solve clashing of robots, serial interrupts are used. With the help of serial interrupts, robots can be solved even when they are moving. Stop commands and wait commands should be given in ISR and this solves the failed cases. Different possible scenarios of vehicle to vehicle communication is tested. Case 1 is depicted in fig 5.100, this is the scenario where both the robots reaches the junction together. In such a case the robot that's sends the notification first will move to the junction and the other one will wait for the robot to pass the junction. Case 2 is depicted in fig 5.101, this is the scenario where robot2 reaches the junction j1 early and the other robot just started moving, so robot2 will wait and robot 1 will continue its path planning. Case 3 is depicted in fig 5.102, this is the vice versa of case 2.

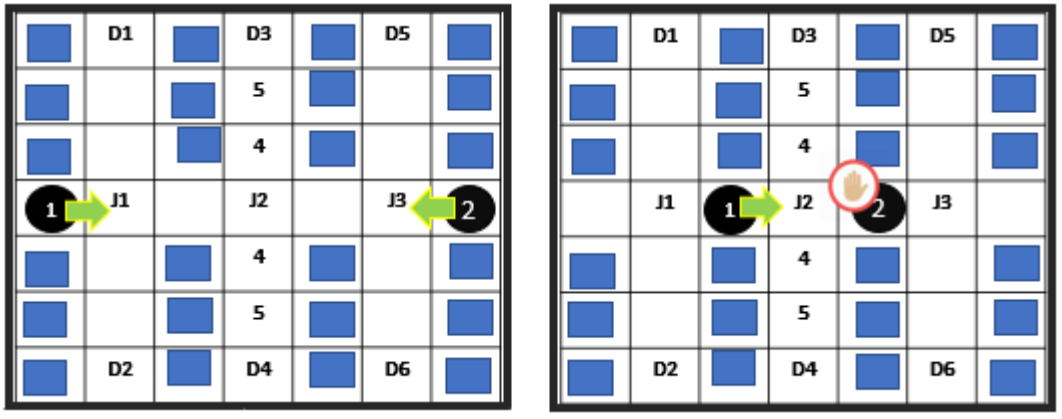


Fig 5.100: Case 1 where both the robots reach the junction earlier

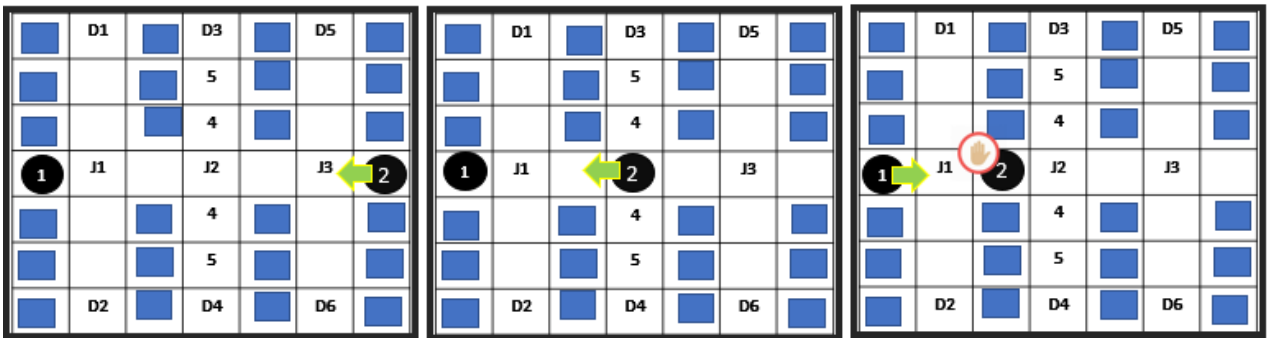


Fig 5.101: Case 2 where one of the robots reaches the junction early and the other robot is near the junction

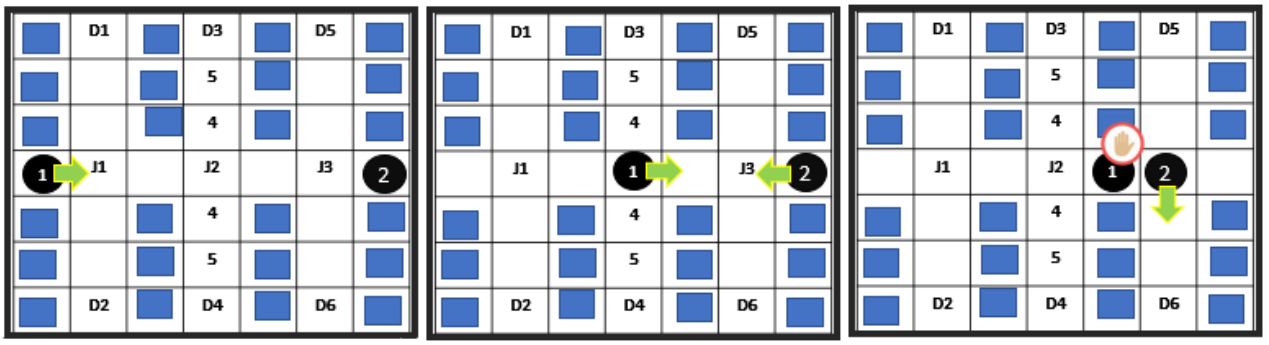


Fig 5.102: Case 3 where one of the robots reaches the junction early and the other robot is near the junction

The hardware setup of robot to robot communication is as shown in fig 5.103. The blue lines indicate the grid separations. The white tick rectangular objects separate the real path from the path that cannot be taken. The two round objects are the two vehicles i.e. iRobot. The two iRobot is shown in fig 5.104.

