TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Minor Project Report

On

Dynamic Maze Solving using D*-Lite and Dead-End Exclusion Algorithm

Submitted By:

Anish Dahal (THA074BEX004)

Dipak Gurung (THA074BEX009)

Prajwol Pakka (THA074BEX022)

Sujal Subedi (THA074BEX043)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

February, 2021

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Minor Project Report
On
Dynamic Maze Solving using D*-Lite and Dead-End Exclusion Algorithm

Submitted By:

Anish Dahal (THA074BEX004)

Dipak Gurung (THA074BEX009)

Prajwol Pakka (THA074BEX022)

Sujal Subedi (THA074BEX043)

Submitted To:

Department of Electronics and Computer Engineering
Thapathali Campus
Kathmandu, Nepal

In partial fulfillment for the award of the Bachelor's Degree in Electronics and Communication Engineering.

Under the Supervision of

Er. Dinesh Baniya Kshatri

February, 2021

DECLARATION

We hereby declare that the report of the project entitled **"Dynamic Maze Solving using D*-Lite and Dead-End Exclusion Algorithm"** which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, in the partial fulfillment of the requirements for the award of the Degree of Bachelor of Engineering in **Electronics and Communication Engineering**, is a bonafide report of the work carried out by us. The materials contained in this report have not been submitted to any University or Institution for the award of any degree and we are the only author of this complete work and no sources other than the listed here have been used in this work.

Anish Dahal (THA074BEX004) _____

Dipak Gurung (THA074BEX009) _____

Prajwol Pakka (THA074BEX022) _____

Sujal Subedi (THA074BEX043) _____

Date: February, 2021

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus**, a minor project work entitled "**Dynamic Maze Solving using D*-Lite and Dead-End Exclusion Algorithm**" submitted by **Anish Dahal, Dipak Gurung, Prajwol Pakka** and **Sujal Subedi** in partial fulfillment for the award of Bachelor's Degree in Electronics and Communication Engineering. The Project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled and ready to undertake any related work to their field of study and hence we recommend the award of partial fulfillment of Bachelor's degree of Electronics and Communication Engineering.

_____

Project Supervisor

Er. Dinesh Baniya Kshatri

Department of Electronics and Computer Engineering, Thapathali Campus

_____

External Examiner

_____

Project Coordinator

Er. Dinesh Baniya Kshatri

Department of Electronics and Computer Engineering, Thapathali Campus

_____

Head of Department

Er. Kiran Chandra Dahal

Department of Electronics and Computer Engineering, Thapathali Campus

February, 2021

COPYRIGHT

ACKNOWLEDGEMENT

We would like to express our sincere gratitude towards the Institute of Engineering, Tribhuvan University for the inclusion of minor project in the course of Bachelor's in Electronics and Communication Engineering. We are also thankful to our Supervisor Er. Dinesh Baniya Kshatri and the Department of Electronics and Computer, Thapathali Campus for providing us with the resources and support which is needed for this project.

Anish Dahal (THA074BEX004)

Dipak Gurung (THA074BEX009)

Prajwol Pakka (THA074BEX022)

Sujal Subedi (THA074BEX043)

ABSTRACT

This project involves designing a robot that navigates around a maze, to optimally reach the goal from the start position. The robot performs this task in two-phases. In the first phase, also known as the exploration phase, it navigates around the maze and performs the mapping of the maze. In the second phase, also known as the goal-seeking phase, it moves to the goal from start in the optimal path calculated using the map. The maze used is dynamic i.e., obstacles are added to the robot's path during the second phase. Each time the robot encounters an obstacle, it updates its path to the next optimal one until there is no possible path to the goal, in which case the robot stops in the track. The robot uses the Dead-End Exclusion algorithm for mapping and the D*-Lite algorithm to find the optimal path from start to goal.

Keywords: Dead-End Exclusion Algorithm, D-Lite Algorithm, Dynamic Maze*

Contents

List of Figures

List of Tables

List of Abbreviations

BFS	Breadth First Search
DMP	Digital Motion Processor
EDA	Electronic Design Automation
FIFO	First In First Out
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
PID	Proportional Integral Derivative
RRT	Rapidly Exploring Random Tree
VRLM	Virtual Reality Modeling Language

1. INTRODUCTION

A maze is a network of paths from a start point to the finish point. It has been around humans for ages. Maze solving problems in robotics started in the 1970s and it is being more popular in the present. The maze solving problem is based on one of the key and important area in robotics “The Decision-Making Algorithms” where a robot has to decide what to do next and in maze solving context which path to take next. In maze solving, a robot is placed in an unknown environment and it has to find a way to the goal.

1.1 Motivation

People are always captivated by robots moving and taking decisions on their own. The team members were always interested in robotics and also managed to make few robots in the past. However, the maze-solving robots caught attention last year, while trying to participate in micro mouse competition in India. The team tried building a maze solving robot for the first time, but failed. Limited research and underestimating the complexity were the reason. But after a year of learning and researching about these types of bots, the members are experienced and have better understanding of the problem.

People making these types of bots are generally focused on static maze solving. The team wanted to explore, what if the maze is dynamic? What if obstacles are added to the path? To explain such a scenario, the team decided to choose this topic.

1.2 Problem Definition

This project aims to solve a dynamic maze by exploring it and then finding the shortest path to the goal. This is accomplished by using Dead-End Exclusion Algorithm for exploration and mapping of the maze and D*-Lite Algorithm for goal seeking.

1.3 Project Objectives

The specific objectives of the project are listed below:

- To solve a dynamic maze with real-time obstacles
- To use Dead-End Exclusion algorithm for maze mapping
- To use D*-Lite algorithm for goal seeking

1.4 Project Applications

There are many applications of maze solving robots. Some of the prominent real-life applications of the project are listed below.

- Disaster Management – The robot can be used to provide essential goods like food, water, and other medical supplies to people during a crisis. For example: earthquakes, and pandemics like Covid-19.

- Warehouse Management – Robots that can efficiently travel between two places can be used to shift goods from one place to another. This helps to reduce the required manpower and make the process efficient.

- Navigation – There are many automobiles which provides autonomous and semi-autonomous navigation. They use map of the location and the data provided by the sensors to avoid traffics and reach the destination which is similar to a maze solving robot which also uses a map and sensor data to move to the goal avoiding obstacles. Example of such automobile include Tesla self-driving cars.

- Speleology – It is the study and exploration of caves. There are a lot of caves that are like mazes where humans could get lost in and haven't been fully explored yet. A robot that can solve a maze, could map it and take samples, if it is small or dangerous for humans.

- Delivery Robots – These robots prioritize its packages and thus need to avoid obstacles in the path. They should navigate to the destination area in shortest

time possible. An efficient algorithm which performs these tasks is significant. Companies like Amazon, Robomart are using delivery robots for home delivery.

- Cleaning Robots – These robots need to map the room completely and then calculate the best path to the uncleaned areas. A better algorithm always helps the bot to clean the room efficiently.

1.5 Scope of Project

The robot can map a maze without completely moving to every location of the maze. Then that map is used to calculate the shortest path to the goal from the starting position. If the robot encounters an obstacle in the calculated path, it avoids the obstacle and chooses the next shortest path until there is no possible path to the goal, in which case the bot will stop in its track.

There are a few limitations to the project as well. The robot is unaware if the obstacles in the maze are removed and another shortest path is formed. It would continue to follow the path that it believes to be the shortest. The robot will not work in a scenario where the goal location can change.

1.6 Report Organization

This report is divided into 8 chapters. Each chapter discuss different issues about the project. Chapter 1 explains the basic introduction about maze solving, project statement, objectives and the applications of the project. Chapter 2 covers the background information about different algorithms regarding maze solving. This section describes the previously developed algorithms. Chapter 3 gives insight about different software components required for the project. Methodology and working principles of the robot are described in Chapter 4, along with the block diagram of the system and the algorithms used for maze mapping and goal seeking. In chapter 5, the implementation of the project along with the environment setup and robot design regarding simulation are explained. In chapter 6, simulated outputs of the Dead-End Exclusion Algorithm and the D*Lite Algorithm is shown. The future enhancements which can be made in this project is described it chapter 7. Conclusion which the project members have drawn from this project is covered in chapter 8.

2. LITERATURE REVIEW

The concept of maze-solving was introduced by IEEE spectrum magazine in May, 1977 and the first announced competition was held in New York in 1979 where a robot had to find a way out of a 10×10 square feet maze [1]. For solving a maze, minimum of two algorithms are used, first for exploration and mapping of the maze then another one for finding the optimal path to the goal.

Exploration algorithms are those algorithms that help the robot to explore the maze, tracking the walls and path with the help of different sensors. These algorithms can find a path to the goal but the path found is not optimum. These algorithms will map the maze, tracking where the walls are and which path leads to goal. Tremaux's algorithm is one of the algorithms, which is used to explore and map a maze.

In Tremaux's algorithm, each cell has a parameter that counts the number of times the cell is visited by the robot. For each path in the maze, the path can be unmarked, marked once, or marked twice. This is also known as marking [2]. The robot remembers these markings and travels the maze, finding the goal, and making additional searches to find the optimal path to goal. Although Tremaux's Algorithm has its advantages, it has its limitations as well. It uses a random path going out of the junction which can cause some disorder in the exploration as that path can take the robot to some explored path. Also, it is time-consuming as it explores every cell of the maze, so it explores even the dead-ends which are present in the maze. The limitations of this algorithm are taken into account and Dead-End Exclusion algorithm is introduced.

Dead-End Exclusion algorithm marks the cell which has been explored. If there exists a cell that has been marked in three directions, this algorithm adds the fourth mark making the cell unnecessary to be explored [3]. This way the dead-end is found out in the maze and the path to the dead-end is virtually closed. After the implementation of the maze exploration algorithm, goal-seeking algorithm is implemented.

Goal-seeking algorithms are those algorithms that find the shortest path from the start to the goal. These algorithms use the map from the maze exploration process and the sensor data to find the optimal path from start to goal. These algorithms are broadly categorized into two types, search-based algorithms, and sampling-based algorithms.

A* is a search-based algorithm that combines the feature of uniform-cost search and heuristic search. It is BFS in which the cost associated with each node is calculated using admissible heuristic [4]. The nodes are used to record the progress of the search. In addition to holding the map location, each node has three other attributes that are fitness, goal, and heuristic commonly known as f, g, and h respectively [5]. A* search algorithm is implemented to find the shortest path between source and destination. Although A* algorithm has a lot of advantages, it has limitations as well. The execution speed of A* algorithm is greatly dependent upon the heuristics used 'h'. A* algorithm cannot find another path to the goal if the calculated path to the goal has been blocked. To solve these types of problems D* algorithm was introduced.

D* resembles A*, except that it is dynamic, in the sense that arc cost parameters can change during the problem-solving process. The path of the robot changes in the middle of the goal-seeking process when an obstacle is detected [6]. Each node has two other attributes that are fitness, heuristic commonly known as f and h respectively. For D* algorithm, fitness is always equal to heuristic, which represents the estimated cost to the goal. The corrections of the arc cost that came from the sensors could be made at any time, also the estimated and measured arc costs are comprised to the environment map. This strategy could be used in any planning representation, including grid cell structures and visibility graphs [7]. Although D* algorithm solves a dynamic maze, it has its limitations. The heuristics used in D* algorithm isn't suitable to find the next shortest path if an obstacle is detected while traversing the maze to reach the goal.

RRT is a sampling-based goal-seeking algorithm. The RRT algorithm is primarily aimed at single-query applications. In its basic version, the algorithm incrementally builds a tree of feasible trajectories, rooted in the initial condition [8]. The new node generated is connected to the node which is closest to the newly generated node in the space. RRT algorithm is used in self-automating vehicles in urban areas [9] where a vehicle has to evade different obstacles which may be cars, pedestrians, street lights. The performance of the RRT algorithm can be poor in environments containing narrow passages. The narrow passages stunt the tree growing process which slows down searching for the resulting path [10]. The final path found out by the RRT algorithm may not be the optimal path to the goal. Hence RRT* algorithm was introduced.

RRT* algorithm is similar to the RRT algorithm but instead of joining to the nearest node, the newly generated node is jointed to the node which is closest to the starting node. In RRT*, for each new node added to the tree, its neighborhood of nodes is generated with a given radius distance. the new sampled node is not added to the nearest node. Instead, the newly sampled node is added to its neighbor node that provides the shorter path length to the root node [11]. When the number of nodes tends to infinity, an optimal path is found. Although it gives the optimal path to the goal, the time taken to compute that path is large. For finding the optimal path to the goal, RRT* algorithm is slow.

The D*-Lite algorithm solves a dynamic maze by finding out the shortest path from goal to start. This algorithm finds the next shortest path to the goal when an obstacle is detected and follows that path. In D*-Lite algorithm the number of nodes is constant as the cell of the maze is constant, so the computation capacity required is lower.

3. REQUIREMENT ANALYSIS

The tools used in this project are briefly explained. Only the software tools are used because of the unavailability of the hardware components like distance sensor and IMU sensor, the maze creation process was also affected due to the absence of common sharing location among the team members due to the COVID-19 pandemic.

3.1 Software Tools

The project is completed in Webots robot simulator, where the creation of robot, the environment and programming are performed.

3.1.1 Webots Robot Simulator

The R2020B revision1 version of Webots is a beginner-friendly robotics simulator that provides a complete development environment to model, program, and simulate robots. In Webots, a library of sensors is provided which can be integrated into the robot and can be customized. Among the different functionalities of Webots, the following specific functions have been used:

3.1.1.1 Sensor Library

Among the variety of available sensors, specifically two types of sensors have been used. Three distance sensors are used for detecting the maze walls and obstacles. Similarly, the IMU sensor is used to maintain the orientation of the robot.

3.1.1.2 Actuator Library

The actuators used in this project include a differential wheel motor unit and independent wheel motors. The motion of these motors is carried out using signal provided by the controller.

3.1.1.3 Controller

Webots allows the development of a controller for a robot using C, C++, Java, MATLAB and Python. The programming language chosen for this project is C++. The controller receives signals from the sensors and generates control signals for the actuators.

4. SYSTEM ARCHITECTURE AND METHODOLOGY

The main entities in the project are the maze and the robot. The robot contains a controller which performs Dead-End exclusion algorithm and D*-Lite algorithm to solve the maze.

4.1 System Block Diagram

The block diagram of the system features the maze in which the robot navigates, the robot itself and the controller which governs the robot. The system block diagram is shown in Figure 4-1.

