OPTIMIZATION ALGORITHM OF AUTONOMOUS MAZE SOLVING ROBOT

ABD RAHIM BIN KASIMAN

A project report submitted in partial
fulfillment of the requirement for the award of the
Degree of Master of Electrical Engineering

Faculty of Electrical and Electronics Engineering
Universiti Tun Hussein Onn Malaysia

JULY 2015

ABSTRACT

Maze solving has gained increasing attention in the field of Micromouse competition and Intelligent Robot. The first maze solving using mathematical approach is done by Leonhard Euler in 1736 after solving a problem known as the Seven Bridges of Königsberg. Since then, several algorithms which originate from graph theory and non-graph theory were developed. The main objective of this project is to study the efficiency of the famous graph theory based algorithm which is Dijkstra's Algorithm in term of the shortest distance covered and time taken to solve the maze. A benchmark will be compared to a non-graph theory which is the famous Left Hand Rule Algorithm. To compare the algorithms efficiency, each algorithm are experimented by using a Zumo Pololu mobile robot. The experimental result obtained will lead towards a conclusion about the behavior, nature and efficiency of each algorithm.

ABSTRAK

Penyelesaian maze telah mendapat perhatian yang semakin meningkat dalam bidang persaingan Micromouse dan Inteligent Robot. Penyelesaian pertama maze menggunakan pendekatan matematik dilakukan oleh Leonhard Euler pada 1736 selepas menyelesaikan masalah yang dikenali sebagai '*The Seven Bridges of Königsberg*'. Sejak itu, beberapa algoritma yang berasal dari teori graf dan teori bukan graf telah dibangunkan. Objektif utama projek ini adalah untuk mengkaji keberkesanan algoritma yang terkenal berdasarkan teori graf iaitu Algoritma Dijkstra dari segi jarak terdekat dari tempat permulaan dan tempat pengakhiran dan masa yang diambil untuk menyelesaikan maze. Penanda aras akan dibangdingkan dengan teori bukan graf iaitu Left Hand Rule yang dikenali ramai. Untuk membandingkan kecekapan algoritma, setiap algoritma dimuat turun dan diuji dengan menggunakan robot mobil Zumo Pololu. Hasil pengujian yang diperoleh akan membantu memberi kesimpulan mengenai tingkah laku, sifat dan kecekapan setiap algoritma.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

°C	-	Degrees Celcius
F	-	Force
I_L	-	Load Current
V_S	-	Voltage Supply
R_B	-	Base Resistor
h_{FE}	-	Transistor Current Gain
V_{CC}	-	Microcontroller 5v Voltage
c_f	-	Fixed Cost
v	-	Volume (Number of Units Produced)
c_V	-	Variable Cost Per Unit
p	-	Price per Unit
A	-	Ampere
Ω	-	Ohm
V	-	Voltage
Mm	-	Millimeter
M	-	Micro

LIST OF ABBREVIATIONS

PLC	-	Programmable Logic Controller
PIC	-	Programmable Interface Controller
KV	-	Kolej Vokasional
VDC	-	Direct Current Voltage
SPST	-	Single Pole Single Throw
SPDT	-	Single Pole Double Throw
RM	-	Ringgit Malaysia
I/O	-	Inputs / Outputs
AC	-	Actuating Current
DC	-	Direct Current
V	-	Voltage
F	-	Farad
CPU	-	Central Processing Unit
RAM	-	Random Access Memory
ROM	-	Read Only Memory
GND	-	Ground
ILP	-	Institut Latihan Perindustrian

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Project overview

A maze is a complex puzzle where the solver has to a find a path between the entry and exit of the maze. Technically, a maze is not similar to a labyrinth where the labyrinth has only a single through route twists and turns but without any junction and is considered much easier than a maze itself [1].

Leonhard Euler is the first person to solve a maze using mathematical approach. He solved the problem known as the famous Seven Bridges of Königsberg in 1736, Prussia [2]. The city includes two large islands which were connected to each other with the mainland by seven bridges.

The problem statement for the Seven Bridges of Königsberg is to ensure that a route is to be designed so that each bridges is crossed once are and able to return to the starting point. The route was able to solved by implementing a system of nodes and vertices or currently known as a graph theory. This solution is considered to be the first theorem of graph theory, specifically of planar graph theory [3].

Research in robotic maze solving problems has been highly active for the past few decades in both robotics and artificial intelligence. The general objective for this kind of problem is to use a mobile robot to navigate an unknown or semi-known environment and to accomplish some specific tasks. Generally, a common task is to navigate from the starting point, to the goal, and sometimes back to the starting point again.

Most maze solving techniques are based on graph theory where mazes without loops are equivalent to a branch of a tree where there are junctions and eventually a deadlocks if the route is not correct [4]. Graph theory algorithm are said to be more efficient compared to non-graph theory [5]

IEEE has held the Micromouse Competition every year since 1970's. The contest has been gaining favour since the first launch. Due to this, many techniques was developed to achieve the shortest path and fastest time to solve the maze. Nowadays, through advancement of technologies, hardware used has evolved and can withstand much more complex and sophisticated algorithm to solve the maze. In real application, maze solving algorithms are widely used in autonomous vacuum cleaners, lawn mowers, car park systems and many more.

1.2 Problem Statement

Maze solving robot needs to be instructed with a maze solving algorithm in order to solve a maze intelligently. Due to this, a variety of algorithms can be implemented in the mobile robot. However, the objective of the maze solving needs to be achieved which are in terms of the shortest distance travelled and time taken to solve the path. Therefore, this project is to compare two types of maze solving algorithms which are 1 algorithm for graph theory and 1 algorithm for non-graph theory in terms of its efficiency.