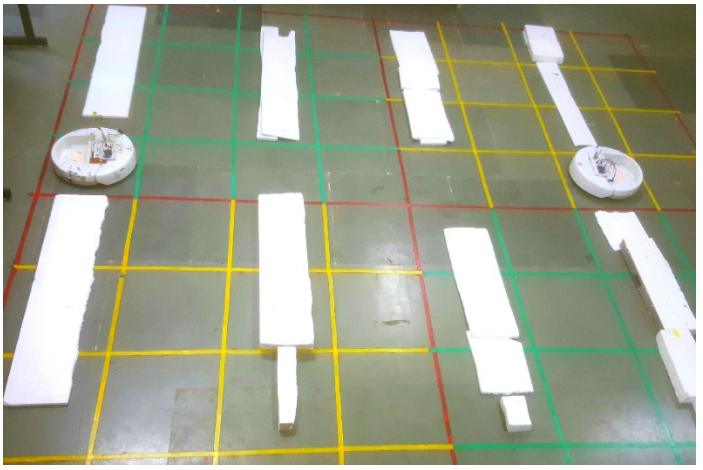


Fig 5.103: Hardware setup for robot to robot communication

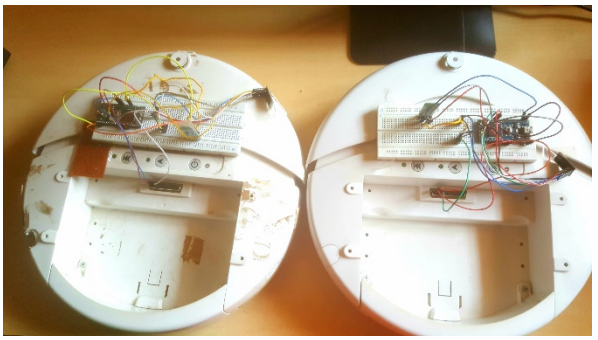


Fig 5.104: Robots

5.11 IMPLEMENTATION OF PATH PLANNING FOR A LOAD CARRYING ROBOT

Path planning is done for four rooms room 1, room 2, room3 and room4 as shown in fig 5.105. Each room has its own SP (source point) and DP (destination point). There will be one iRobot and it will navigate to the four rooms according to the information communicated to it. The initial point of the robot is the SP of room1. The size of each room is 4X4 grid. It is assumed that the load is carried from the initial point and then carried to respective destination points. The information about which room it should navigate is given through an app as shown in fig 5.106. The first horizontal arrangement contains the buttons to select which room to navigate. The second horizontal arrangement contains the buttons to select the number of obstacles and last

arrangements are to select the position of obstacles. Then app is communicated to mbed microcontroller through HC05 Bluetooth module.

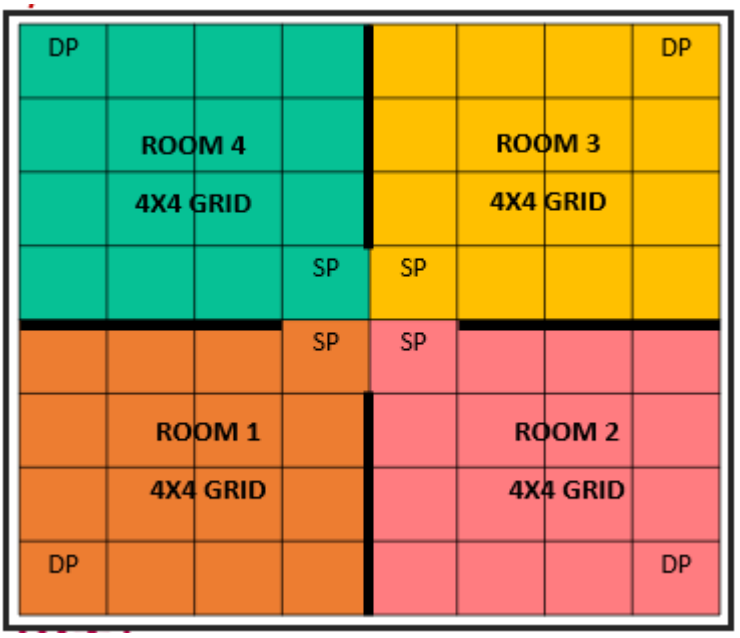


Fig 5.105: System Layout of a load carrying robot

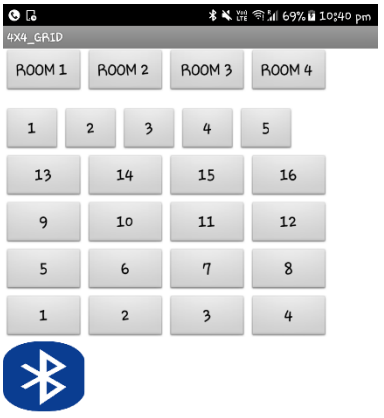


Fig 5.106: APP for a load carrying robot

Different obstacle occurrence scenarios were tested for all the rooms. Three cases for each room is mentioned here. Fig 5.107, fig 5.108, fig 5.109 and fig 5.110 shows the path planning cases tested for room1, room2, room3 and room 4 respectively.