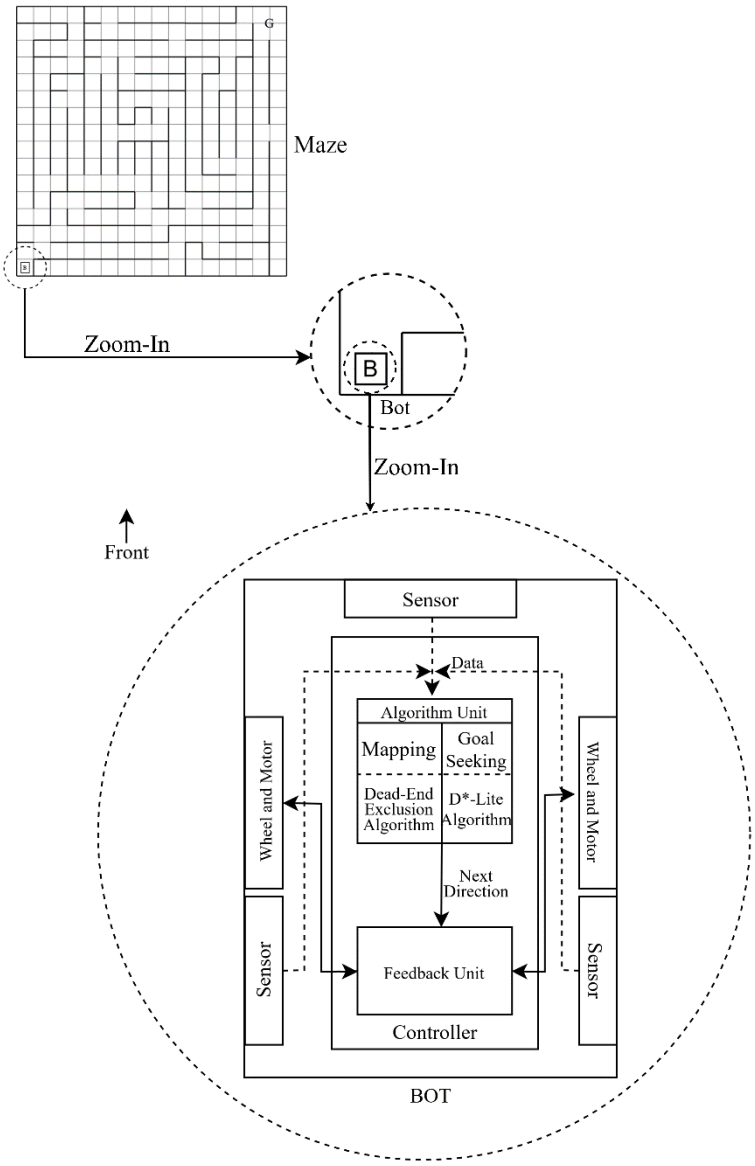

Figure 4-1: System Block Diagram

4.1.1 Maze Description

The maze is the environment where the robot moves. The robot has to map this environment and go to the goal from the start position. The maze is virtually divided into many cells that are not discoverable using sensors.

4.1.1.1 Maze Structure Modelling

The maze in our project only contains three things, walls, the robot and obstacles. The maze structure was modelled by keeping the relation between them in mind. Following are the parameters which decides the structure of the maze.

Table 4-1: Maze Structure Modelling

S.N.	Maze Entities	Specifications
1.	Maze Size	16 × 16 square array of 180 mm × 180 mm unit squares
2.	Wall Height	150 mm
3.	Wall Width	12 mm
4.	Passageway	168 mm
5.	Outside Wall	The outside wall of 6 mm wide will enclose entire maze
6.	Start of Maze	Located at one of the four corners
7.	Starting Square	Will have walls on three sides
8.	Goal	Will be a large opening composed of 4-unit squares.

The blueprint of the maze is shown in figure 4-2. The bottom left corner and the top right corner represents the start and goal respectively.

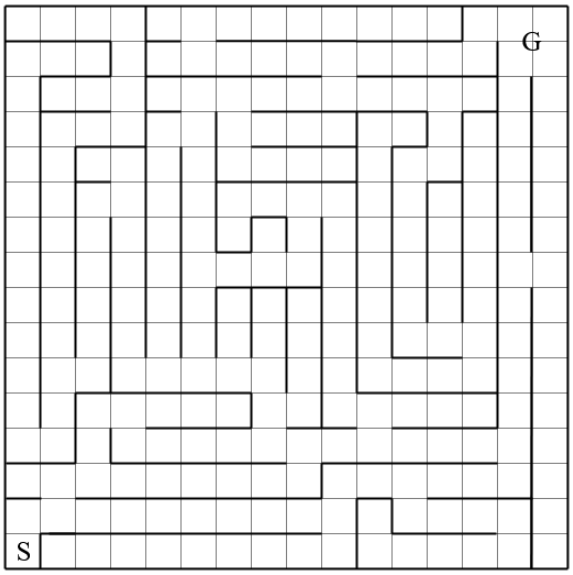

Figure 4-2: Maze Blueprint

Figure 4-3 shows the dimension of single unit cell. The size of cell is 180 mm × 180 mm, but the robot cannot use all of that space to travel. Some part of the cell is covered by the walls. Thus, the net area available for the robot to move is given by passageway.

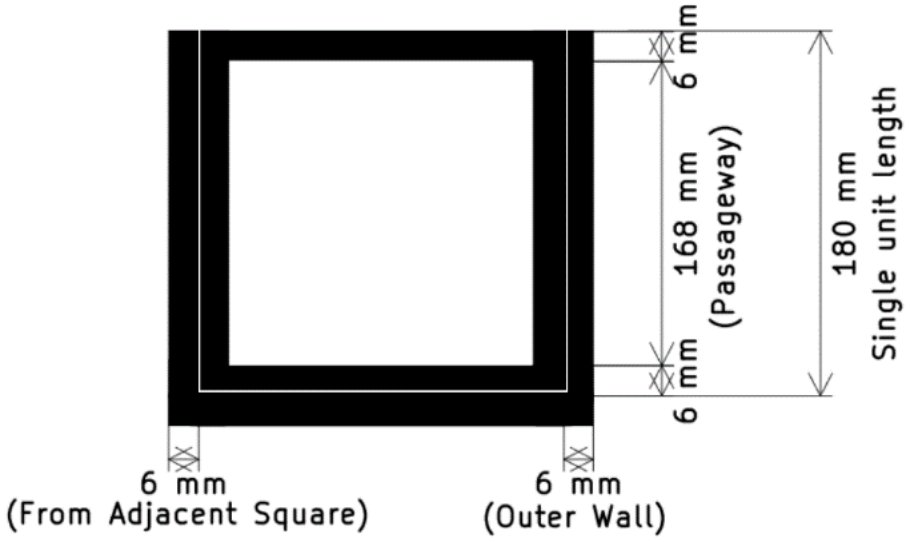

Figure 4-3: Dimension of Single Cell

The area of the maze is insufficient to infer the scenario of how the interaction between the robot and the maze will take place. If the height of maze is smaller than the height of robot, then the robot will not interact with the maze at all and will just go straight to the walls. Thus, figure 4-4 shows the corner of the maze which shows the height of the maze. The height of the maze is uniform throughout the maze.

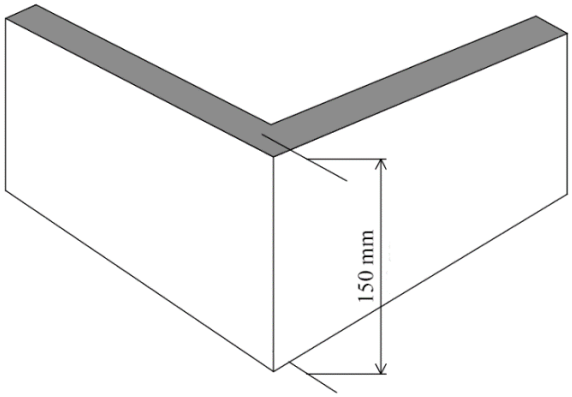

Figure 4-4: Height of Maze

4.1.2 Robot Description

The robot navigates in a virtual surrounding. To maneuver itself in the environment, the robot relies on data obtained from its sensors which are used by its controller for navigating towards the goal.

4.1.2.1 Bot Structure Modelling

The structure of the bot is modelled keeping the sensors and wheels in mind. The modelling of the bot is as important as the modelling of the maze. The Table 4-2 shows all the parameters which is required for bot structure modelling.

Table 4-2: Robot Structure Modelling

S.N.	Robot Description	Dimension
1.	Robot Size	100 mm × 90 mm × 40 mm
2.	Wheel Diameter	40 mm
3.	Wheel Width	20 mm
4.	Caster Wheel Diameter	15 mm
5.	Caster Wheel Height	15 mm

Figure 4-5 shows the top view of the bot. It displays the size of the robot, number of wheels used and their relative position.

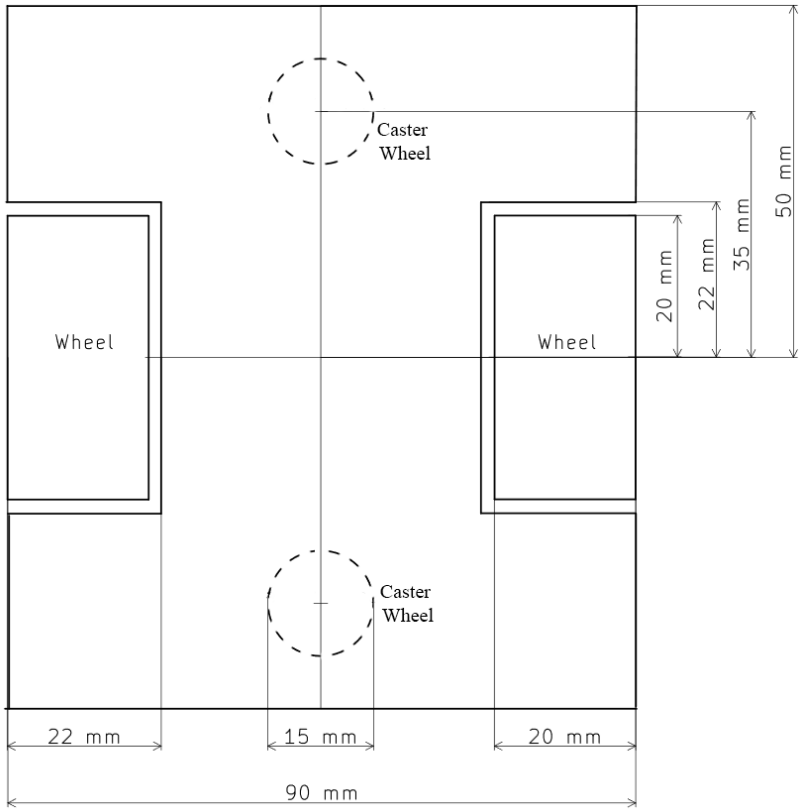

Figure 4-5: Robot Top View

The top view of the robot is insufficient to conclude the 3D model of the robot. Thus, figure 4-6 is used to show the left side view of the bot.

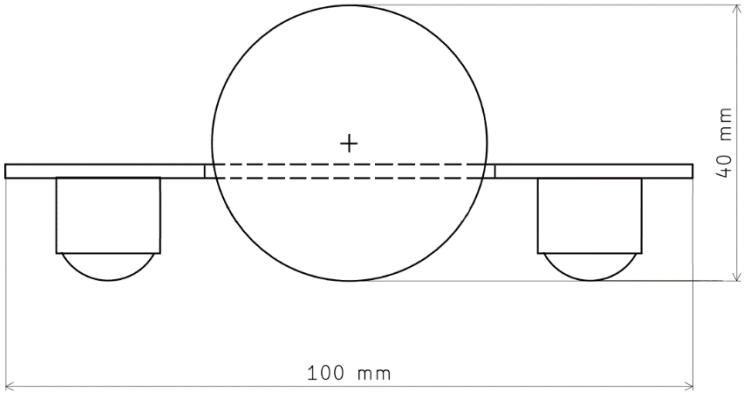

Figure 4-6: Robot Left Side View

4.1.2.2 Bot Motion Modelling

For the motion modelling of the robot, it is considered that the caster wheels don't contribute to the motion of the robot. The known value from the robot are: the change in the rotation of the left wheel ($\triangle_L$) and right wheel ($\triangle_R$) expressed in radian, from which the linear distance is calculated by multiplying it with the radius of wheel (R=20mm).

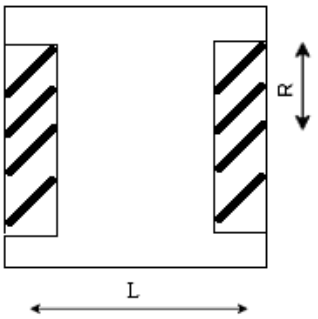

Figure 4-7: Bot Parameters

In any time instant, the total distance travelled by robot is calculated by taking the mean of linear distance of left and right wheel. The orientation of robot is given by the IMU sensor.

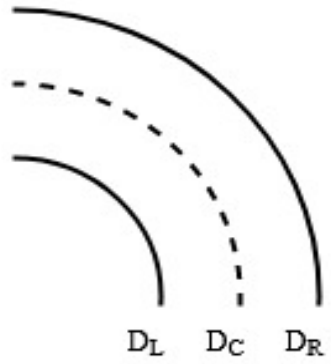

Figure 4-8: Single Point Approximation of Bot

$$D_L = \triangle_L \times R = 20\ \triangle_L \quad (4.1)$$

$$D_R = \triangle_R \times R = 20\ \triangle_R \quad (4.2)$$

$$D_c = \frac{D_R + D_L}{2} \quad (4.3)$$

where D_c = distance travelled by center of robot,

D_L = distance travelled by left motor,

D_R = distance traveled by right motor,

In any instant, a robot is either moving forward, or rotating. While moving forward, if the given goal location is (x_d, y_d) with an initial location (x, y), the required parameters to move to the goal location are given by the equations 4.4 and 4.5.

$$d = \sqrt{(x_d - x)^2 + (y_d - y)^2} \quad (4.4)$$

$$v_L = v_R = v_{max} \text{ if } d > D_{tolerance} \text{ or } 0 \quad (4.5)$$

where d = distance between the points,

v_L = velocity of the left wheel,

v_R = velocity of the right wheel

While rotating, the required orientation is given by the control unit and the current orientation of robot is given by the IMU sensor. The robot rotates by turning both wheels in opposite direction. The robots keep rotating unless the difference of the robot's actual orientation and target orientation is smaller than the tolerance value.

$$w = (\theta_d - \theta) \quad (4.6)$$

$$v_L = v_{max} \text{ if } w > w_{tolerance} \text{ or } 0 \quad (4.7)$$

$$v_R = -v_{max} \text{ if } w > w_{tolerance} \text{ or } 0 \quad (4.8)$$

where d = distance between the points,

θ_d = required orientation of robot,

θ = current orientation of robot,

w = difference between required angle and current angle

Hence from these equations, the bot can move in a straight line and rotate to the required orientation. If there are some errors in movement, it is compensated by the PID controller.

4.1.2.3 Robot Entities

The robot consists of many entities for the completion of the given task of maze exploration and goal seeking. The entities include controller, sensors and wheels.

a. Controller

It is the brain of the robot which performs overall process of mapping and goal-seeking. The controller takes the data from the robot sensors and then makes a decision based upon the data. The controller is divided into two parts: one which is dedicated to the mapping and goal-seeking process and the other which makes sure that the robot is moving to the place the controller is telling it to, this part is also known as the Feedback controller. It takes feedback from the encoders and the decision from the algorithms to do its job.

b. Encoder Motor

They are a special type of motors which rotates wheels like any other motors but also provides feedback about the wheel rotated. The tick count is used for the calculation of the current position of the robot and the feedback input.

c. Distance Sensor

Distance Sensor is the sensory unit of the bot which helps the robot to perceive the environment. It gives the distance to the object pointed by it. It is used to know the position of the wall and obstacles in the project. It is also used by the PID controller to move the robot in a straight path.

d. IMU Sensor

It is a device that can measure the orientation of an object. This is attached to the robot which then provides its alignment. This sensor value changes after the robot rotate in any direction.

4.2 Working Principle

The distance from the distance sensors gives the availability of the movement in a certain direction of the maze cell. The working of the robot can be divided into two main parts, namely Maze Exploration and Goal Seeking.

4.2.1 Maze Exploration

It is an important part of the maze solving process. The rest of the process is irrelevant if this process is not done properly. A proper map is the first requirement for the further process. Dead-End exclusion algorithm is implemented for this process.

4.2.1.1 Dead-End Exclusion Algorithm

This algorithm detects the walls and finds out the path which leads to a dead-end with the help of distance sensors. It does not allow the robot to go to those ends when exploring.