1.3 Project Objective

The objectives of this project are to develop an optimum algorithm of graph theory which is Dijkstra's Algorithm in term of the shortest distance and time taken in solving the maze. The benchmark made is referred with the Left Hand Rule Algorithm which is a non-graph theory.

1.4 Limitation and Specification

This project mainly focuses on the comparison of time taken and distance travelled by an autonomous mobile robot which implements Dijkstra's Algorithm and Left Hand Rule Algorithm. The autonomous robots used are a 3pi Pololu mobile robot as the platform to study the efficiency of each algorithm.

Specifications of the robot are as follows:

Dimension	: 3.7" diameter
Processor	: ATmega168/328P
Motor Driver	: TB6612FNG
Motor Channel	: 2
User I/O Lines	: 2
Minimum operating Voltage	: 3V
Maximum operating Voltage	: 7V
Maximum PWM frequency	: 80 kHz
Reverse Voltage protection	: Yes

Specifications of the track are as follows:

Material	: Plywood
Type of line tracing	: Black line over white background
Width of black tape	: ¾”
Size of plywood	: 6” x 6”
Size of maze	: 5 x 5 of the plywood

1.5 Project report layout

This project report is organized as follows;

i) Chapter 1 briefs the overall background of the study. A quick glimpse of study touched in first sub-topic are the heart of the study such as problem statement, project objective, limitation and specification project report layout is present well through this chapter.

ii) Chapter 2 covers the literature review of previous case study based on development of algorithms to solve maze using various techniques. Some basics and general information regarding the rule of maze solving and the mobile robot used are also described in this chapter.

iii) Chapter 3 presents the methodology used to integrate both algorithms into the mobile robot itself. A basic explanation for each algorithm is presented.

v) Chapter 4 discusses on the results obtained based on the problem statements as mentioned in the first chapter. The results from several runs of the mobile robot on a particular maze are recorded and analyzed for each algorithm.

v) Chapter 5 mainly discusses the conclusion and recommendation for future study. References cited and supporting appendices are given at the end of this project report while the documentation CD is attached on the back cover of this project report for future reference.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

At the beginning of this project, a research regarding the newest algorithm used in maze solving robot, the physical hardware used, and any information that is relevant is gathered to lead this project to the correct path. The information gathered is mainly from IEEE's database and websites. This chapter discusses some of the projects and also the studies that have been done. The information provided served as a guideline to this project.

2.2 Maze Solving

A maze is a puzzled way which consists of different branch of passages where the aim of the solver is to reach the destination by finding the most efficient route within the shortest possible time. Artificial Intelligence plays a vital role in defining the best possible way of solving any maze effectively. Graph theory appears as an efficient tool while designing proficient maze solving techniques. Graph is a representation or collection of sets of nodes and edges and graph theory is the mathematical structure used to model pair wise relations between these nodes within the graph. By proper interpretation and mathematical modelling, it is possible to figure out the shortest distance between any of the two nodes. This concept is deployed in solving unknown maze consisting of multiple cells. Depending on the number of cells, maze dimension may be 8x8, 16x16 or 32x32. Each cell can be considered as a node which is isolated by walls or edges. This is nothing but the interpretation of graphs onto a maze. This analogy between graph and maze provides the necessary foundation in developing maze solving algorithms.

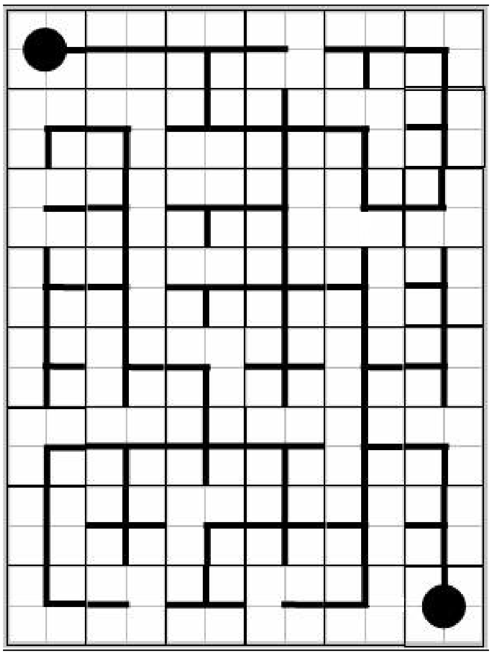


Figure 2.1: Example of Line Maze Solving

This report is based on solving a particular type of two dimensional maze using graph and non-graph theory algorithms where the solver is a self-governing robot. The maze may be made up of a 5x5 grid of cells. Each cell is 18cm square. By incorporating intelligent procedure with the existing graph theory algorithms, some Micromouse Algorithms have been developed. This research implemented the use of graph theory algorithms which is Djikstra’s Algorithm in maze solving. Analytical approach is taken to evaluate Djikstra’s Algorithm. The performance and outcome of these algorithms are also analysed and compared. With adequate reference and observation it is obvious that graph theory technique is the most efficient in solving various mazes than other techniques.