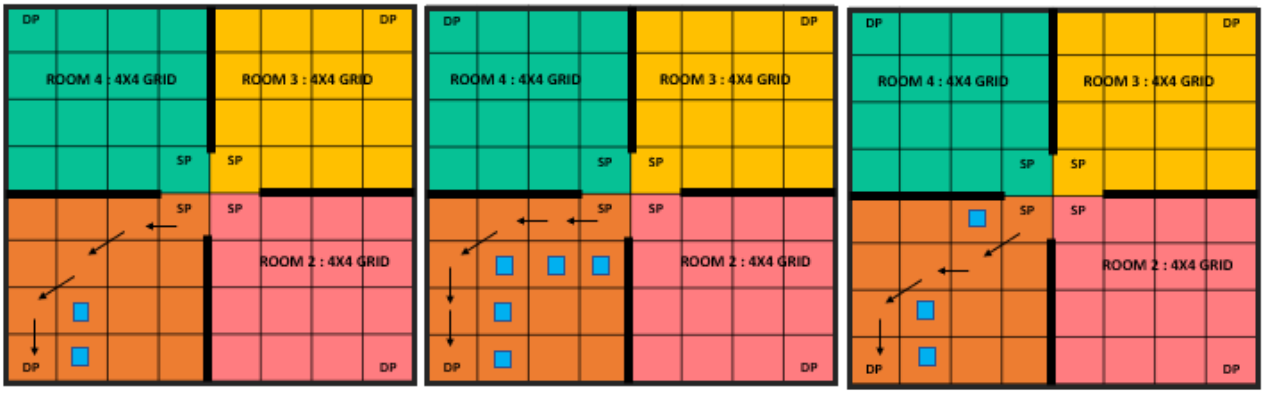


Fig 5.107: Obstacle occurrence scenarios tested in Room 1

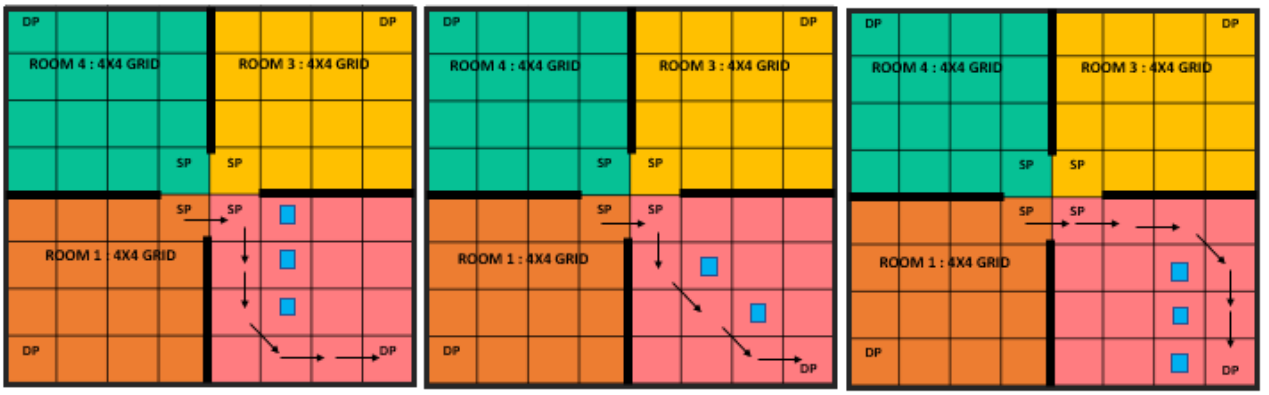


Fig 5.108: Obstacle occurrence scenarios tested in Room 2

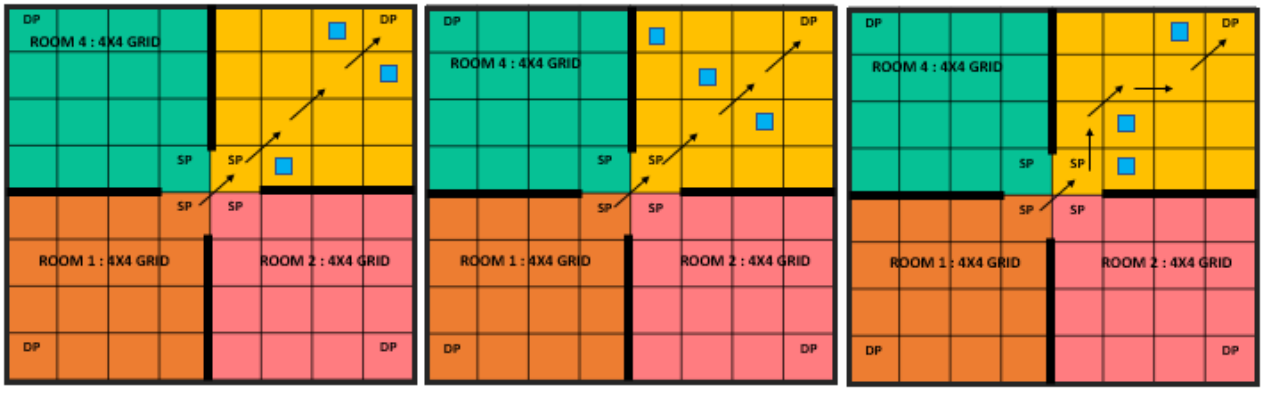


Fig 5.109: Obstacle occurrence scenarios tested in Room 3

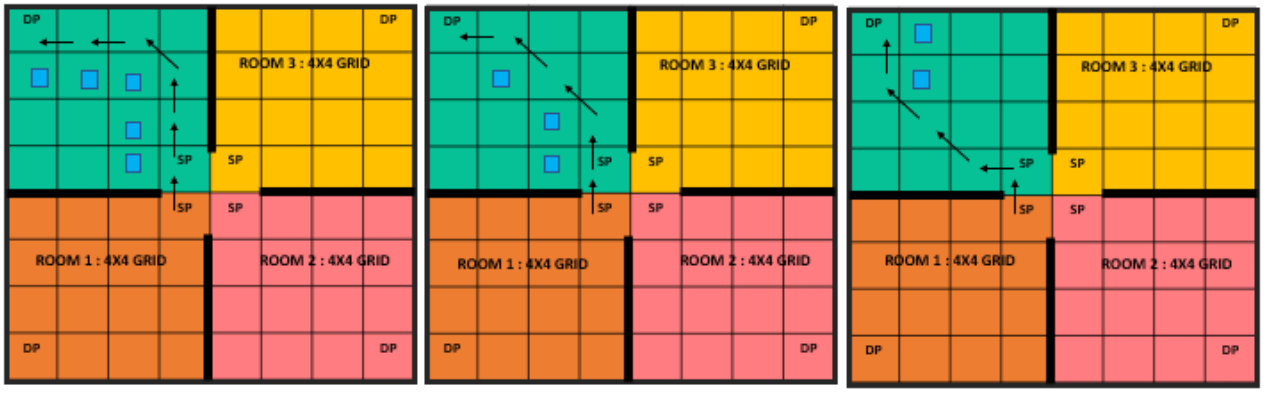


Fig 5.110: Obstacle occurrence scenarios tested in Room 4

The system layout of load carrying robot is shown in fig 5.111. It contains an app section for communication, mbed section which is coded with SARSA algorithm according to the four rooms and the iRobot control part. The hardware setup of load carrying robot is shown in fig 5.112. The initial orientation of robot in the SP of room 1 is shown in fig 5.113.