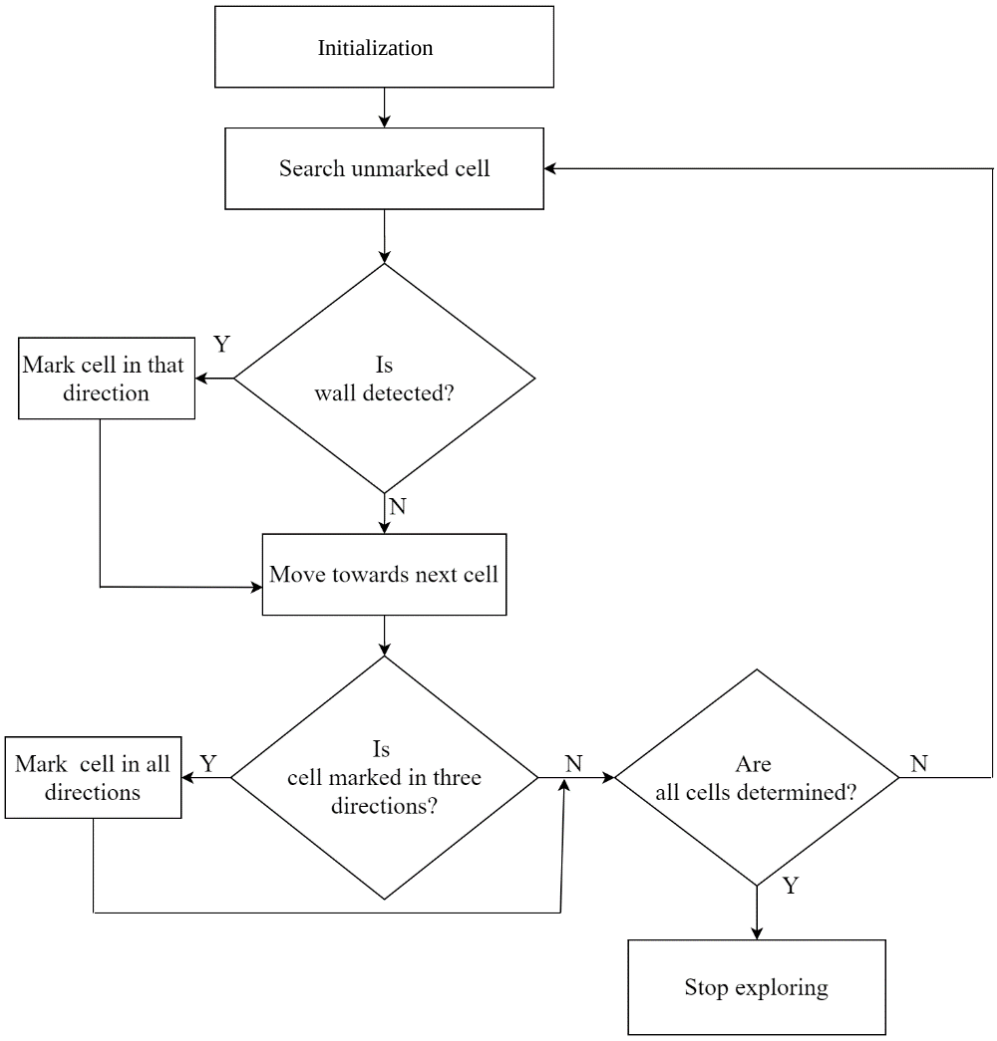

Figure 4-9: Flowchart of Dead-End Exclusion Algorithm

4.2.2 Goal Seeking

The previous maze exploration process provides us with a map. This map is then used to extract features like goal position and the wall position, then a shortest path is calculated to the goal from the starting process. Goal seeking in the project is done using the D*-Lite algorithm.

4.2.2.1 D*-Lite Algorithm

This algorithm is used to find out the shortest path from start to goal. This algorithm is flexible as it can be used to change the path of the robot when it detects any obstacle in the path.

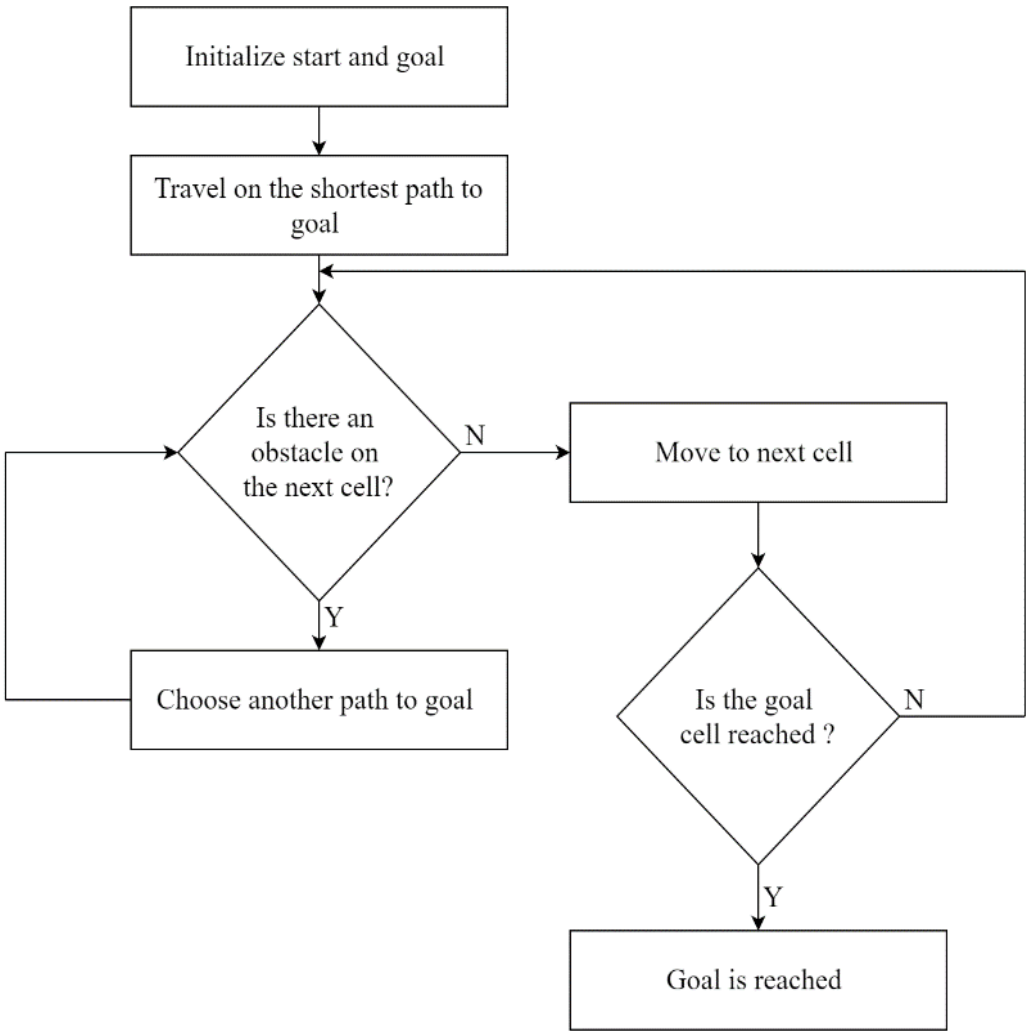

Figure 4-10: Flowchart of D*-Lite Algorithm

D*-Lite searches from the goal vertex to the start vertex. After searching is finished, the path from start to the goal is generated by iteratively moving through the successor from the start. The D*-Lite algorithm maintains path optimality by expanding heuristically towards the start cell. It reuses information from the previous search to repair path in a series of similar searches, which is much efficient than calculating path from scratch.

4.3 Interaction with Data Types

While mapping and path finding, the information of the whole maze and the parameters of the path must be stored by the controller. Different data types are used to store the information about the maze.

4.3.1 1- Dimensional Array

A one-dimensional array (or single dimension array) is a type of linear array. Accessing its elements involves a single subscript which can either represent a row or column index. It is used to store the information of the cells. Each cell is represented by a unique number called the “Cell ID”. The 1-dimension array is used to store the maze which consists of many cells.

4.3.2 Priority Queue

A priority queue is an abstract data type in which each element additionally has a "priority" associated with it. The open list used in D*-Lite algorithm is a priority queue where all the cells of the maze are stored with the help of the key values to determine the priority. The key value of each cell is checked before a cell is stored in the priority queue.

4.3.3 Movement Queue

The movement queue follows the queue principle i.e., FIFO. Here the paths to the goal are stored when the D*-Lite Algorithm is implemented. The start cell is stored at first and then the successor cells are stored until the goal is reached. While moving on the path the cells are popped out to direct the robot to the next cell.

5. IMPLEMENTATION DETAILS

The project is implemented by simulating the robot in a virtual environment. Both the environment and the robot are made in Webots robotics simulator. The controller for the robot is programmed in C++ language.

5.1 Environment Setup

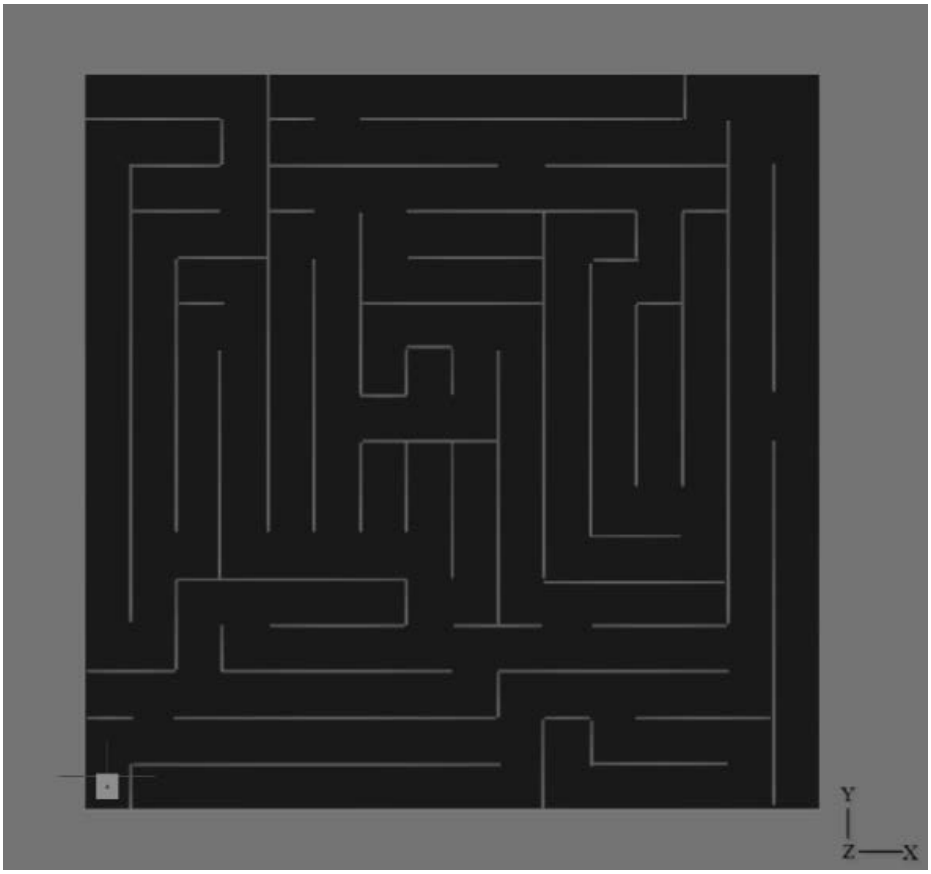

Figure 5-1: Top View of 3D Maze

The environment for the robot is set by creating a floor of dimensions 2880mm × 2880mm where each unit cell is a 180mm × 180mm square. Hence the floor consists 16 × 16-unit cells. The 3-D view of maze with coordinate axis for the maze is shown in figure 5-1.

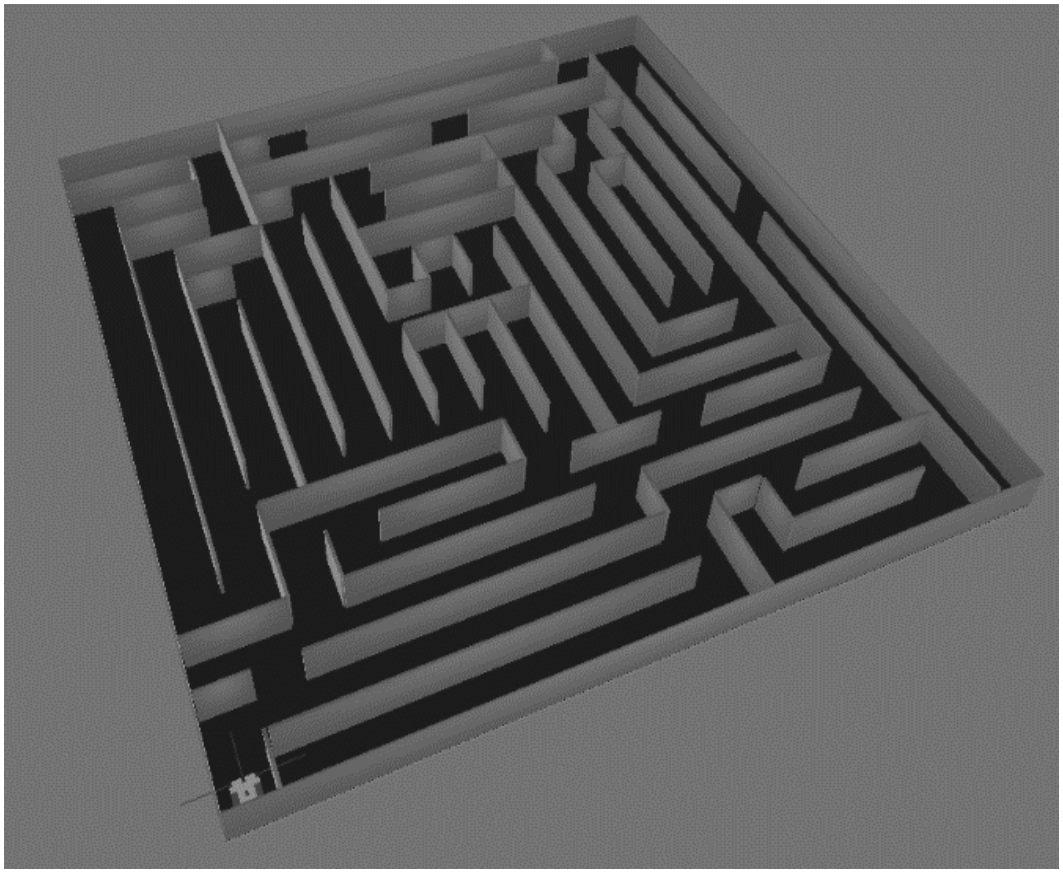

Figure 5-2: Side View of 3D Maze

The walls surrounding the maze and the walls inside the maze are made with the height of 150mm. The width of walls inside is 12mm and the walls surrounding the maze is of 6mm width. Each cell has a pathway of 168mm as mentioned in maze modelling.

5.2 Implementation of Robot

The chassis of the robot is made rectangular with dimensions 90mm × 100mm × 40mm and is fitted with distance sensors, rotational motors, encoders, caster wheel and IMU sensor to traverse its way in the maze.

5.2.1 Distance Sensors

Three distance sensors are used in the robot to detect the walls and path on the maze They are placed pointing the left, front and right side of the robot. The distance sensor provides an analog output of 0-15. The value 15 is outputted when no wall is detected or there is a clear path and the value is less than 15 when wall or any obstacle is detected. The raw value from the distance sensor is the distance data in cm.