2.3 Related Works

Euler’s topological theory, Wall follower algorithm were the initial attempts to solve mazes. More algorithms like dead-end filling algorithm, shortest path algorithm also were used in solving the mazes in the traditional domain. Euler publishes a paper titled “The solution of a problem relating to the geometry of position” in 1736. The title itself indicates that Euler was aware that he was dealing with a different type of geometry where distance was not relevant. From Euler’s topological theory, in today’s notation would be stated as a *graph has a path traversing each edge exactly once if exactly two vertices have odd degree.* This was the first step in graph theory algorithm and was mainly used as guidance in most of other algorithms established.

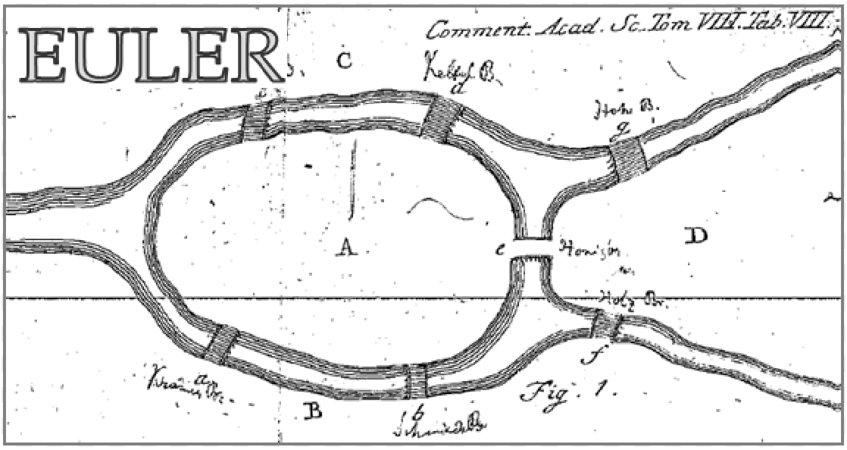


Figure 2.2: Euler’s Topological Theory

Pledge algorithm [6] solves the problem when the entry is in the interior of maze and exit is on the exterior of the maze. Pledge algorithm is an algorithm that instructs the robot on leaving an unknown maze see algorithm 1, which performs only two types of movements which are either the robot follows the wall of an obstacle and counts the turning angles, or the robot moves through the free space between the obstacles in a fixed direction. The latter task always starts at vertices of the obstacles when the angle-counter reaches a pre-defined value. As soon as the robot leaves the maze it receives a signal of success.

Algorithm 1: Pledge

```
REPEAT
    ω = 0
    REPEAT
        Move in direction ω in the free space
    UNTIL Robot hits an obstacle
    REPEAT
        Follow the wall in counter-clockwise direction
        Count the overall turning angle in ω
    UNTIL Angle Counter ω = 0
UNTIL Robot is outside the maze
```

Figure 2.3: Pledge Algorithm sequence

Tremaux's algorithm [7] guarantees the path for all cases by implementing the depth-first search. It solves the maze by using the backtracking till it finds the exit. It's similar to the recursive backtracker and will find a solution for all Mazes: As the robot runs down a passage, it will draw a line behind the robot to mark its path. When the robot hit a dead end, it will then turns around and go back the way it came. When the robot encounter a junction which it haven't visited before, it will pick a new passage at random. If it is walking down a new passage and encounter a junction which have visited before, treat it like a dead end and go back the way it came. If walking down a passage it have visited before (i.e. marked once) and it encounters a junction, take any new passage if one is available, otherwise take an old passage. All passages will either be empty, meaning they haven't visited it yet, marked once, meaning it have gone down it exactly once, or marked twice, meaning it has gone down it and were forced to backtrack in the opposite direction. When the robot finally reach the solution, paths marked exactly once will indicate a direct way back to the start. If the Maze has no solution, it will find itself back at the start with all passages marked twice.

Recently, maze solving has become a true multi-disciplinary topic, as many researchers tried to solve related problems with wide range of algorithms originating in different fields. Few of the attempts need special mentioning here as the work is much inspired by these multi-disciplinary publications. Some of the works are explained briefly in the next sentence.

Wyard-Scott [8] solved the maze problem in the electrical domain with varying potential differences driving the micro mouse to the destination. Yang and Meng [9] used neural network for generating the trajectory of robotic path.

Dijkstra's algorithm [10] is a legendary work on the lowest cost path finding using the graph theory. Floyd-Warshall [11] algorithm finds the shortest path between each pair of the nodes existing in the graph using dynamic programming.

Most of the recently used algorithms for path finding are graph searching algorithms. A* algorithm [12] is a heuristic based best first search algorithm with many variants. D* algorithm [13] is an incremental graph search algorithm with many variants and give better results. Rapidly Exploring Random Trees (RRT) algorithm [14] is used for solving the path problems with constraints.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter describes the methods that have been implemented in comparing maze solving algorithm. The main topics discussed in this chapter are the mechanical hardware, algorithms that have been used and maze configuration.

3.2 Project Overview

In order to analyse the maze solving algorithm, a line maze has been constructed and an autonomous mobile robot is utilized to solve the maze using 2 different algorithms which are Dijkstra's Algorithm and Left Hand Rule Algorithm.

A good and stable autonomous mobile robot is needed in order to solve the maze smoothly. The mobile robot should be able to:

a) Know the position and bearing in the maze.
b) Control the desire distance which needed to travel.
c) Make precise 45°, 90° and 180° turn.
d) Good chassis to keep the structure stable.
e) Navigate the maze intelligently.

3.3 Hardware

The hardware chosen for this project is 3pi Pololu Mobile Robot. The 3pi Pololu robot is a small, high performance, autonomous robot designed to excel in line following and line-maze solving competitions.