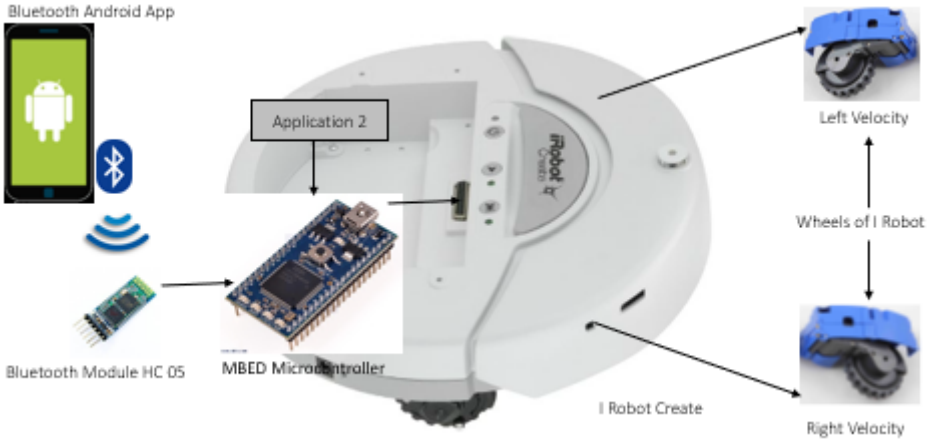


Fig 5.111: System layout of load carrying robot

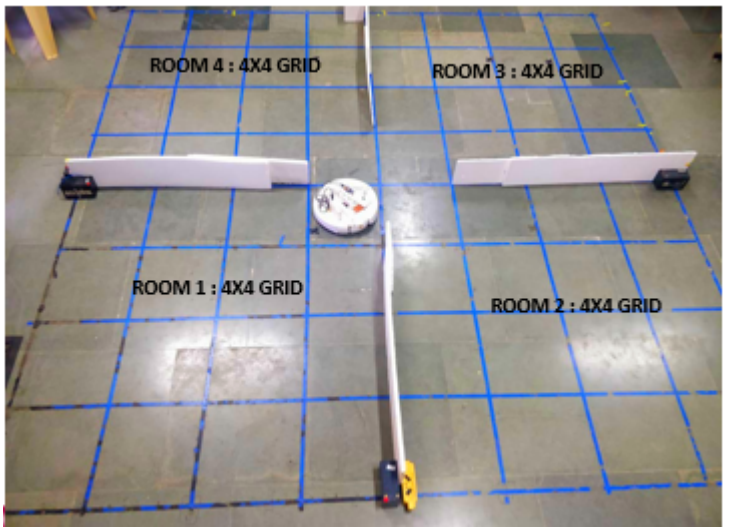


Fig 5.112: Hardware Layout of load carrying robot

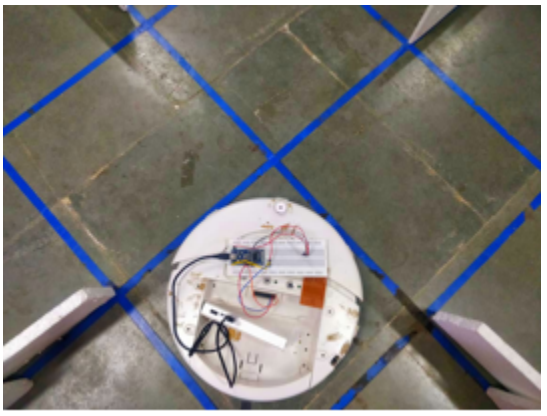


Fig 5.113: Orientation of iRobot

5.12 IMPLEMENTATION OF Q LEARNING ALGORITHM BASED PATH PLANNING FOR A 9X9 GRID

Path planning is done using Q learning algorithm for a 9x9 grid. Q learning is a RL based algorithm that trains the data according to greedy policy. The algorithm is implemented for altering source and destination points. The obstacles and the environment are fixed as shown in fig 5.114. There are 22 obstacles in the environment. The robot indicates the source point and the flag indicates the destination point and the rest are obstacles. Q learning algorithm is implemented in python IDLE. After implementing Q leaning algorithm the same cases are compared with SARSA

algorithm as shown in Table 5.13. In almost all the cases SARSA takes lesser than Q learning but there are few cases where Q leaning takes lesser though. Even though the time is more for SARSA in some cases, it always takes the safer route. The route taken by SARSA and Q learning is shown in fig 5.115 for a particular case and its evident that SARSA takes the path which is away from obstacles but Q learning moves closer to obstacles.

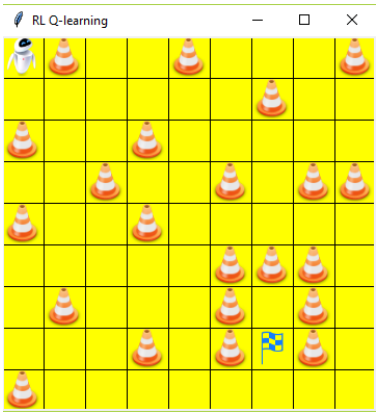


Fig 5.114: Path planning environment for Q learning

Table 5.13: Comparison between SARSA and Q learning

SL NO	SOURCE POINT	DESTINATION POINT	TIME TAKEN BY SARSA (seconds)	TIME TAKEN BY Q LEARNING (seconds)	TIME TAKEN BY Q LEARNING (seconds)
1	(0,0)	(6,7)	88.5889801979065	97.14508700370789	97.14508700370789
2	(1,8)	(8,4)	63.4701445102691	58.96404004096985	58.96404004096985
3	(8,1)	(1,8)	85.5343763828277	97.9881820678711	97.9881820678711

4	(2,0)	(8,7)	96.5203764438629	117.8740122318267	117.8740122318267
5	(0,3)	(5,0)	41.1859951019287	40.31235694885254	40.31235694885254