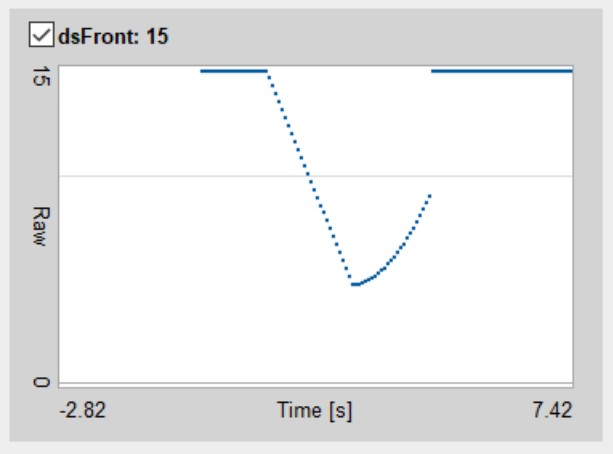

Figure 5-3: Data of Front Distance Sensor

In figure 5-2 the sensor data is available from 0 sec. The data is decreasing gradually while the robot approach towards a wall and the data starts to increase while turning and jumps to 15 as the no obstacle is detected in the turned direction.

5.2.2 Rotational Motors

Two rotational motors are used in the robot to move the robot in the maze. The motor wheels are jointed to the chassis of the robot with the hinge joint which allow the motor to rotate in both directions. The controller adjusts the speed of the wheels when necessary. While turning right, the right motor moves backwards and the left motor moves forward in same speed and vice versa.

5.2.3 Encoders

Two encoders are used in the robot to calculate the distance travelled and to find the position of the robot in the maze. The encoders are attached to the same hinge joint which consists of the rotational motors. The encoder provides the angular rotation made by the wheel to the controller. The angular rotation is in the form of radians and thus linear distance is calculated by multiplying with the radius of the wheel.

When the robot is moving forward, both the wheels rotate with nearly equal speed. The data of angular rotation increases in nearly the same rate while moving straight in the maze. The small difference the rate of change is caused due to the robot's inability to make exact 90 degree turn and stay in the middle of the path.

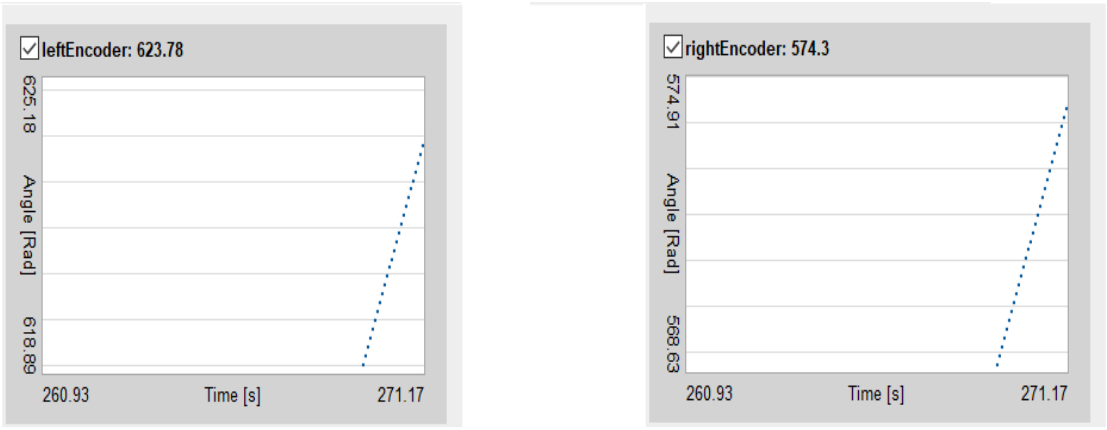

Figure 5-4: Encoders Data while Moving Forward

According to figure 5-3, the distance travelled by left wheel is

$$623.78 \times 0.02m = 12.475m$$

The distance travelled by right wheel is

$$574.30 \times 0.02m = 11.486m$$

The total distance travelled by the robot upto that instant of time is calculated by:

$$\frac{distance\ travelled\ by\ left\ wheel + distance\ travelled\ by\ right\ wheel}{2}$$

$$= \frac{12.475 + 11.486}{2}$$

$$= 11.981m$$

When the robot makes clockwise turns, the left wheel should move in forward direction but right wheel should move in backward direction with same speed due to which the encoder value of left wheel will increase and right wheel will decrease.

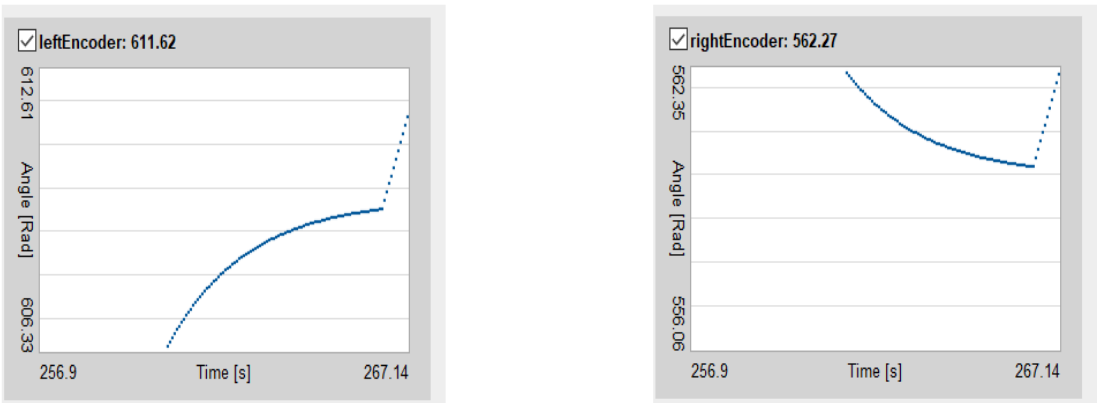

Figure 5-5: Encoders Data while Making Clockwise Turn

When the robot makes counter clockwise turns, the left wheel should move in backward direction but right wheel should move in forward direction with same speed due to which the encoder value of left wheel will decrease and right wheel will increase.

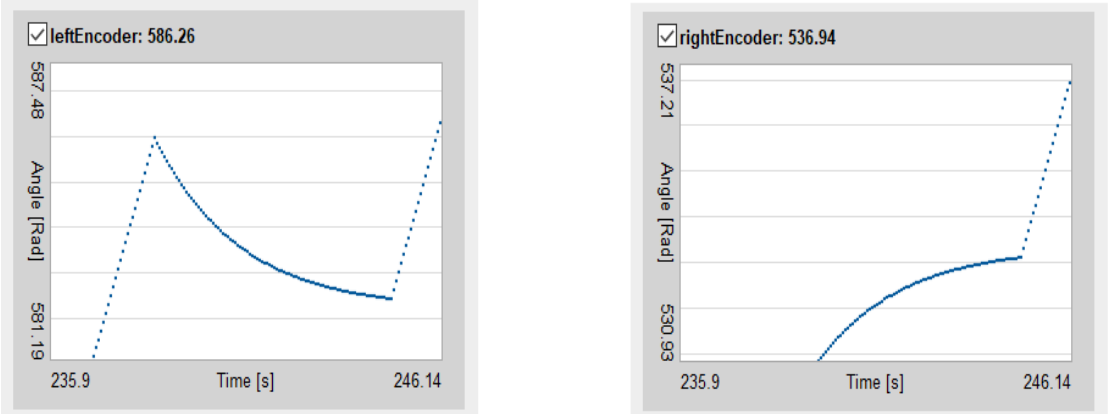

Figure 5-6: Encoders Data while Making Counter Clockwise Turn

5.2.4 Caster Wheel

Two caster-wheel are used to balance the robot. One wheel is placed in front and another is placed at the back of the robot. The caster wheels are jointed to the chassis of the robot using the ball-joint which freely allow the wheels to move in all directions.

5.2.5 IMU Sensor

The IMU sensor is used to calculate the orientation of the robot. It provides the roll, pitch and yaw of the robot which are the orientation of the robot in radian. For the

project’s purpose, only yaw is considered, which is the rotation around Y-axis. The yaw data changes by -1.57 radian for turn in clockwise direction.

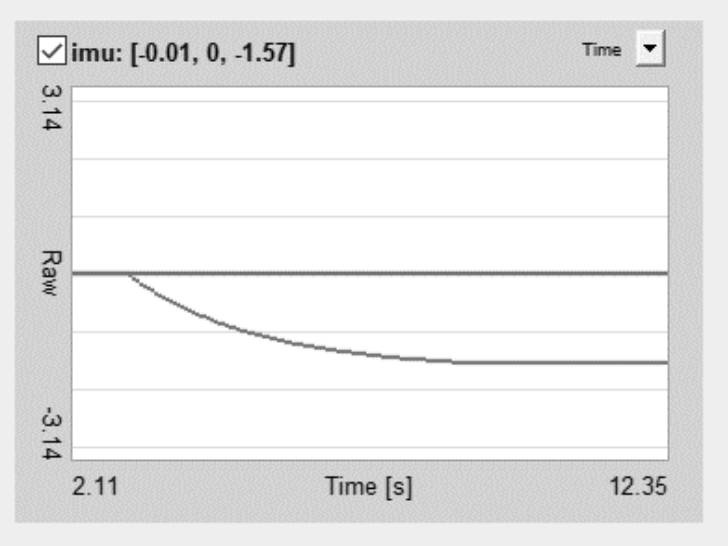

Figure 5-7: IMU Sensor data for Clockwise Rotation

The yaw data changes by +1.57 radian for counter clockwise direction. The change doesn’t take place at an instant as there is some time taken by the robot to make the turn.

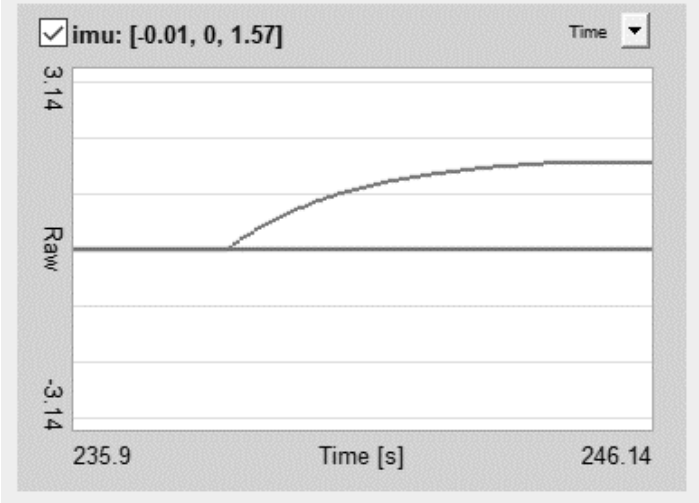

Figure 5-8: IMU Sensor data for Counter-clockwise Rotation

5.2.6 Robot Motion Implementation

In any time-instant, the robot is either moving forward or rotating, with the exception that the robot stops moving when it reaches the goal. The motion of robot is broken down to a simple process of moving to adjacent cell. The robot keeps track of the distance it travelled since last cell. Every time the bot moves 180 mm (length of a cell) ± tolerance, the robot gets the next direction from the algorithm unit. The PID controller corrects the errors in the bot motion and helps the bot to move in straight direction.

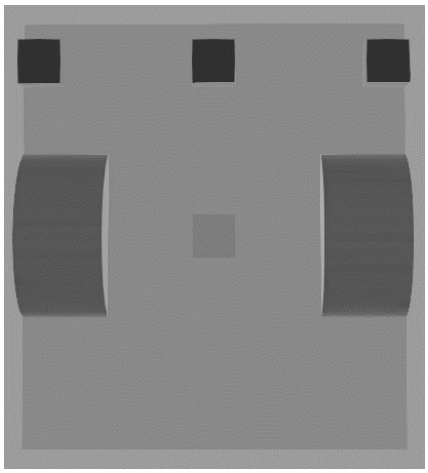

Figure 5-9: 3D Top View of Robot

The three cubes are in the front of the robot are the three-distance sensor and the cube in-between the two wheel is the IMU sensor. The two spheres in either side of the main wheel as shown in Figure 5-9 are the caster wheels of the robot.

Figure 5-10: 3D Side View of Robot

5.2.7 Cell Cost Evaluation

The cost of moving through the wall is provided a very high value. i.e., 1000. The cost of moving to the closed cell is given a varying value. i.e., 500 – no. of unique cells visited. This approach is used so as to return from the dead end as the innermost cell has higher cost value and the outward cell has relatively low cost. If given a constant cost similar to that of the wall, the robot will get stuck in the virtual and real walls and will keep on rotating. Thus, the maximum cost any cell can get is 499 and the minimum is 1. The cost of moving to the cell increases by 1 every time the cell is visited. This prevents the robot from taking the same path next time.

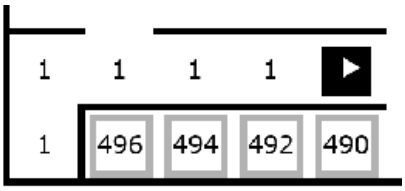

Figure 5-11: Cell Cost Evaluation

5.2.8 Algorithm Implementation

5.2.8.1 PID Controller

The PID controller is used to correct the errors while moving forward. The feedback(β) is calculated by the sum of the current error(e), past error(pE) and total error(tE). and the feedback is calculated as

$$\beta = k_p \times e + k_d . \times (pE - e) + k_i \times tE \quad (5.1)$$

Where k_p = Proportional Gain

k_d = Derivative Gain

k_i = Integral Gain

The behavior of a system is studied to find how the values of k_p, k_d and k_i effect a system. To perform the analysis, the system chosen as an example has an open loop transfer function given by $\frac{1}{s^2 + 10s + 20}$ and the input given is the unit step function.

a) Proportional Control

The proportional gain is analyzed by increasing and decreasing the value of proportional gain and keeping the derivative gain and integral gain of the feedback 0 in the system and thus the behavior of k_p is realized.

When the value of k_p is increased the rise time of the system is decreased but the system will tend to overshoot more and when the value of k_p is decreased the rise time will increase but the system will tend to overshoot less. There will be small change in the settling time for the system as the value of k_p increases.

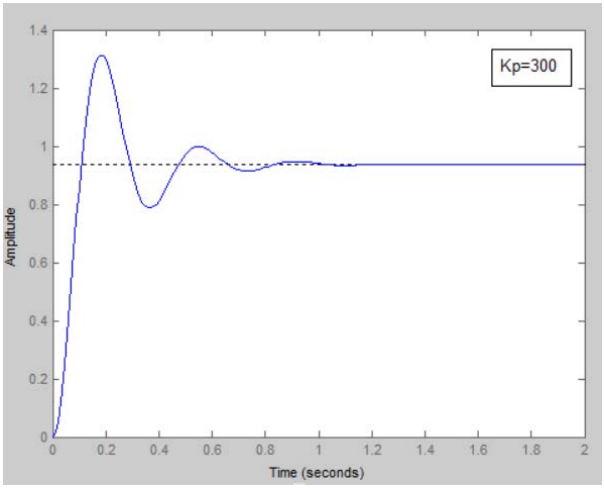

Figure 5-12: Response of System When k_p is High

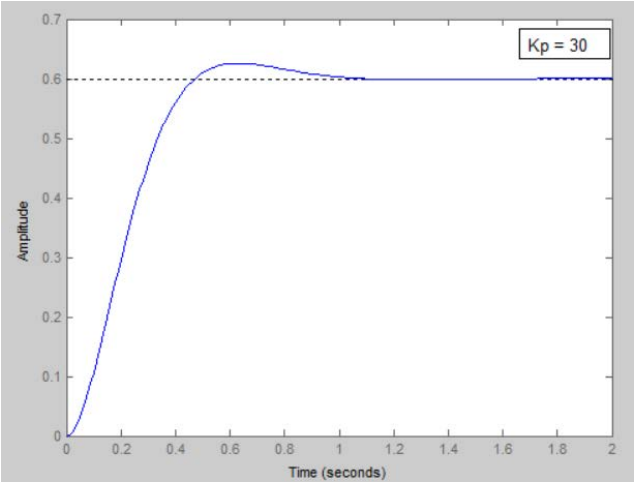

Figure 5-13: Response of System when k_p is low

b) Proportional Derivative Controller

The derivative gain is analyzed by increasing and decreasing the value of derivative gain and keeping the derivative proportional gain and integral gain of the feedback constant and 0 reopen the system and thus the behavior of k_d is realized.