Figure 3.1: Pololu's 3pi Mobile Robot.

As for the maze, an adjustable 5 x 5 grid has been constructed to be used in this project. The term adjustable is used as the pattern of maze can be easily rearranged and changed compared to the fixed pattern mazes which are time consuming. Basically, the mobile robot will explore the map for the first time and search for the end of the map. During this run, the robot will save, analyse and distinguish the shortest path. After that, a second run will instruct the mobile robot to move towards the end of the map using the path that has been calculated. Usually, the first run or usually known as "run time" will have a larger time difference compared to the second run or also known as "maze time". The two algorithms will be programmed in Atmel's Atmega328P microcontroller.

The outputs of each experiment for the corresponding algorithm are displayed on the LED attached on the 3pi Pololu mobile robot. The outputs are in terms of time taken to solve the maze during maze time. Run times are not being considered in the optimisation study.

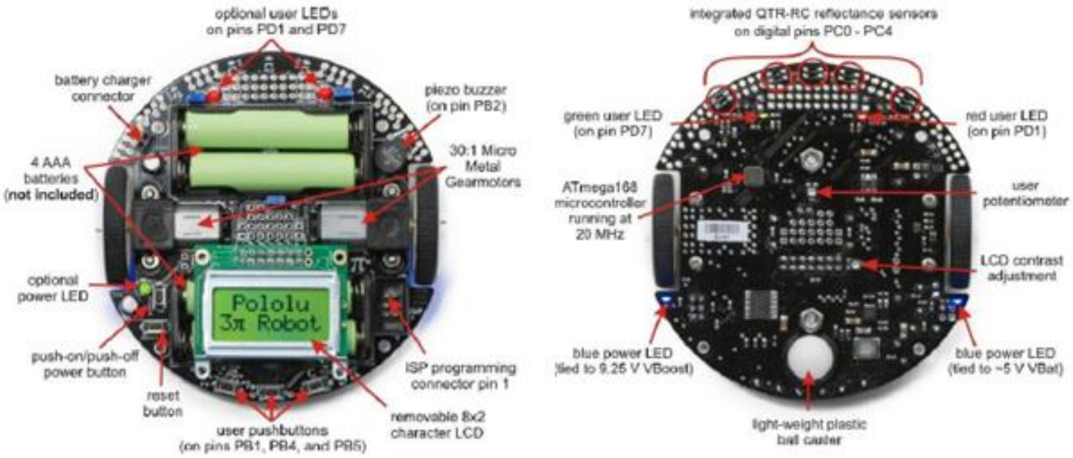


Figure 3.2: 3pi Pololu robot in detail.

In order to startup the 3pi Pololu Mobile Robot, 4 units of AAA batteries are required. It is recommended to use NiMH batteries that have rechargeable ability. This will ensure the project can run continuously without concern of the battery pack being drained during the testing phase.

Apart from that, an AVR ISP programmer with 6 pin connector is required. The 3pi has a standard 6 pin programming connector and thus suffice the need to download a program to the target device.

Lastly, a computer or a laptop is needed for developing algorithms and load it onto the 3pi robot.

3.4 Line Tracing

The robot used will follow a black line on a white background. It uses a sensor array to sense the line and a PID control to follow the line. With the help of a PID control, the robot does not follow the line by oscillating to the left and to the right like a conventional line followers but follows the line smoothly without wavering too much the left and right thus conserving battery power while at the same time following the line much faster.

This robot uses differential drives which are easier to build, controlled and are cheap. Irrespective of the steering mechanisms that are used, the most important thing to keep in mind is to place the line sensors as far as possible from the steering wheel. This will give the robot, more time to react. Keeping the distance between the sensors and the steering wheel big, will largely reduce the number of over-shootings by the robot.

Another important thing is to keep the center of mass as close to the ground as possible. This can be done by, placing all the heavy parts (batteries, motors) of the robot as close to the ground as possible. Keeping the center of mass low will decrease the robot's moment of inertia enabling it to slow down quickly and then accelerate faster in curves. A robot with a high center of mass takes a lot of time to slow down and a lot more time to accelerate afterwards owing to its high moment of inertia.

This robot uses a simple PID algorithm which controls the speed of each wheel individually to ensure that the robot is moving at the center of the line which is usually called as the target. The integration of wheel speed and the line tracing will be as follows:

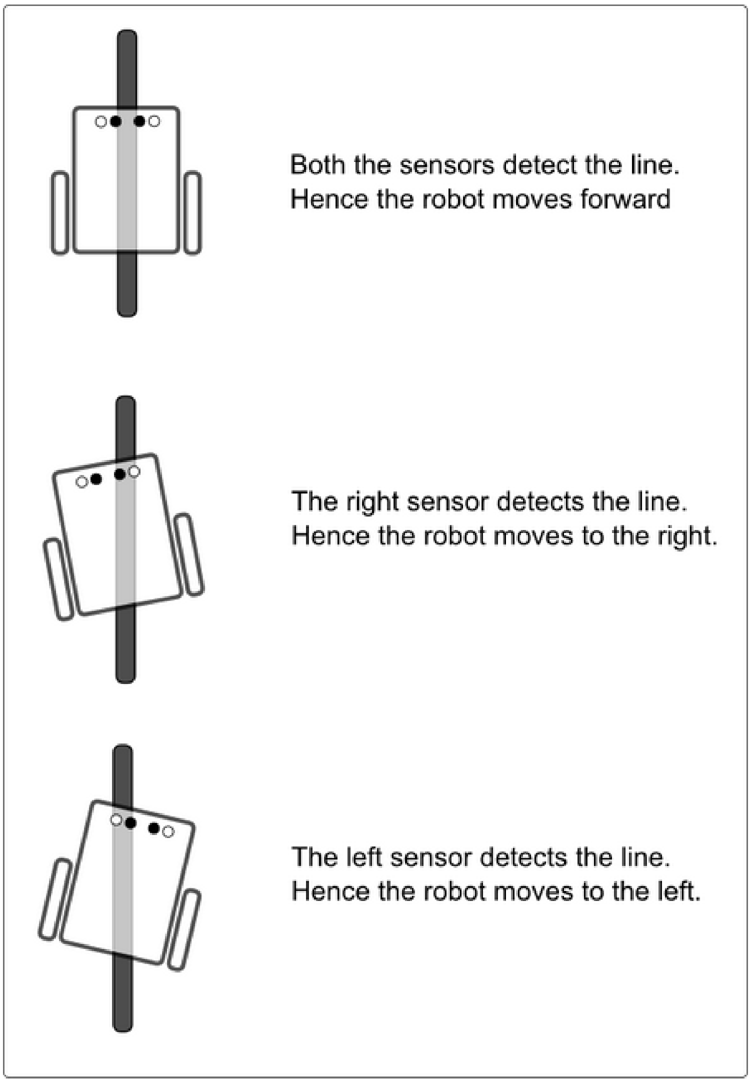
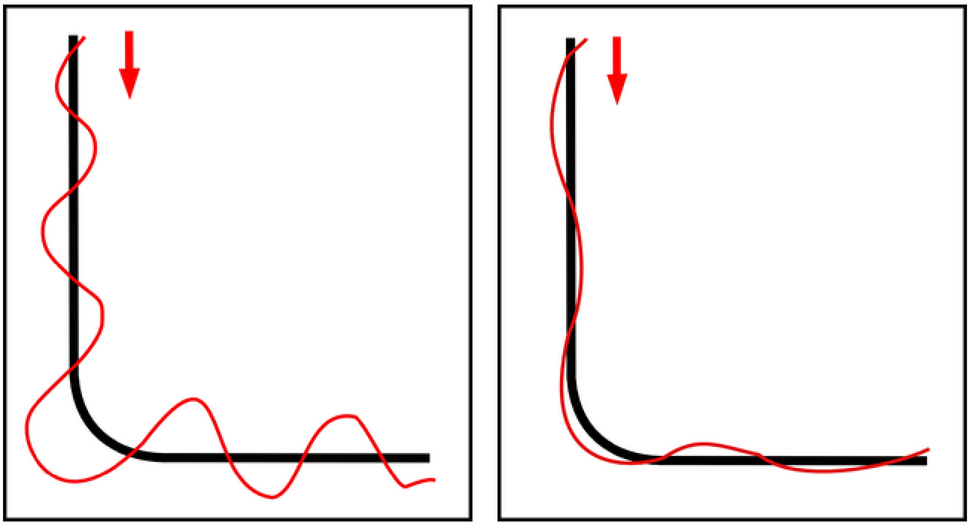


Figure 3.3: Basic algorithm for line tracing.

PID stands for Proportional Integral and Derivative. It is a popular control loop feedback control extensively used in industrial controls systems. But why would one need a PID controller for a line following robot, when there are many simpler algorithms already available for line following?

A conventional robot would follow a line as in figure 3.3(a). The red line shows the robot movement to follow the black line. It can be seen that the robot oscillates a lot about the line, wasting valuable time and battery power. For this algorithm, there is a maximum speed beyond which you cannot use or otherwise, the robot will overshoot the line.

Whereas in figure 3.3(b), the robot moves smoothly along the line keeping its centre always above the line. In straight lines, the robot gradually stabilizes go straight unlike a robot with the left-right algorithm. This enables the robot to follow the line faster and more efficiently.



(a) (b)

Figure 3.4: (a) Without PID algorithm. (b) With PID algorithm

Before explaining about the PID, there are several terms which needs to be clarified first. The terms are as follows:

Target – It is the position you want the line follower to always be (or try to be), that is, the centre of the robot.

Current Position – It is the current position of the robot with respect to the line.

Error - It is the difference between the current position and the target. It can be negative, positive or zero.

Proportional – It tells us how far the robot is from the line like – to the right, to the extreme right, to the left or a little to the left. Proportional is the fundamental term used to calculate the other two.

Integral – It gives the accumulated error over time. It tells us if the robot has been on the line in the last few moments or not.

Derivative – It is the rate at which the robot oscillates to the left and right about the line.

K_p, K_i and K_d are the constants used to vary the effect of Proportional, Integral and Derivative terms respectively.

The controller calculates the current position on the black line. Then calculate the error based on the current position with the target position. It will then command the motors to take a hard turn, if the error is high or a small turn if the error is low. Basically, the magnitude of the turn taken will be proportional to the error. This is a result of the Proportional control.

Even after this, if the error does not decrease or decreases slowly, the controller will then increase the magnitude of the turn further and further over time till the robot centres over the line. This is a result of the Integral control. In the process of entering over the line the robot may overshoot the target position and move to the other side of the line where the above process is followed again. Thus the robot may keep oscillating about the line in order to centre over the line. To reduce the oscillating effect over time the Derivative control is used.