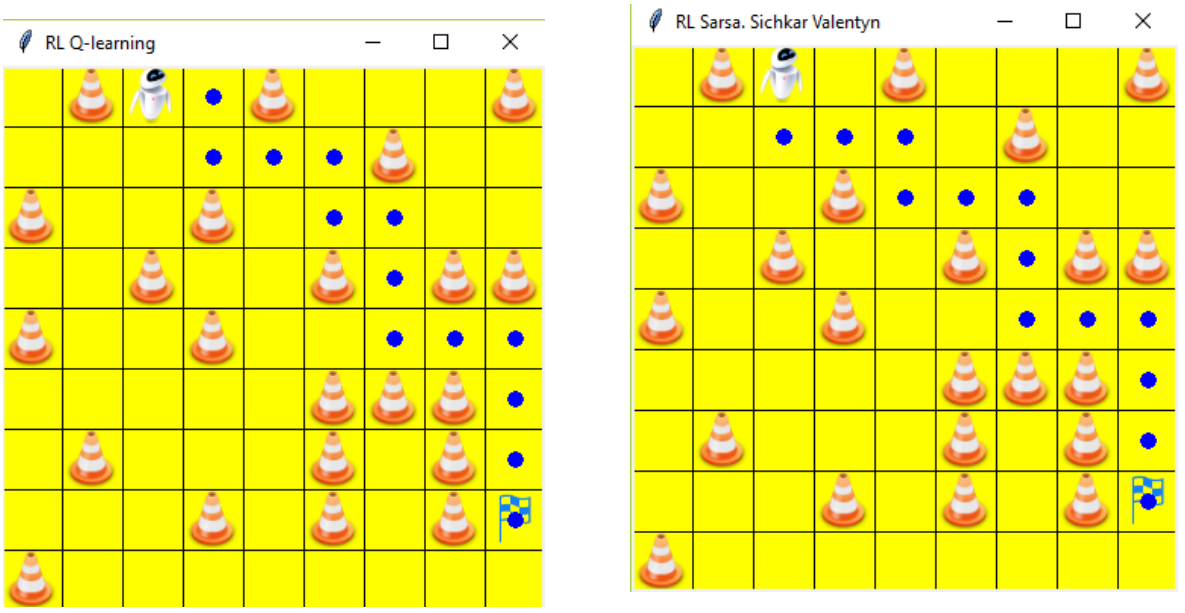


Fig 5.115: Path taken by Q leaning and SARSA algorithm

5.13 IMPLEMENTATION OF DEEP Q NETWORK ALGORITHM BASED PATH PLANNING FOR A 4X4 GRID

Deep Q Network was implemented for a 4x4 grid for different obstacle occurring environments with source and destination point fixed. The source point is fixed at (0.5,3.5) and the destination point is fixed at (2.5,1.5). Five cases have been tested. Case 1 consists of one obstacle at (2.5,2.5) as shown in fig 5.116. Case 2 consists of two obstacles at (2.5,2.5) and (1.5,1.5) as shown in fig 5.117. Case 3 consists of one obstacle at (1.5,3.5) as shown in fig 5.118. Case 4 consists of three obstacles at (0.5,0.5), (1.5,1.5) and (2.5,2.5) as shown in fig 5.119. Case 5 consists of three obstacles at (0.5,1.5), (1.5,3.5) and (2.5,2.5) as shown in fig 5.120.

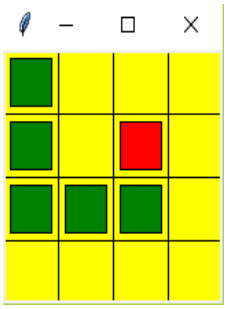


Fig 5.116: Case 1

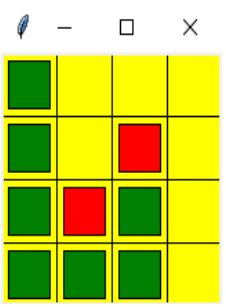


Fig 5.117: Case 2

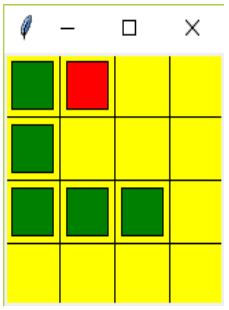


Fig 5.118: Case 3

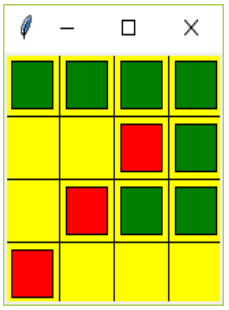


Fig 5.119: Case 4

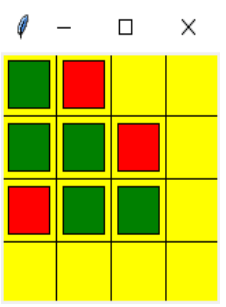


Fig 5.120: Case 5

5.14 IMPLEMENTATION OF SARSA ALGORITHM BASED PATH PLANNING FOR A 4X5 GRID

SARSA algorithm was implemented in python IDLE for a 4x5 grid. The source and destination point can be altered. Obstacle position is fixed. There are four obstacles. Different cases have been tried in simulation and hardware. The path found in python IDLE is communicated to Mbed microcontroller through Bluetooth communication. According to the values obtained in the controller the iRobot is controlled. The block diagram is shown in fig 5.121. The simulation of two cases are shown in fig 5.122 and 5.123 and their respective hardware setup is shown in fig 5.124 and 5.125 respectively.