When the value of k_d is increased the rise time of the system is also increased and the system will tend to overshoot less and when the value of k_d is decreased the rise time will decrease and the system will tend to overshoot more. There will be decrease in the settling time for the system as the value of k_d increases.

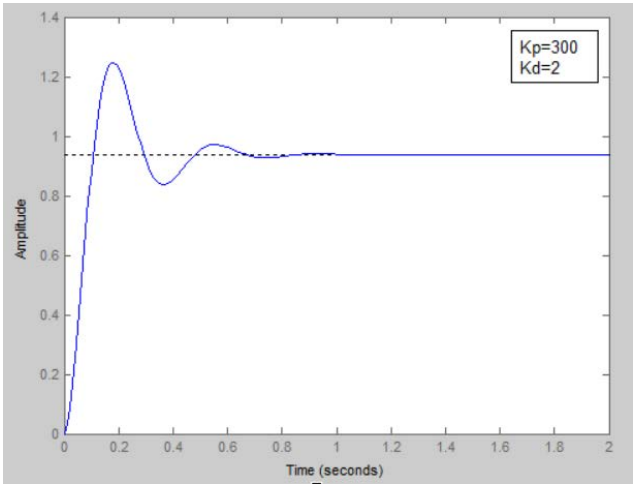

Figure 5-14: Response of system when k_d is low

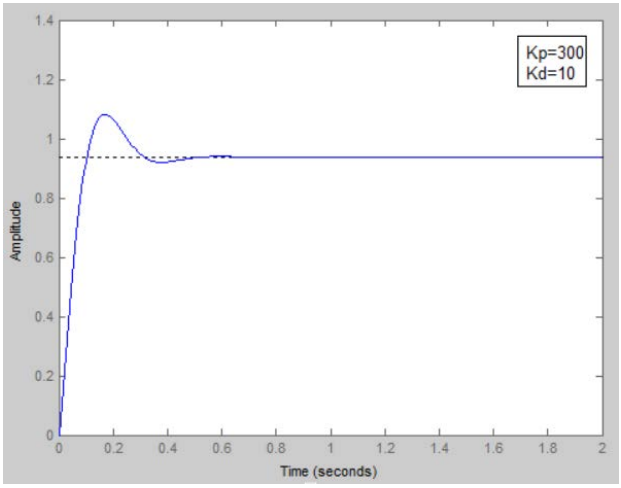

Figure 5-15: Response of system when k_d is high

c) Proportional Integral Controller

The integral gain is analyzed by increasing and decreasing the value of integral gain and keeping the integral proportional gain and derivative gain of the feedback constant and 0 reopen the system and thus the behavior of k_i is realized.

When the value of k_i is increased the rise time of the system decreases but the system will tend to overshoot more and when the value of k_i is decreased the rise time will increase but the system will tend to overshoot less. There will be decrease in the settling time for the system as the value of k_i increases.

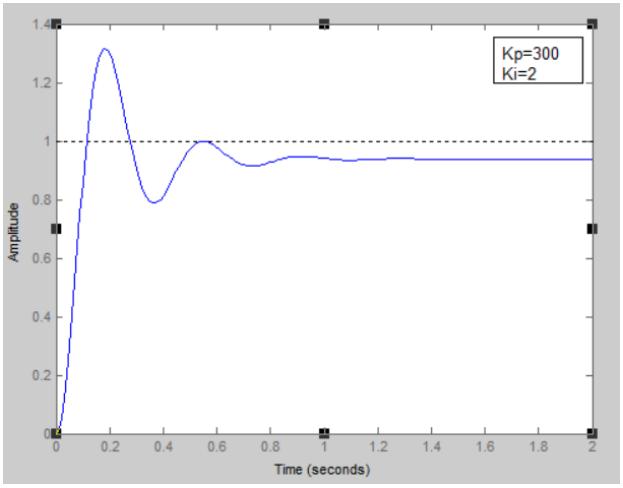

Figure 5-16: Response of system when k_i is low

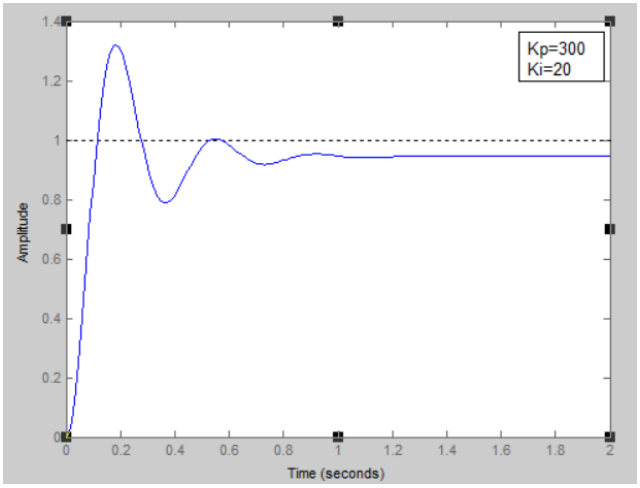

Figure 5-17: Response of system when k_i is high

From the analysis done above table 5-1 describes the conclusion on how the values of k_p, k_d and k_i effect the rise time, overshoot, settling time and stability of a system. The findings are tabulated in table 5-1 below

Table 5-1: Effect of PID parameters on System

Parameter	Rise Time	Overshoot	Settling Time	Stability
$\mathbf{K_p}$	Decrease	Increase	Small change	Degrade
$\mathbf{K_d}$	Decrease	Decrease	Decrease	Improve
$\mathbf{K_i}$	Decrease	Increase	Increase	Degrade

For this project, the robot needs to stay in the middle of the track. To satisfy these criteria, the values of $k_p = 0.05$, $k_d = 0.0001$ and $k_i = 0.00001$ are obtained from PID tuning.

Here k_p is the greatest as it deals with the error with is generated at current instant rather than k_d and k_i which deals with the change in error or the total error. The values of k_d and k_i cannot be ignored because even the slightest change in velocity causes a drastic change in orientation of the robot.

For the robot, the error is taken from the difference of left and right distance sensors. The robot tries to move straight in a maze by keeping equal distance from both sides of the wall. As perfect turns are not possible the robot comes closer to either side of a wall while travelling in the maze. The feedback is used to mitigate this error allowing the robot to stay in center of the grids while travelling in the maze.

The error is calculated by taking the difference of the data provided by the right distance sensor from left distance sensor. The error represents the proximity of the robot to a wall. The feedback in equation 5.1 will regulate the speed in either wheel and make the robot stay in the center. The error is maximum while moving forward after a turn.

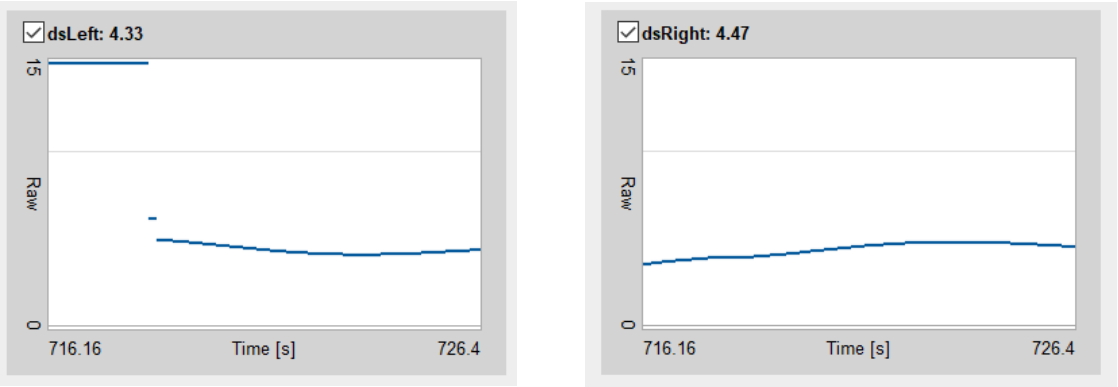

Figure 5-18: Left and Right Distance Sensor Data After a Turn

After a turning is made the feedback aligns the robot in the center of the path until the error tends to zero. In figure below the error is 0.02 which is very close to zero.

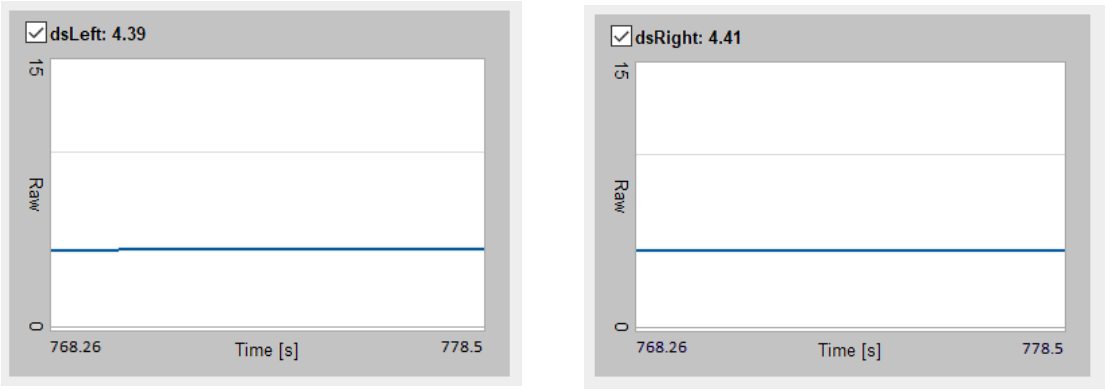

Figure 5-19: Left and Right Distance Sensor Data After a Turn

5.2.8.2 Dead-End Exclusion Algorithm

The Dead-End Exclusion algorithm works by taking the cost of the neighboring cells, and the value from the sensors. The value from the sensors is updated to the current cell and the neighboring cells. Both the current cell and neighboring cells are checked if they are closed or not. If any cells are closed, their cost is updated as required. If any of the neighboring cells are closed, their neighboring cells are checked as well. This happens in a recursive function which ends if all of the neighboring cells are already closed or if the neighboring cells can’t be closed. This process ends when all 256 cells are determined.

5.2.8.3 D*-Lite Algorithm

The D*-Lite algorithm uses two parameters (rhs) and (g) values for expanding the cells from the goal position. The algorithm initializes the both parameters for closed cells as infinity and expands the path from the goal to the start. The goal position initializes rhs as 0 and cell cost for available cells are evaluated. The shortest path is calculated and then successor (id) of the cell is moved into a movement queue.

After obstacle is detected in a cell, the (rhs) value of that specific cell is changed to infinity and the next cells are pushed into the open list. Then the D*-Lite Algorithm calculates the shortest path again and pushes the successor (id) into the movement queue.

When all the paths to the goals are blocked by obstacles, the (rhs) and (g) values of all the cells are infinity and the robot stops as there is no cells to move in the movement queue.

6. RESULTS AND ANALYSIS

The result of the project is explained using the maze used in the maze modelling section. The result obtained from using Dead-End Exclusion algorithm is the result after maze exploration process and the result obtained from D*-Lite algorithm is the result of the goal seeking process.

6.1 Dead-End Exclusion Algorithm

Dead-End Exclusion Algorithm marks the dead-end of the maze, closing the cells by creating a virtual wall in the maze. The following figures are used to explain the result obtained from the working of Dead-End Exclusion Algorithm.

The Dead-End Exclusion Algorithm excludes the dead-ends while navigating the maze. To explain the outcome of this algorithm, certain portion of the maze is taken which is shown within the dotted rectanngle in Figure 6-1.

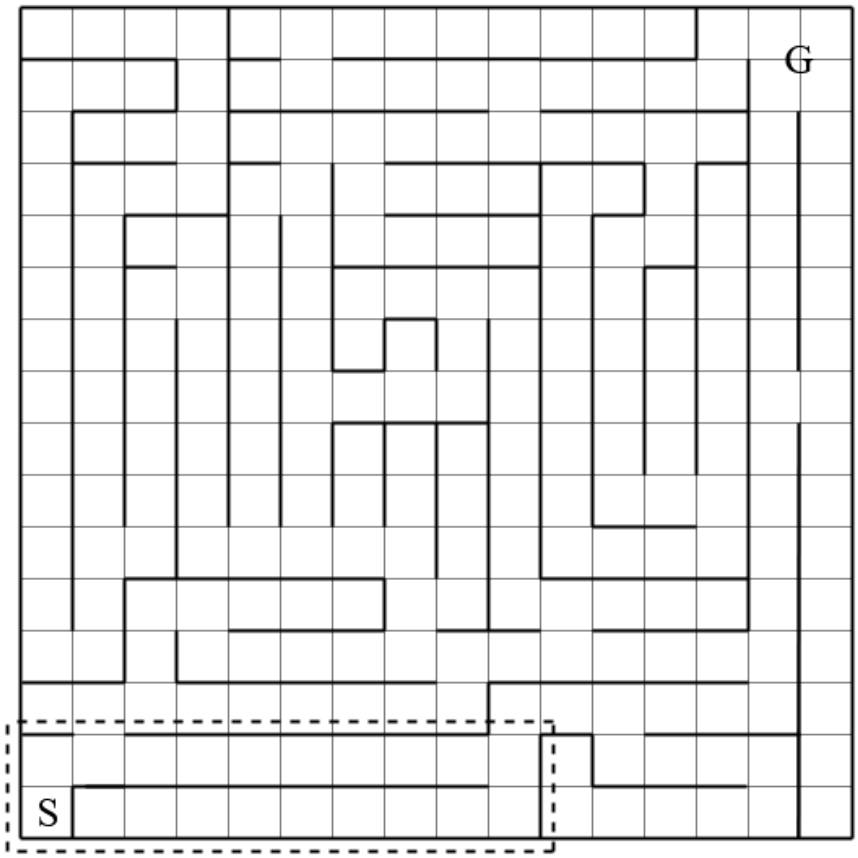

Figure 6-1: Selected Region for Explaining Maze Mapping

In the given maze, the bottom left corner is the starting position and the top right corner is the goal position. Initialization process is performed, in which the cells near to the boundary of the maze are marked once and the cells in the corner are marked twice. The starting cell is not marked in the bottom position. This helps in preventing the starting cell from closing so as to prevent the starting cell from completely shutting the path during goal seeking process.

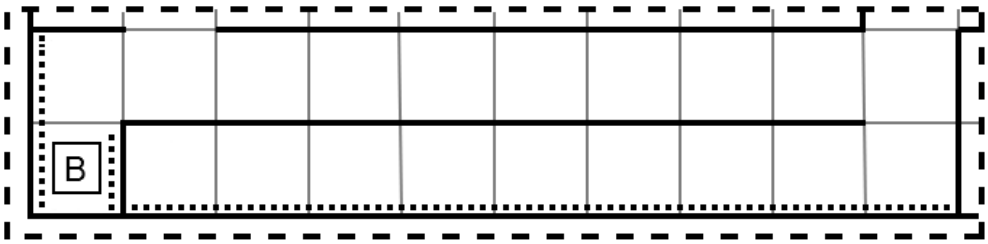

Figure 6-2: Dead-End Exclusion Initialization

The robot takes data from sensors and then updates the current cell and neighboring cells. It then checks if the neighboring cells can be closed. This is shown in figure 6-4.