3.4.1 Implementing PID control for a line following robot

The first requirement in implementing PID for a line follower is calculating the error of the robot. To calculate the error, the robot needs to know the current position of the robot with respect to the line. There are a number of ways of knowing this.

A simple approach would be to place two IR sensors on either side of the line. The IR sensors should be tuned to give an output voltage that is promotional to the distance between the line and the sensor. The output can then be connected to the ADC pin of a microcontroller and the error can be calculated.

Though this method may seem simple and easy to implement it has a few drawbacks. Firstly, a robot using this method will have problems following a line whose width is varying. Secondly, the output of these sensors will be highly susceptible to interference from external light. And lastly, for this method to work, user will have to ensure that the track is completely flat. Any change in the distance between the sensor and the surface will affect the sensor readings.

A better approach would be to use the traditional sensor array. Using an array of sensors, it is easy to calculate the error by knowing which sensor is on the line. Consider the sensor have an array of 10 each placed 1cm apart. When the 7^{th} sensor from the left detects the line it can be calculated that the centre of the robot is 2 cm to the right of the line. Using a sensor array the processor can calculate the error faster as there is no ADC required to be done and thus the sampling rate can be increased. The robot can also be used as a grid solver.

When building a sensor array, there are a few things to keep in mind. Firstly, the more sensors in an array, the better will be the performance of the robot, because more sensors translates to more range of error. For line following anything between 6 to 10 sensors will be sufficient. The distance between each adjacent sensor determines the resolution of the readings. It is always best to place the sensors as close to each other as possible because the resolution will increase as the sensors are closer.

The next important thing in implementing PID after the sensors is the writing the code itself. The algorithm for a PID control for line followers is described as below:

```
Error    = target_pos – current_pos      //calculate error
P = Error * Kp                           //error times proportional constant gives P
I = I + Error                            //integral stores the accumulated error
I = I * Ki                               //calculates the integral value
D = Error – Previous_error               //stores change in error to derivate
Correction = P + I + D
```

The next step is to add this correction term to the left and right motor speed.

3.5 Algorithm

In this section, the famous Graph Theory based Algorithm used in this project are explained which are Dijkstra’s Algorithm. A Non-Graph Theory based algorithm which is the Left Hand Rule are used as a benchmark. The steps for each algorithm to solve the maze are discussed.

3.5.1 Dijkstra's Algorithm

One of the main reasons for the popularity of Dijkstra's Algorithm is that it is one of the most important and useful algorithms available for generating (exact) optimal solutions to a large class of shortest path problems.

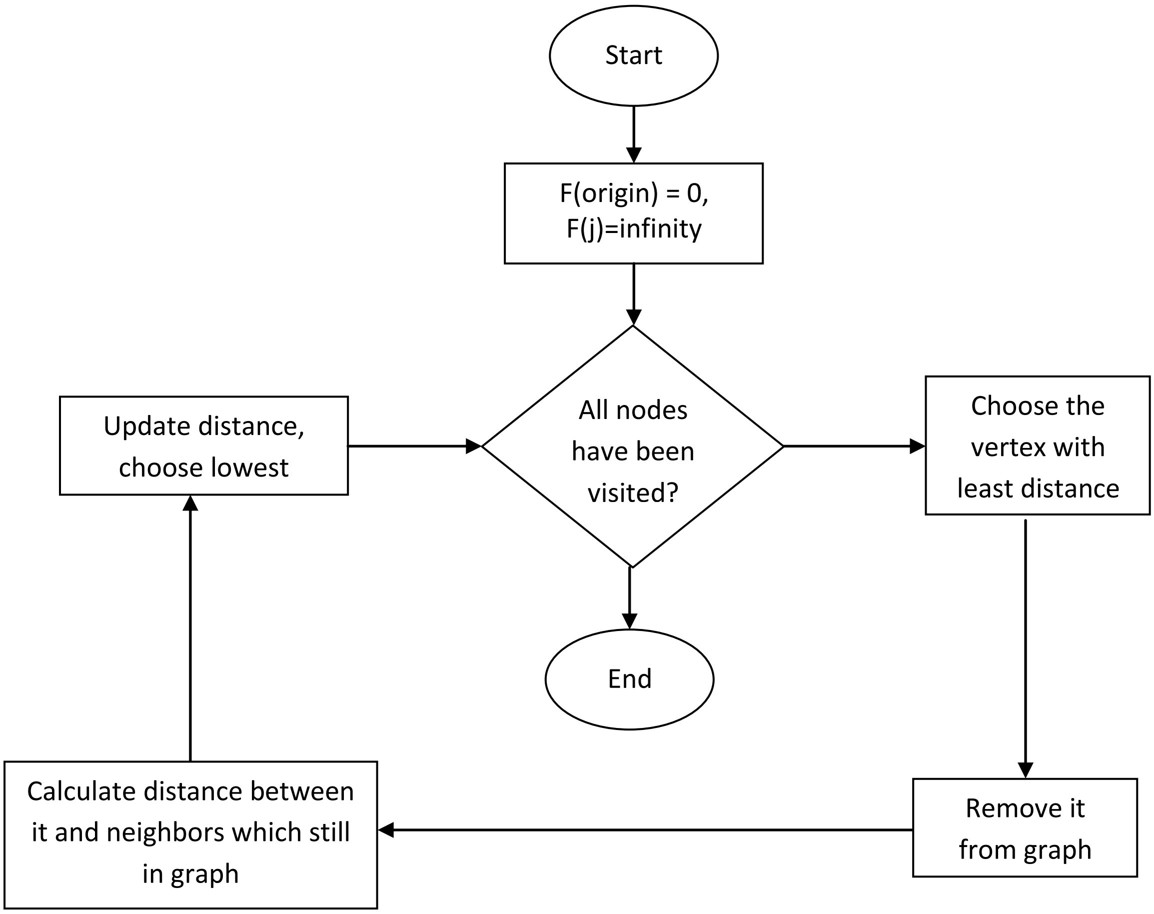


Figure 3.5: Flowchart of Dijkstra's Algorithm.