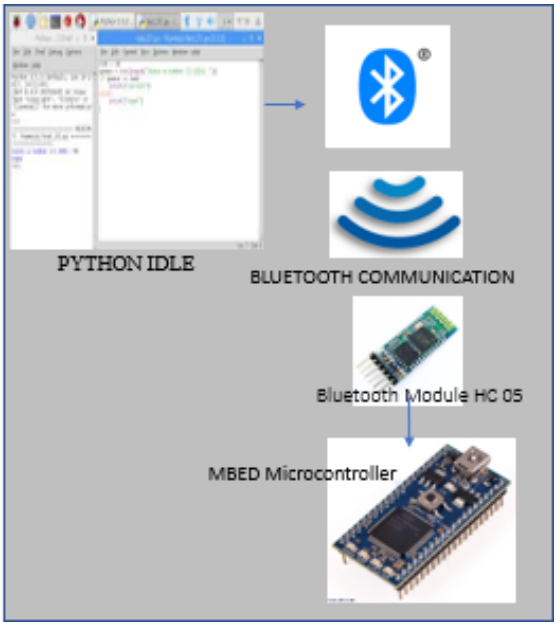


Fig 5.121: Block Diagram of SARSA algorithm for a 4x5 grid

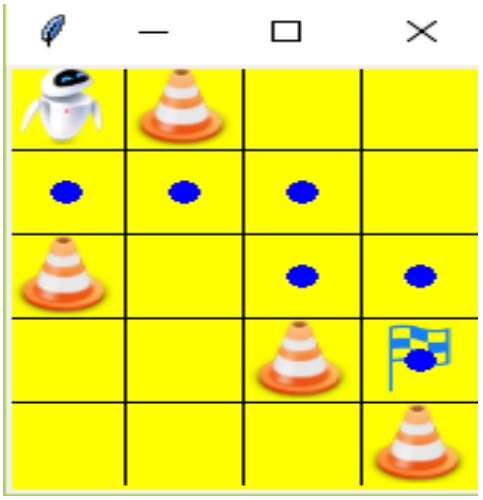


Fig 5.122: Case 1

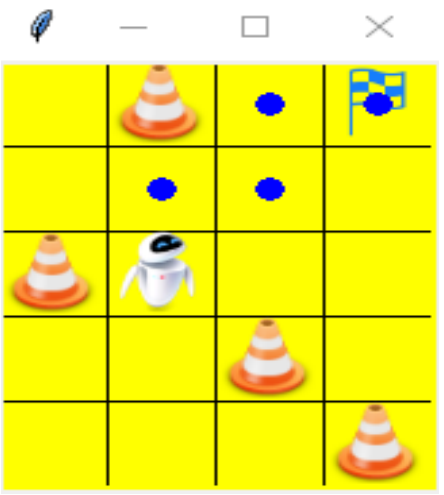


Fig 5.123: Case 2

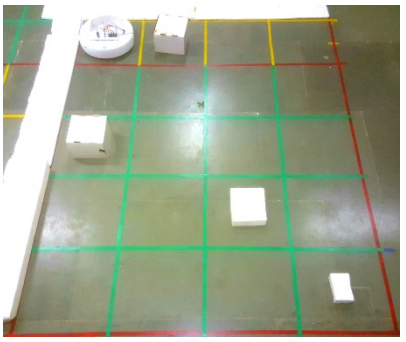


Fig 5.124: Hardware of Case 1

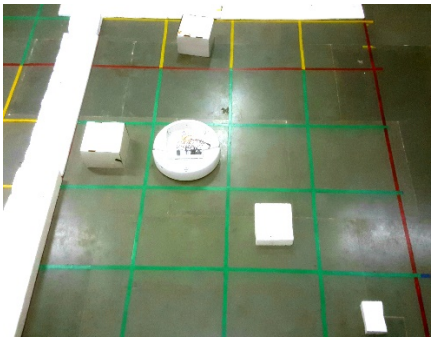


Fig 5.125: Hardware of Case 2

5.15 TIMELINE

Table 5.14: Timeline

MONTH	WORK TO BE COMPLETED	WORK COMPLETED
September- 2018	Literature Survey +Study TD learning	✔
October-2018	Literature Survey +Study of RL algorithm	✔

November, December-2018	Literature Survey +Simulation in MATLAB + Testing in MATLAB	✔
January-2019	Literature Survey + Hardware Implementation- Study of Mbed microcontroller + Development of Android based Bluetooth app+ Study of iRobot	✔
February-2019	Literature Survey + Hardware Implementation – Communication between Android app and Mbed+ Interface of Mbed and iRobot+ Implementation of TD Algorithm	✔
March-2019	Literature Survey + Hardware Implementation – Implementation of SARSA algorithm	✔
April-2019	Literature Survey + Integration and Testing	✔
May, June-2019	Literature Survey + Integration and Testing	✔

5.16 CONCLUSION

The work done till now is shown in this chapter. The algorithm and simulation results of TD, A*, Dijkstra and SARSA algorithm is shown. The hardware section is also shown. Also, the timeline of the project is shown at the end of this chapter.

CHAPTER 6

CONCLUSION

Literature survey for some papers is attempted. SARSA was studied and simulated in MATLAB for an 8x8 grid and 16x16 grid. It was also used to solve complex mazes with more than 15 obstacles for 8x8 grid and more than 150 obstacles for 16x16 grid. Different cases of obstacle occurrence are tested and verified. By this the software section of the project was completed. In the hardware section tan android-based Bluetooth app was developed. Communication between the android app and Mbed microcontroller was also done. Studied about iRobot and controlled iRobot in different directions. Implemented SARSA and TD algorithm in hardware. Incorporated IoT concepts in project. Implemented SARSA algorithm for 8x8 grid. Implemented robot to robot communication using SARSA algorithm.

PUBLICATIONS

[1] Laya Harwin and P Supriya, “Comparison of SARSA and Temporal Difference Learning Algorithm for Robotic Path Planning for Static Obstacles”, Presented the paper in 3rd International Conference on Inventive Systems and Control (ICISC Jan 10, 2019).

REFERENCES

1. Ravishankar, N.R, Vijayakumar and M.V," Reinforcement Learning Algorithms: Survey and Classification", Indian Journal of Science and Technology**,** [S.l.],2017, doi:10.17485/ /2017 /v10i1/109385.

2. Chen Xia, "Intelligent Mobile Robot Learning in Autonomous Navigation. Automatic Control Engineering" Ecole Centrale de Lille, (2015):2015ECL10026.

3. Shreyas J and Sandeep J. " Modern Machine Learning Approaches For Robotic Path Planning", IJCSIT(2016):256-259

4. P. Joshy and P. Supriya, "Implementation of robotic path planning using Ant Colony Optimization Algorithm," *2016 International Conference on Inventive Computation Technologies (ICICT)*, Coimbatore, pp. 1-6, 2016.

5. D. Davis and P. Supriya, "Implementation of Fuzzy-Based Robotic Path Planning", 3rd IEEE International Conference on Engineering and technology (ICETECH), 2016.