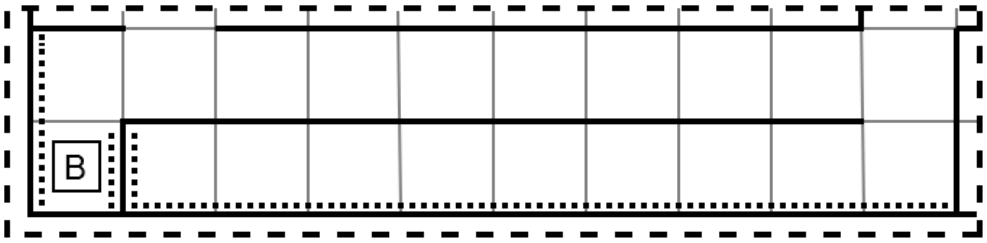

Figure 6-3: Updating Current Cell and Neighboring Cell

The robot moves forward and repeats the process. When the robot reaches the cell on (1,1). It updates its cell value and checks if neighboring cells can be closed. The neighboring cell on (1,0) can be closed. Thus, it is closed and its neighbor's cell value is updated as well, and are checked if they can be closed, if they aren't closed already.

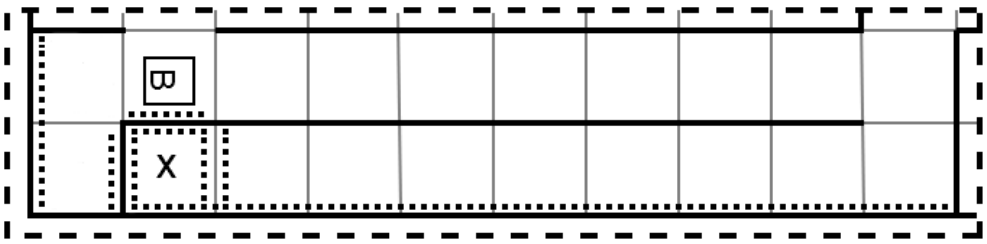

Figure 6-4: Closing Nearby Cell

Now when the bot reaches the end of the second row, it marks the second row and the first row too. The first row satisfies the closing conditions so it closes the cells in the first row without visiting. Hence it doesn't go to the dead ends and still maps the complete maze.

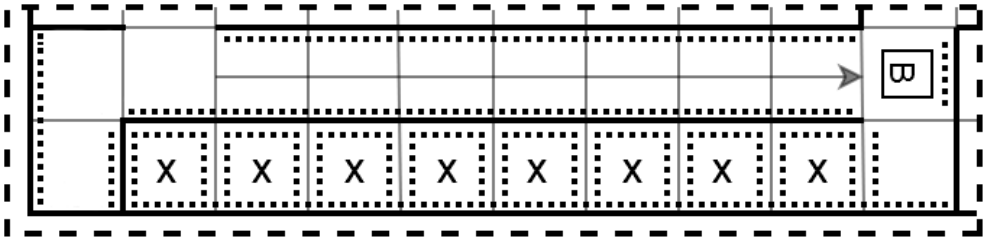

Figure 6-5: Closing Series of Cells

The black square in figure 6-6 represents the robot and it shows the outcome of the Dead-End Exclusion Algorithm after completely mapping the maze in Webots robot simulator. The triangle in the robot represents the direction that the robot is facing. The numbers in the cells represent the cost to move to that cell from adjacent cells.

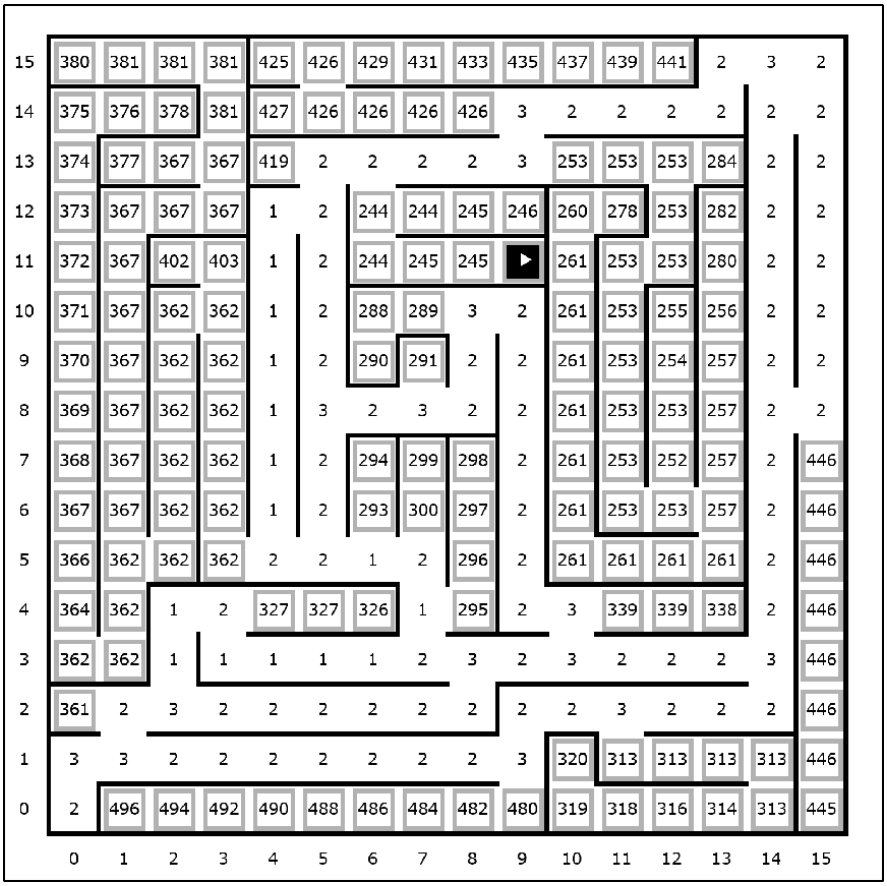

Figure 6-6: Output of Dead-End Exclusion Algorithm in Webots

After the completion of dead-end exclusion algorithm, the robot stays stationary in the last cell and is manually brought back to the starting cell to start the goal seeking process by using D*-Lite algorithm. The ‘s’ key is pressed from the keyboard to start the goal seeking process.

6.2 D*-Lite Algorithm

The maze exploration process has been completed and the goal seeking process is remaining, which is done through the D*-Lite Algorithm. After the complete implementation of D*-Lite Algorithm the following results are obtained. Initially, the start and goal position are figured out from the mapping of the maze. Figure 6-7 and 6-8 shows the (rhs) and (g) values of each cells. The rhs value of goal is initialized to be 0. The rhs value of successive cells from goal is calculated as the sum of g value of previous cell and 1 i.e., assumed cost of moving from one cell to another.

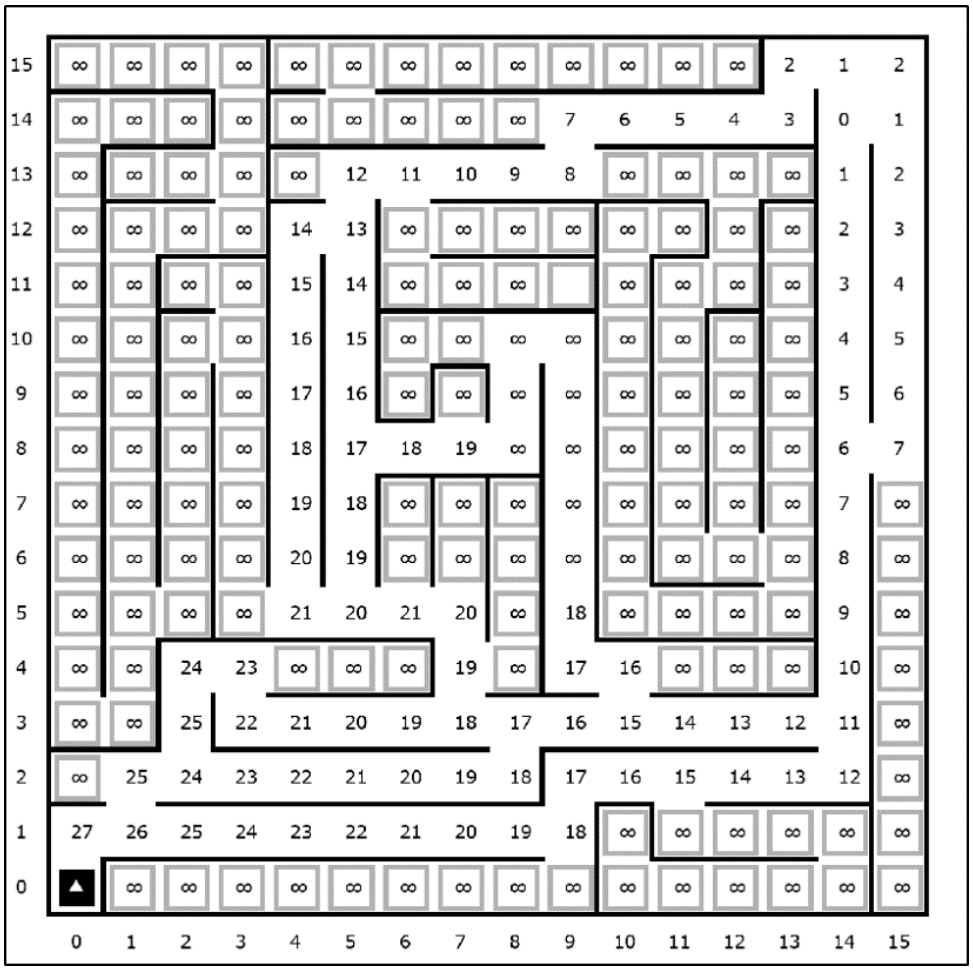

Figure 6-7: (rhs) Values after Initial Shortest Path Calculation

In figure 6-7 and 6-8, the (rhs) and (g) values of the closed cell is infinity, which shows that the cells are not to be visited. The (rhs) and (g) values of the cells are seen to be expanding from the goal to the start.

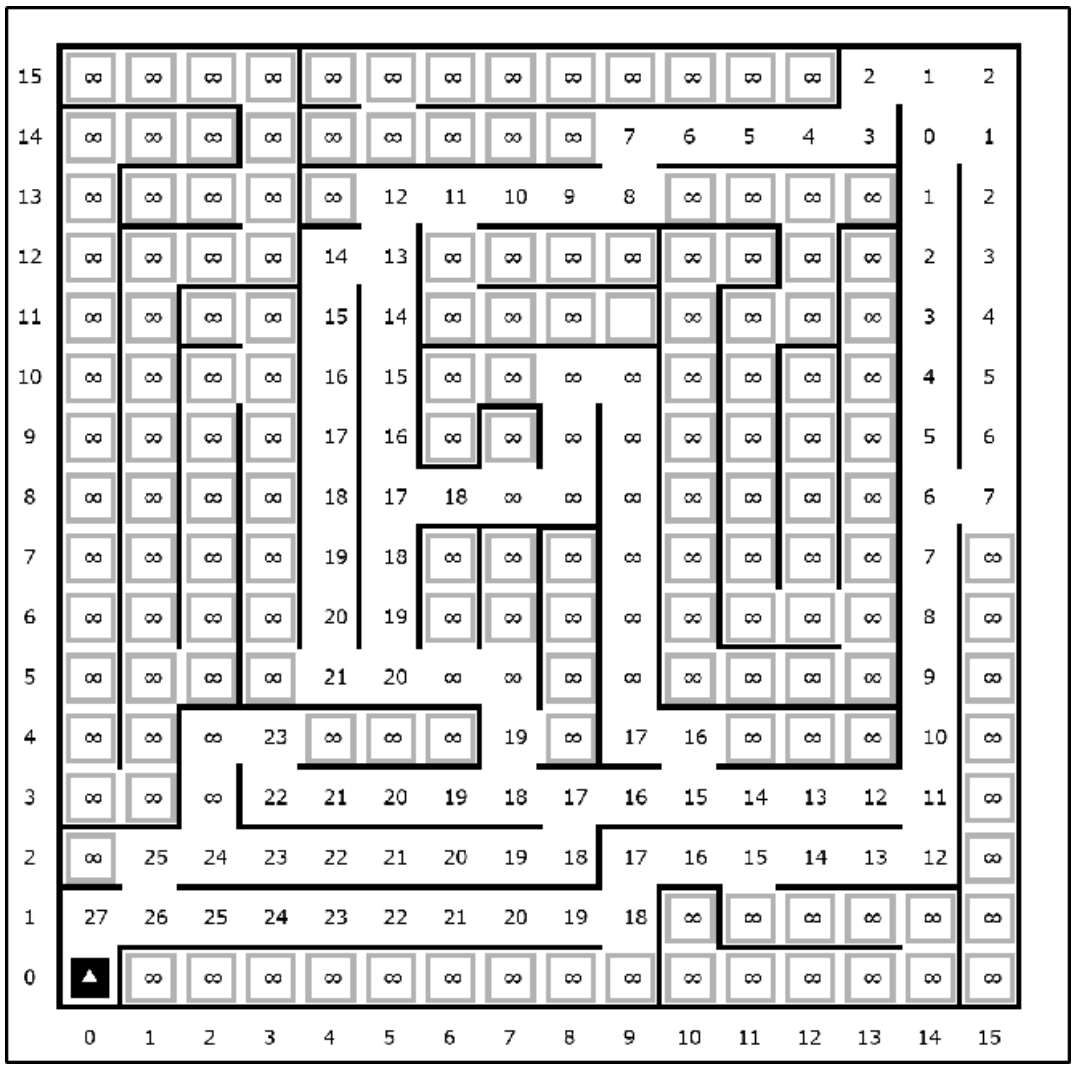

Figure 6-8: (g) Values after Initial Shortest Path Calculation

The figure 6-9 shows the path that the robot should take if there are no obstacles in the path. This path is generated by following the successor (id) from the start to the goal. The (rhs) and (g) values are not shown in the figure 6-9 as the black arrows would have overlapped them. However, the (rhs) and (g) values of the cells are still there and is equal to the (rhs) and (g) values shown in the figure 6-7 and 6-8.