Basically, Dijkstra's Algorithm uses a technique known as BFS (Breadth First Search) to solve the maze [15]. It works in two phases which are the "filling phase", the cells are marked and the "retrace phase", which uses the concept of back tracking. The algorithm are sequenced as in the numbered below:

1) Label the first vertex with label 0 and order label 1.
2) Assign temporary labels to all the vertices that can be reached directly from the start.
3) Select the vertex with the smallest temporary label and make its label permanent. Add the correct order label.
4) Put temporary labels on each vertex that can be reached directly from the vertex you have just made. The temporary label must be equal to the sum of the permanent label and the direct distance from it. If there is an existing temporary label at a vertex, it should be replaced only if the new sum is smaller.
5) Select the vertex with the smallest temporary label and make its label permanent. Add the correct order label.
6) Repeat until the finishing vertex has a permanent label.
7) To find the shortest path(s), trace back from the end vertex to the start vertex. Write the route forwards and state the length.

Given below is a table which are used in Dijkstra's calculations. It is only intended to help visualize the reader on how the calculation works. The algorithm is best explained by using an example which will be discussed in detail.

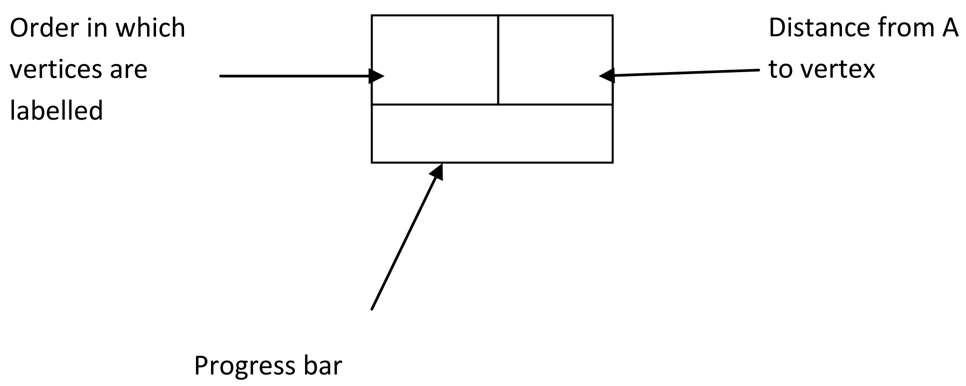


Figure 3.6: Table used in Dijkstra's calculations.

Given below is an example of 5 vertices labelled as A, B, C, D and E. It can be observed that the numbers in between the vertices represents the distance from each respective vertex. From the first rule of Dijkstra's Algorithm, it states to label the first vertex with label 0 and order label 1. Therefore, at vertex A, the upper left of the table is the label and since it is the starting point, it is labelled as 1 and as for the upper right table, it is the distance of vertex from the starting point. As it is at the beginning of the vertex, the distance is 0.

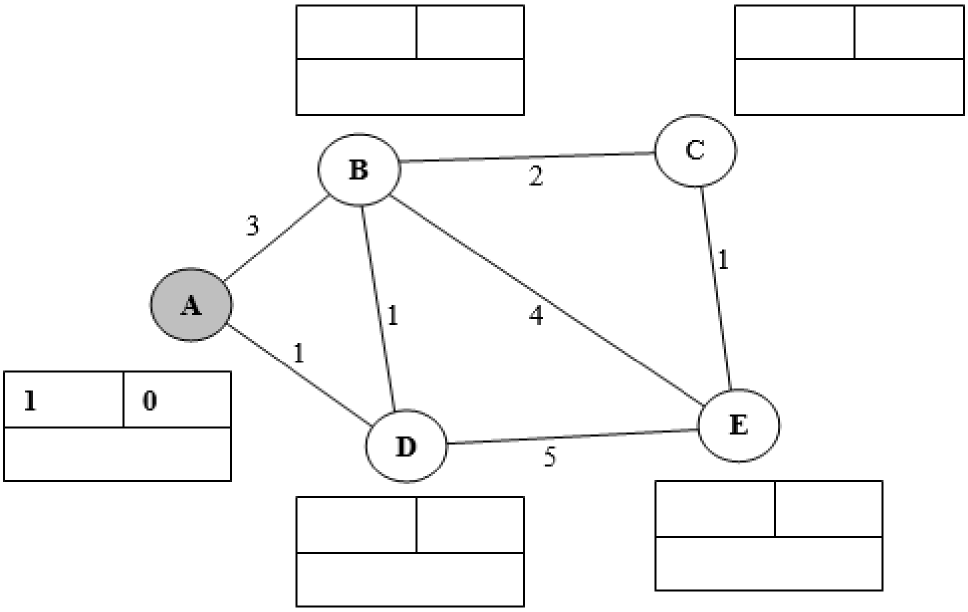


Figure 3.7: Step 1 of Dijkstra's Algorithm.