6. Devika S. Nair and P. Supriya," Comparison of Temporal Difference Learning Algorithm and Dijkstra's Algorithm for Robotic Path Planning", International Conference on Intelligent Computing and Control Systems, Madurai. 2018. (to be published)

7. Zhang, Qian et al. "Reinforcement Learning in Robot Path Optimization.", Journal of Software(*JSW*7)(2012):657- 662.

8. D. P. Romero-Martí, J. I. Núñez-Varela, C. Soubervielle -Montalvo and A. Orozco-de-la- Paz ,"Navigation and path planning using reinforcement learning for a Roomba robot," *2016 XVIII Congreso Mexicano de Robotica*, Sinaloa, 2016, pp. 15.

9. Zhang, Qian et al. "Reinforcement Learning in Robot Path Optimization." , Journal of Software(*JSW* 7) (2012):657-662.

10. Nihal Altuntas 1, Erkan Imal2, Nahit Emanet1, Ceyda Nur Ozturk, "Reinforcement learning based mobile robot navigation", Turkish Journal of Electrical Engineering and Computer Sciences 24(3):1747-1767DOI:10.3906/elk –1311- 129

11. Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, Wolfram Burgard, " Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments" {2016arXIv1 ,612 05533Z}2016, arXiv e-prints,arXiv:1612.0 5533.

12. H. Iima and Y. Kuroe, "Swarm reinforcement learning algorithms based on Sarsa method", *2008 SICE Annual Conference*, Tokyo, 2008, pp. 2045-2049

13. C. Li, M. Wang, S. Yang and Z. Zhang, "Urban Traffic Signal Learning Control Using SARSA Algorithm Based on Adaptive RBF Network", 2009 International Conference

on Measuring Technology and Mechatronics Automation, Zhangjiajie, Hunan, 2009,pp.658-661.doi:10.1109/ICMTMA . 2009.445

14. N. Lilith and K. Dogancay, "Distributed reduced-state SARSA algorithm for dynamic channel allocation in cellular networks featuring traffic mobility," *IEEE International Conference on Communications, 2005. ICC*, Seoul, 2005, pp. 860-865 Vol. 2. doi: 10.1109/ICC.2005.1494473

15. M. R. Tousi, S. H. Hosseinian, A. H. Jadidinejad and M. B. Menhaj, "Application of SARSA learning algorithm for reactive power control in power system," *2008 IEEE 2nd International Power and Energy Conference*, Johor Bahru, 2008, pp. 1198-1202. doi: 10.1109/PECON.2008.4762658

16. Ganesha D, Venkatamuni, Vijayakumar Maragal, “Implementation of modified SARSA learning technique in EMCAP”, International Journal of Engineering & Technology, [S.l.], v. 7, n. 1.5, p. 274-278, dec. 2017. ISSN 2227-524X

17. Tran Xuan Sang, TranQuoc Kiet, Nguyen ThiUyen, “Path Finding Algorithm for Autonomous Robots Based on Reinforcement Learning”, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 6, Issue 4, April 2017, ISSN: 2278 – 1323

18. D. Xu, Y. Fang, Z. Zhang and Y. Meng, "Path Planning Method Combining Depth Learning and Sarsa Algorithm," *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, Hangzhou, 2017, pp. 77-82.doi:10.1109/IS CID.2017.145

19. Wei Wu, Zhang Qi Sen, J. B. Mbede and Huang Xinhan, "Research on path planning for mobile robot among dynamic obstacles," *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*, Vancouver, BC, Canada, 2001, pp.763-767 vol.2.doi:10.1109/NAFI PS.2001.944699.

20. W. Yu, C. Yang, K. Su and Y. Tu, "Dynamic path planning under randomly distributed obstacle environment," *2014 CACS International Automatic Control Conference (CACS 2014)*, Kaohsiung, 2014, pp. 138-143. doi: 10.1109/CACS.2014.7097177.

21. Mourtzis, Dimitris & Vlachou, Katerina & Zogopoulos, Vasilios,” An IoT-based Platform for Automated Customized Shopping in Distributed Environments”, Procedia CIRP. 72. 892-897. 10.1016/j.procir.2018.03.199.

22. Libin M George1, Annu Mariam Abraham2, Ananthapadmanabhan J3 , Vineetha Anna Saji4 , Anil A R5,”Implementation of IoT In Smart Robotics: A Survey”, International Journal Of Engineering And Computer Science ISSN:2319-7242 Volume 6 Issue 3 March 2017, Page No. 20762-20764 Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i3.63

23. Mackellar, Bonnie," App inventor for android in a healthcare IT course", SIGITE'12 - Proceedings of the ACM Special Interest Group for Information Technology Education Conference. 245-250. 10.1145/2380552.2380621.

24. F. Turbak, D. Wolber and P. Medlock-Walton, "The design of naming features in App Inventor 2," *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Melbourne, VIC, 2014, pp. 129-132.doi: 10.1109/VLHCC.2014.6883034

25. J. Tyler, "*App Inventor for Android*. Wiley & Sons", 2011.

26. L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcementlearning: A survey," Journal of Artificial Intelligence Research, vol. 4,pp. 237–285, 1996.

27. Kwon, Woo Young & Suh, Il Hong & Lee, Sanghoon & Cho, Young-Jo,"Fast reinforcement learning using stochastic shortest paths for a mobile robot", 82 - 87. 1.

28. Otte, Michael W.. "A Survey of Machine Learning Approaches to Robotic Path-Planning." (2009).