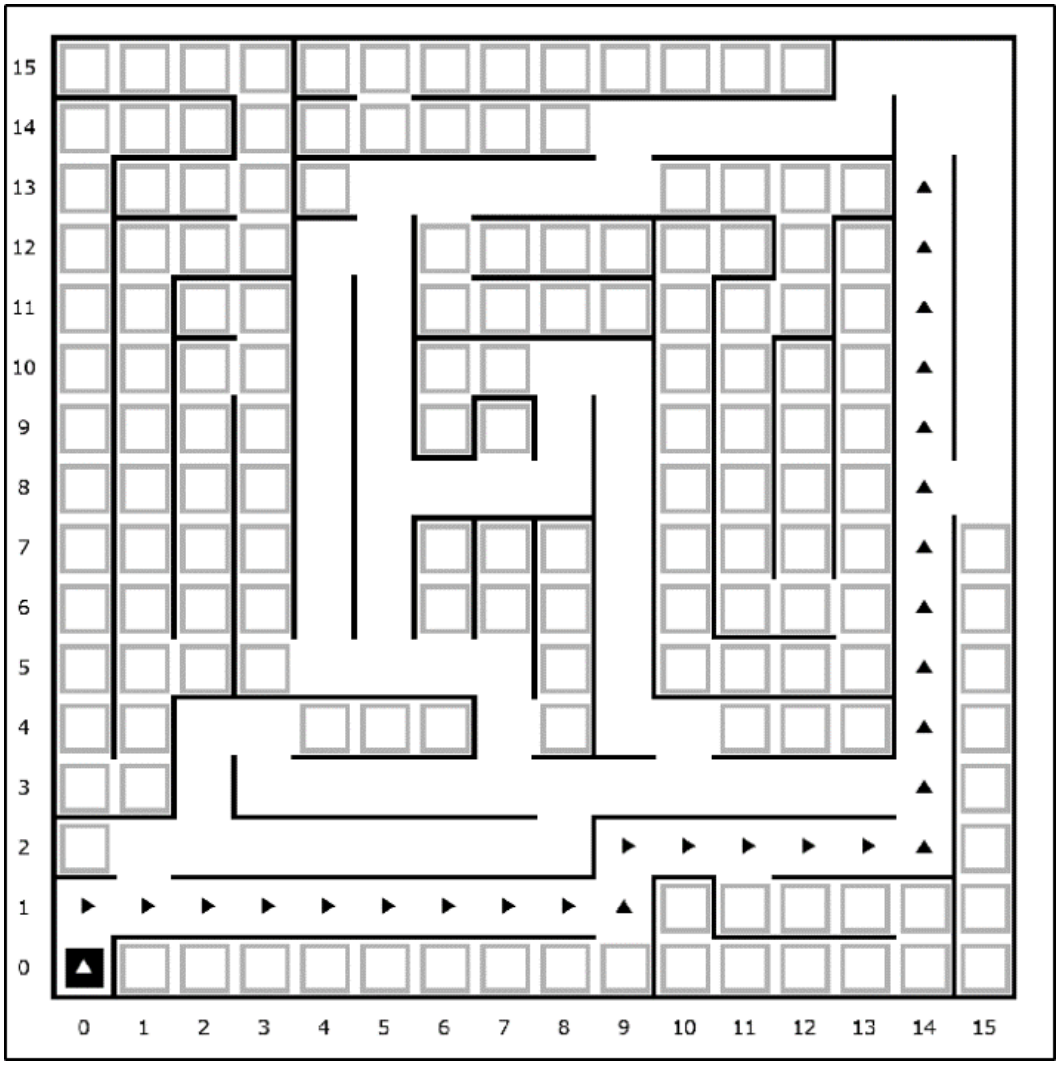

Figure 6-9: Initial Shortest Path found

The figure 6-10 shows that the robot encounters an obstacle while on (14,12). When an obstacle is detected in the path, the current cell is considered as the start position and the (rhs) and (g) value of blocked cell is re-evaluated as shown in figure 6-10 and 6-11. The robot then finds another path to the goal.

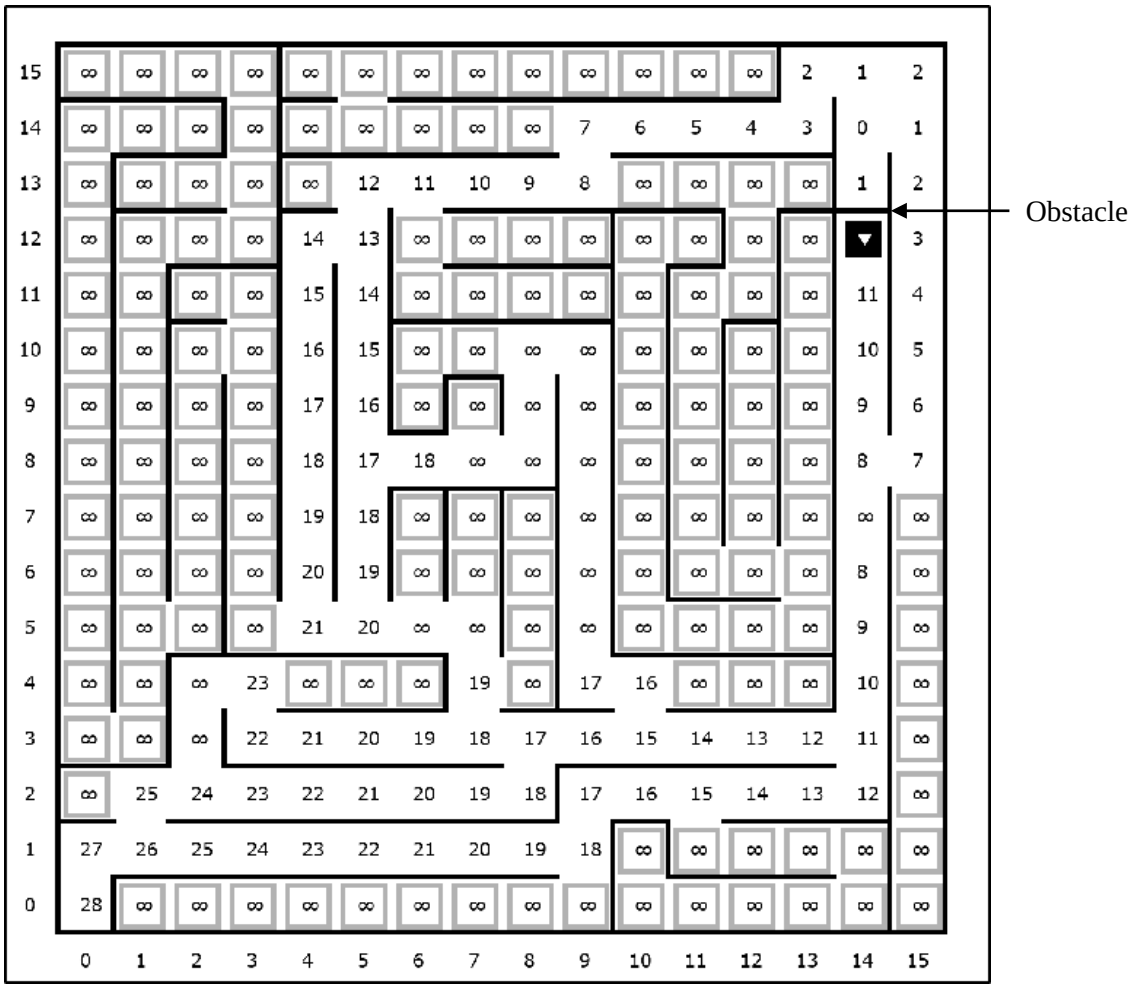

Figure 6-10: Change in (rhs) Values When Obstacle is Detected

In figures 6-10 and 6-11, the cell (7,8) has different (rhs) and (g) value. This phenomenon occurs when the cell is pushed in the open list but is not popped out. Thus, the cells are not further expanded from that cell. When both the (rhs) and (g) values of a cell are equal, the cells are expanded towards its neighboring cells in search for the shortest path to the goal.

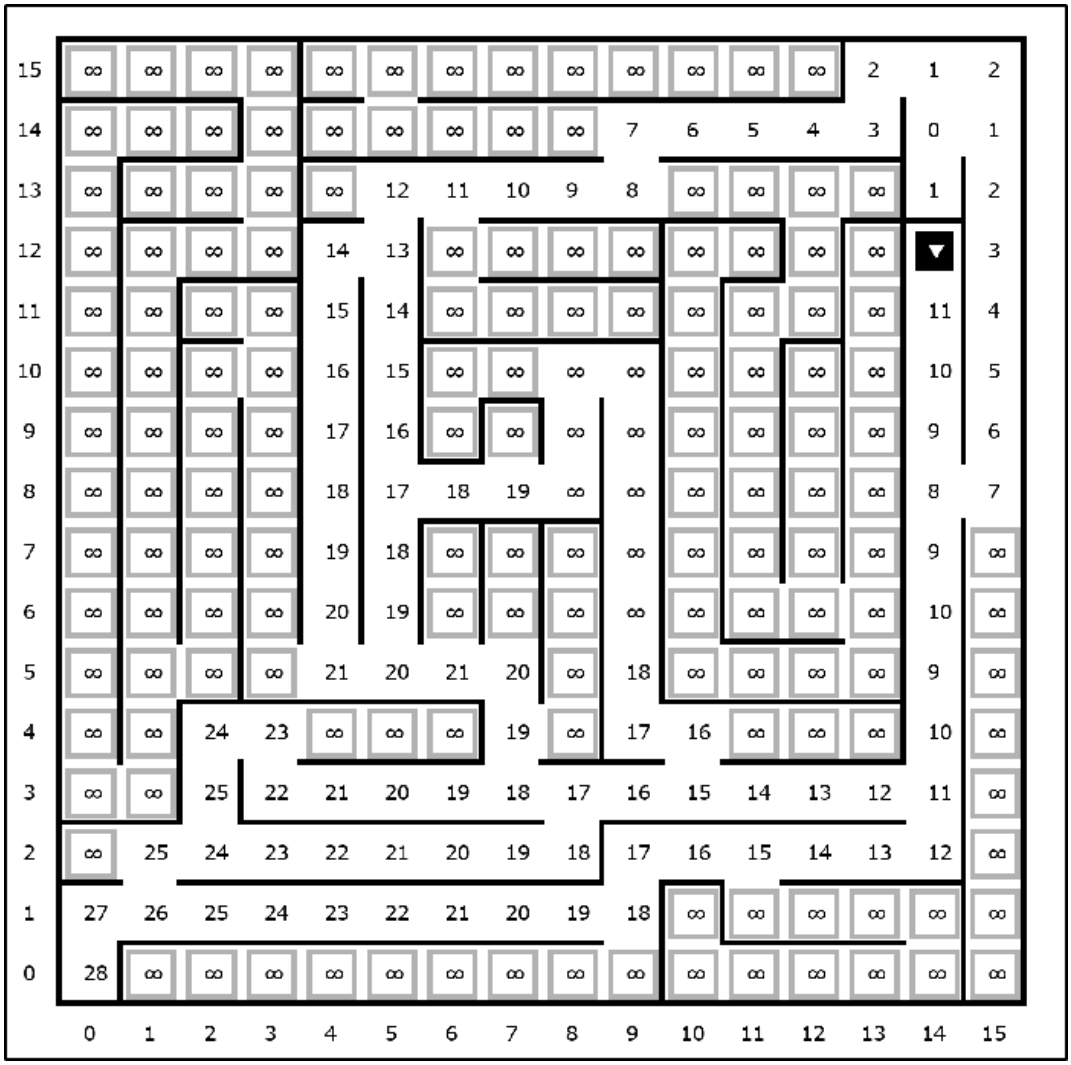

Figure 6-11: Change in (g) Values When Obstacle is Detected

The figure 6-11 shows the change in (g) values when an obstacle is detected. Since the g values change, the shortest path to the goal also changes. The new shortest path is shown in the figure 6-12.

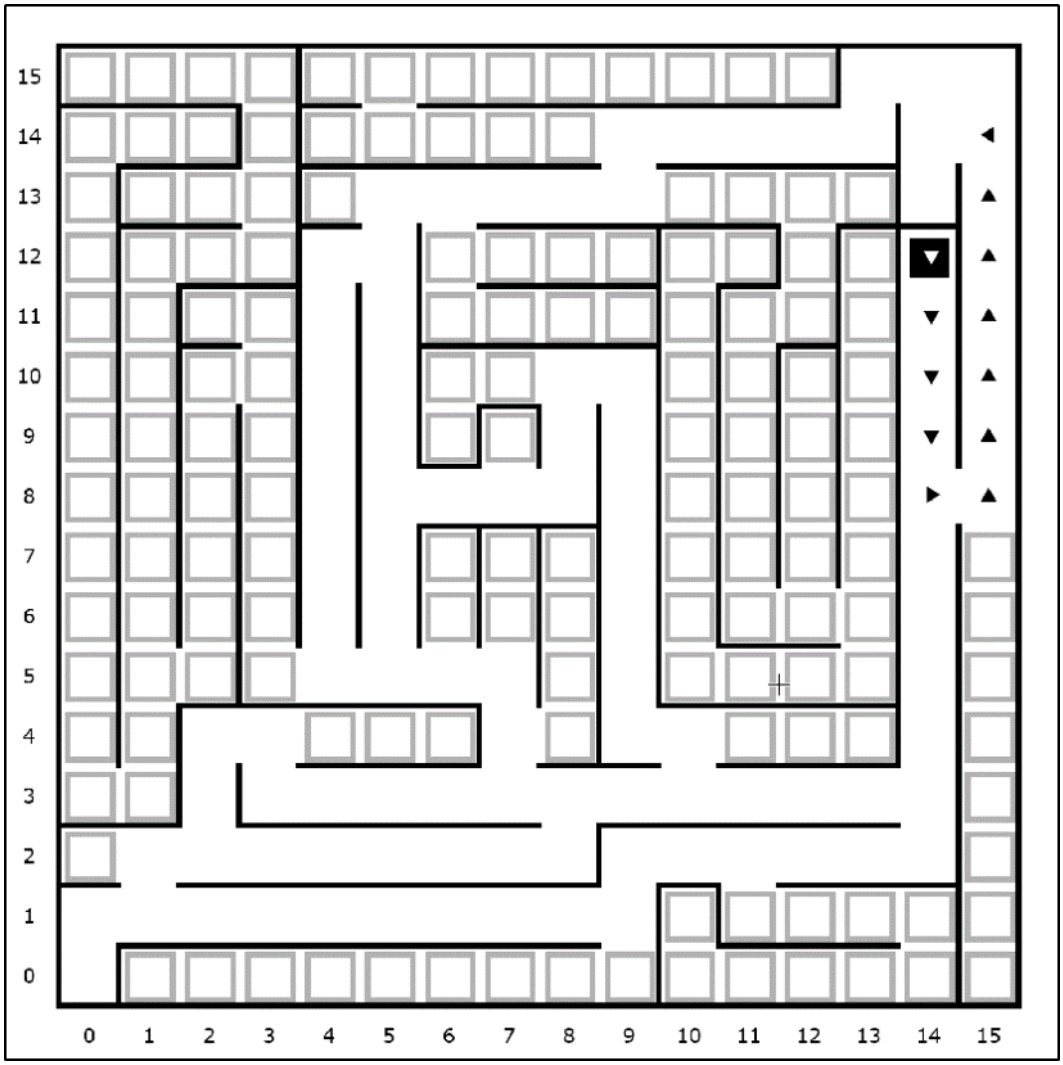

Figure 6-12: New Path to Goal After Obstacle is Detected

After an obstacle was detected between the two cells (14,13) and (14,12), the robot takes the next shortest path and travel on the path shown in figure 6-13. The robot calculates another shortest path every time it encounters an obstacle. This process continues until there is no path remaining to the goal, in which case the robot stops right there.

All paths to the goal are closed by putting obstacles in between cells (14,13) and (14,12), (15,13) and (15,12) and finally between the cells (13,15) and (13,14).

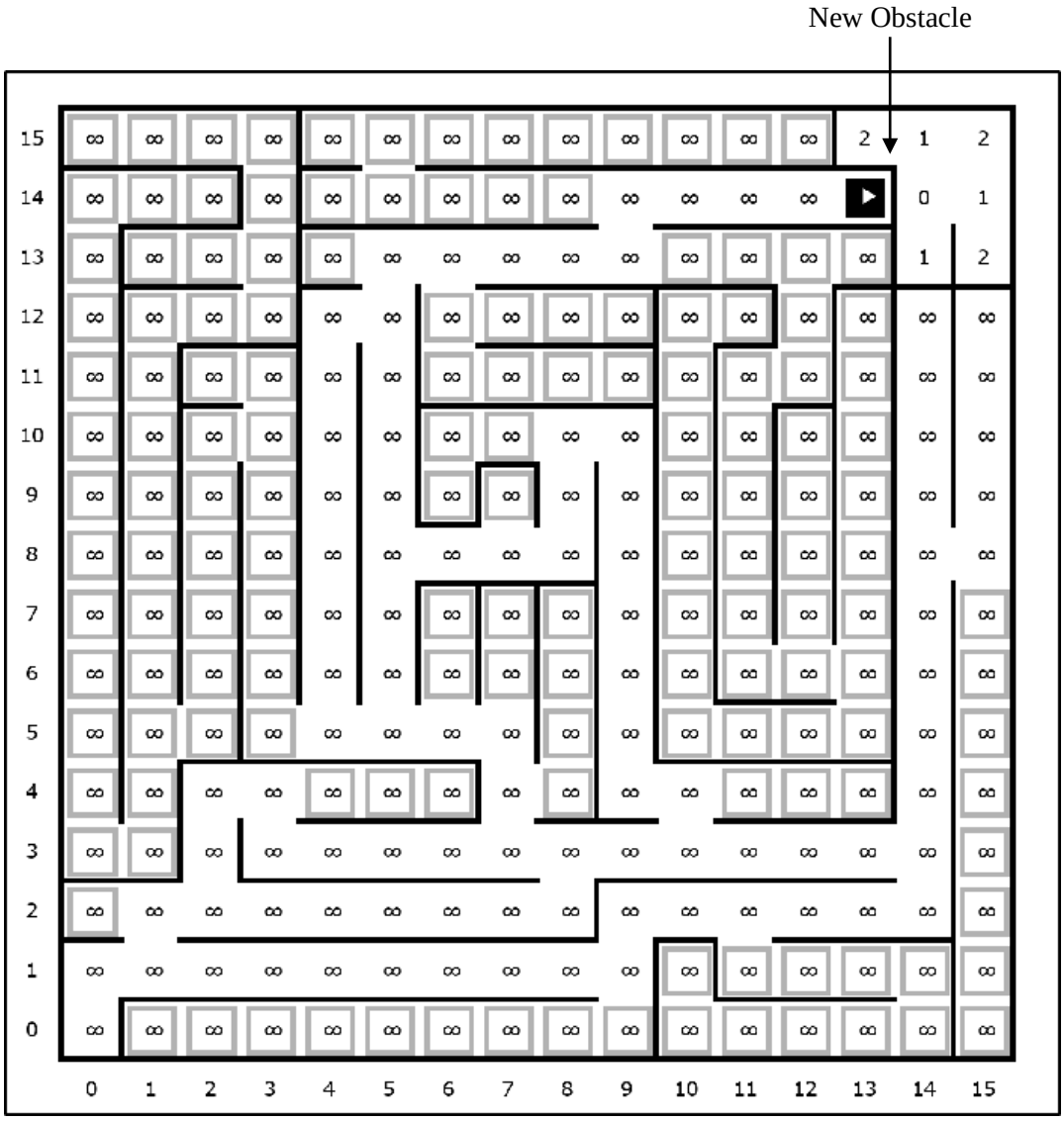

Figure 6-13: (rhs) Values When All Paths are Closed

When all paths are closed with obstacles, the (rhs) and (g) values are infinity. That means that no cells should be travelled and the robot should stay stationary in the path. When the obstacles block the path to the goal, the expansion of the cells from goal position is confined to only a small portion of the maze and the (rhs) and (g) values of the cells outside that space on the maze were assigned as infinity as shown in figure 6-13 and 6-14.

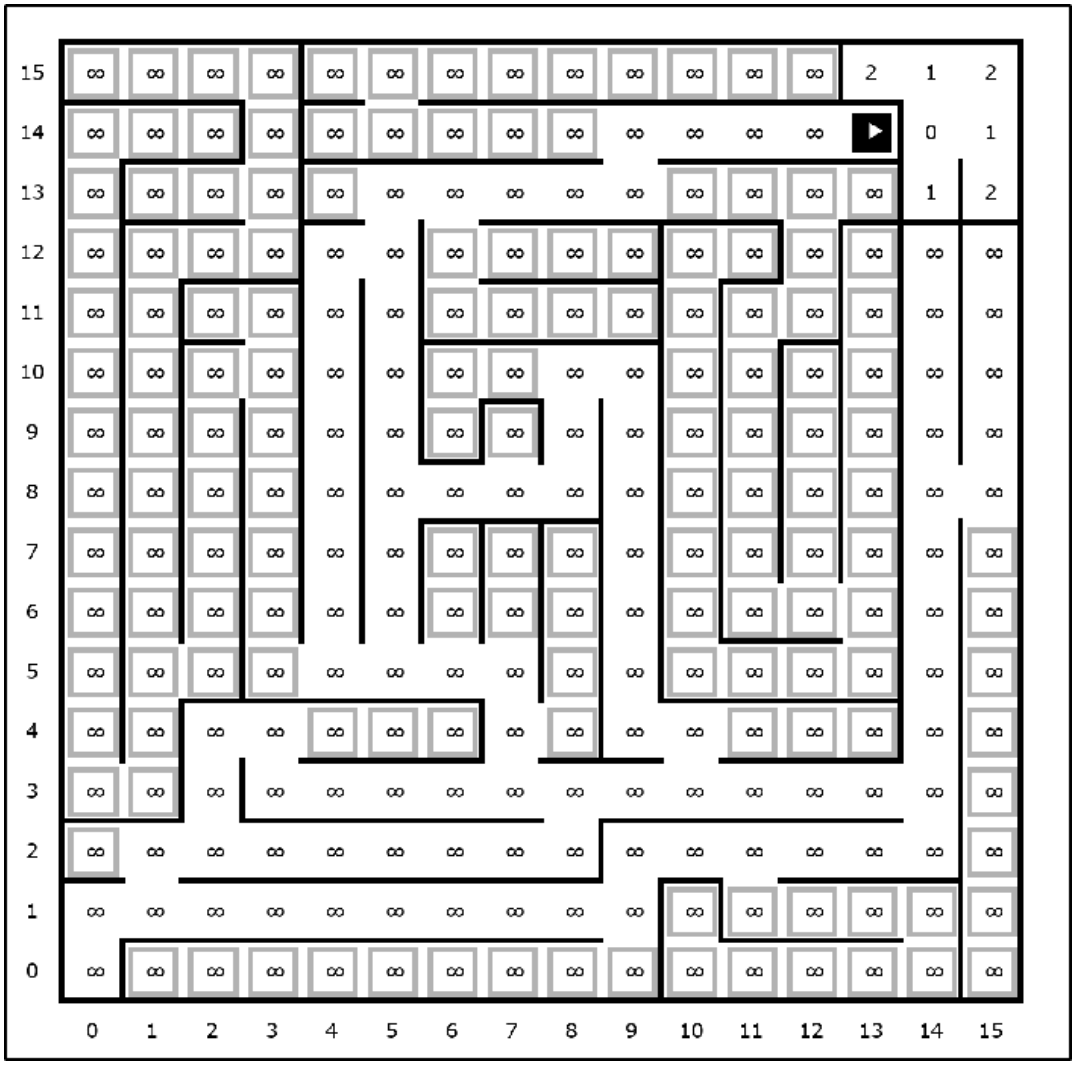

Figure 6-14: (g) Values When All Paths are Closed

When all paths are closed, the robot stays stationary in the path. This can be seen in the figure 6-15, where no arrows are seen to be present.

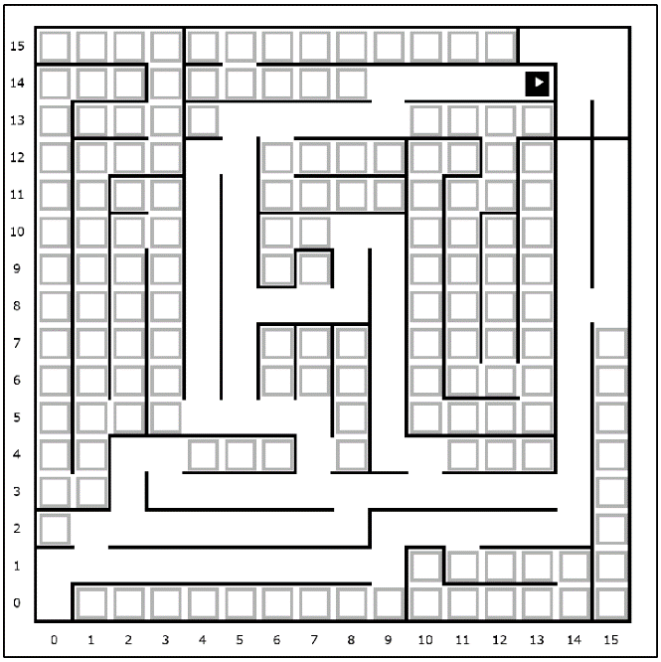

Figure 6-15: Robot's Behavior When no Path to Goal

The results of D*-Lite Algorithm explained above is for the case when the goal is on a corner of the maze. Similar results were obtained, when the goal position is moved from the corner to a random position in the maze. The path found by the robot for non-corner goal is shown in figure 6-16.

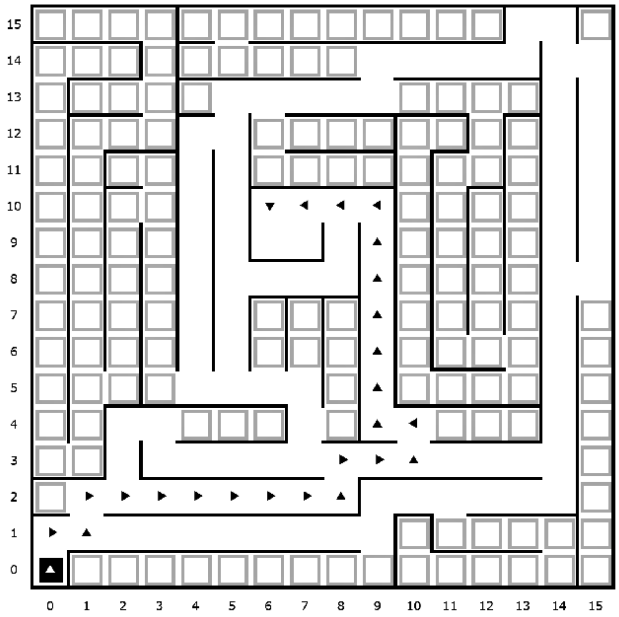

Figure 6-16: Result Obtained From D*-Lite Algorithm for Non - Corner Goal

7. FUTURE ENHANCEMENT

There is room for improvement in every project and this project is not an exception. There are some rooms for improvement in different functionalities. The chassis of the robot designed for this project is rectangular, for more speed, the chassis can be made circular which allows for better movement in the maze. For the localization of the robot, simple tracking of the encoder data is done. This process can be made more efficient by using Simultaneous Localization and Mapping techniques.

While moving on the maze the robot only makes 90º turns. For further improvement on this project, the robot could be made to move diagonally by which it will allow the robot to have 8 possible cells to access. Also more sensors can be added at an angle of 45º to the robot which will allow detecting the walls more quickly while the maze mapping process is being carried out.

Improvements can be made to the mapping of the maze. The robot does not find the removals of walls which could open an even shorter path to the goal. By using a camera from a top view during the goal-seeking process, the information about the removals of walls can be sent to the robot. The camera can take pictures of the changing maze and use image processing techniques to update the map of the maze.

After the mapping of the maze is finished, the goal-seeking is started. Different sampling-based algorithms can be used for the goal-seeking process. RRT and RRT* are the sampling-based algorithms that could be used. In these algorithms, nodes are generated, which are the potential points where the robot could move. In RRT the newly generated nodes are connected to the nodes which are closest to it but in RRT* the newly generated nodes are connected to the nodes which are closest to the starting node.

This project is the simulation of the maze-solving robot in a virtual environment. This project can be implemented in the real world using different electronic components. During the construction, the slip on the wheels should be taken into consideration, which will bring noise to the encoder data and the turning is also affected. Similarly, the noise should also be considered in the data of the distance sensor. For better performance in finding the walls, the ultrasonic distance sensor could be used.

8. CONCLUSION

In conclusion, the work of this project could be categorized into two phases the designing phase and the algorithm implementation phase. In the designing phase, a 16×16 maze was designed where each cell has a dimension of 180mm×180mm. Walls were designed of width 6mm and 150mm height and were put in between cells leaving a pathway of 168mm. The start position was put on a corner surrounded by walls on three sides and the goal position was a 2×2 square. After the maze was designed, the robot was constructed with the size of 100mm×90mm×40mm containing two wheels of diameter 40mm and width 20mm and caster wheels of diameter 15mm and height 15mm.

After the designing phase was completed, the algorithm implementation phase was started. Here the maze solving process was carried out in two steps, the maze exploration process which was carried out through Dead-End Exclusion Algorithm and the goal seeking process which was done through the D*-Lite Algorithm. In the maze exploration process, the Dead-End Exclusion Algorithm mapped the entire maze, closing all the cells which led toward a dead-end and creating a virtual wall so that the robot did not move towards those cells. When all the 256 cells were mapped, the robot stopped at its current position and was brought back to the starting position manually and the goal of using Dead-End Exclusion Algorithm was completed. After the maze exploration was concluded, the goal seeking process was carried out. Here the D*-Lite Algorithm found the shortest path from the start to the goal. When an obstacle was detected on the path found, the robot took the next shortest path and moved toward the goal. The robot stopped moving when it had reached the goal or when there was no path to goal and the objective of using D*-Lite Algorithm for goal seeking was completed.

To sum up, this project has successfully fulfilled all the objectives that it intended to complete initially by creating a robot in a virtual environment that could solve a maze with real-time obstacles using a combination of both the Dead-End Exclusion Algorithm and the D*-Lite Algorithm.

9. APPENDICES

Appendix A: Project Schedule

Table A: Gantt Chart of Project Schedule

Project Start Date	Sep,2020		Project End Date	Feb,2021							
Task	Progress (%)	Sep	Oct	Nov	Dec	Jan	Feb
Brain Storming and Topic Discussion	100						
Documentation	100						
Follow Webots Tutorial	100						
Learn Maze Solving Robots in Webots	100						
Learn Dead-End Exclusion Algorithm	100						
Learn D*-Lite Algorithm	100						
Maze 3D Structure Design	100						
Robot Structure Design	100						
Program Robot Control	100						
Program Dead End Exclusion Algorithm	100						
Test and Debug Dead End Exclusion Algorithm	100						
Program D*-Lite Algorithm	100						
Test and Debug D*-Lite Algorithm	100						

References

[1] micromouseonline.com, "History of Micromouse," [Online]. Available: http://www.micromouseonline.com/micromouse-book/history/. [Accessed September 2020].

[2] L. K. Li and F. Y. Annaz , "Implementation of the Trémaux Maze Solving Algorithm to an," in *ICEIC*, Kota Kinabalu, Malaysia , 2014.

[3] X. Li, X. Jia, X. Xu, J. Xiao and H. Li, "An improved algorithm of the exploring process in Micromouse Competition," in *School of Automation Science and Electrical Engineering*, Beijing, 2010.

[4] S. Tjiharjadi, M. C. Wijaya and E. Setiawan, "Optimization Maze Robot Using A* and Flood," *International Journal of Mechanical Engineering and Robotics Research*, vol. 6, no. 5, pp. 366-372, 2017.

[5] N. H. Barnouti, S. S. M. Al-Dabbagh and M. A. S. Naser, "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm," *Journal of Computer and Communications*, no. 4, pp. 15-25, 2016.

[6] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in *IEEE International Conference on Robotics and Automation*, San Diego, 1994.

[7] F. A. Raheem and U. I. Hameed, "Path Planning Algorithm using D* Heuristic Method," *American Scientific Research Journal for Engineering, Technology, and Sciences*, vol. 49, no. 1, pp. 257-271, 2018.

[8] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 846, pp. 846-894, 2011.

[9] Y. Kuwata, G. Fiore and E. Frazzoli, "Real-time Motion Planning with Applications to Autonomous Urban Driving," *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, vol. 18, no. 5, pp. 1105-1118, 2009.

[10] K. R. Kozlowski, "RRT-path – A Guided Rapidly Exploring Random Tree," in *Robot Motion and Control*, London, Springer-Verlag London, 2009, pp. 307-316.

[11] L. G. Veras, F. L. Medeiros and L. N. Guimara˜es, "Rapidly exploring Random Tree* with," *International Journal of Advanced Robotic Systems*, vol. 16, no. 1, pp. 1-9, 2019.