After that, the second rule of Dijkstra's states to assign temporary labels to all the vertices that can be reached directly from the start. This means that from vertex A, vertex B and vertex D can be reached. Due to this, the bottom of the table for each vertex will be written with the distance from vertex A as in figure 3.7 below. Next is to implement the third rule of Dijkstra's Algorithm which is to select the vertex with the smallest temporary label and make its label permanent. Add the correct order label. The temporary distance for vertex B and D are compared and vertex D has the smallest temporary label. Therefore, it is labelled as 2 and the permanent distance is 1.

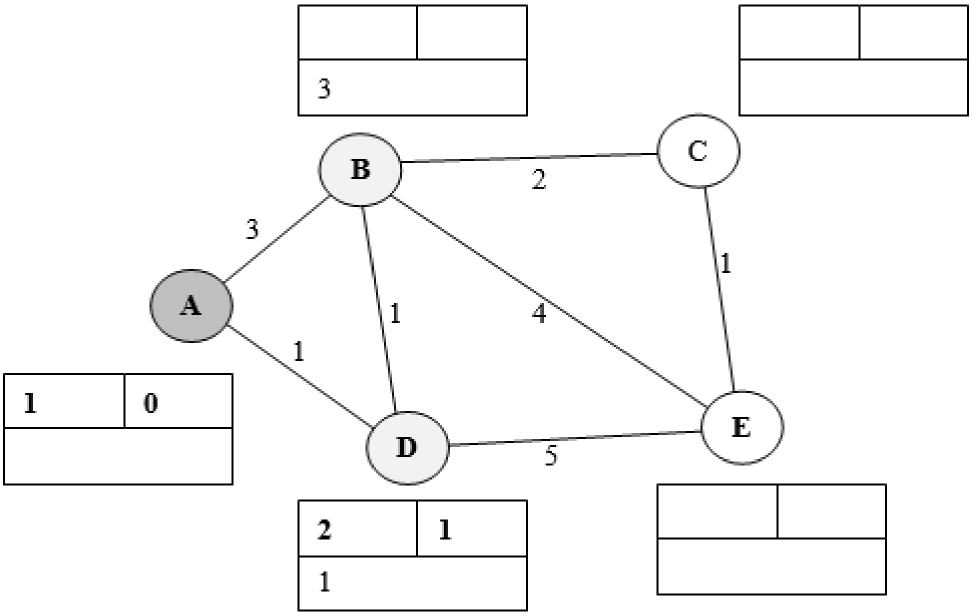


Figure 3.8: Step 2 and 3 of Dijkstra's Algorithm.

The 2nd and 3rd rule for Dijkstra's applies on the next step. The vertex D has been made permanent. Therefore, the neighbor vertexes that can be reached are B and E. The fourth Dijkstra's rule applies which are to put temporary labels on each vertex that can be reached directly from the vertex have just made. The temporary label must be equal to the sum of the permanent label and the direct distance from it. If there is an existing temporary label at a vertex, it should be replaced only if the new sum is smaller.

REFERENCES

[1] Lauren Artress. 2006. *The Sacred Path Companion: A Guide to Walking the Labyrinth to Heal and Transform*. Page 92.

[2] Paul Muljadi. 2008. *Introduction to Graphs*. Page 1.

[3] Alexanderson, G. L. 2006. *Euler and Königsberg's Bridges: A historical view*. **Bulletin of the American Mathematical Society.**

[4] MAZEMASTERS. 2007. *Maze to Tree* [online]. Available: http://www.youtube.com/watch?v=k1tSK5V1pds [Accessed].

[5] ADIL M.J.Sadik, M. A. Dhali, Hasib M. A. B. Farid, Tafhim U Rashid, A. Syeed 2010. *A comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory*. 2010 International Conference on Artificial Intelligence and Computational Intelligence.

[6] H. Abelson and A. diSessa, "Turtle Geometry: The computer as a Medium for Exploring Mathematics," MIT Press, 1986.

[7] Steve Alpern and Shumuel Gal, "*The Theory of Search Games and Rendezvous*," SpringerLink, 2006. ISBN 978-0-7923-7468-8.

[8] L.Wyard-Scott and Q.H.M.Meng, "*A potential maze solving algorithm for a micromouse robot,*" Proc. IEEE Pacific Rim Conf. Commun., Compu., Signal Process., Victoria, BC, Canada, May 1995, pp. 614-618.

[9] Yang, S.X and Meng. M, "*Neural network approaches to dynamic collision free trajectory generation,*" Systems, Man, and Cybernatics, Part B: Cybernatics, IEEE Transaction on, vol 31, Issue 3, June 2001, pp 302-318.

[10] Sniedovich, M., "Dijkstra's algorithm revisited: the dynamic programming connection," Journal of Control and Cybernatics, vol 35, Issue 3, 2006, pp 599-620.

[11] Floyd and Robert W., “Algorithm 97: Shortest Path,” Communications of the ACM, vol 5, Issue 6, 1962, pp 345.

[12] Xiao Cui and Hao Shi, “A* based path finding in Modern Computer Games,” International Journal of Computer Science and Network Security, vol 11, issue 1, 2011, pp 125-130.

[13] Ferguson D. and Stenz A., “Multi-resolution field D*,” Proceedings of the International Conference on Intelligent Autonomous Systems (IAS), 2006.

[14] Ferguson D. and Stenz A., “Replanning with RRTs,” IEEE International Conference on Robotics and Automation, 2006.

[15] Micheal, Gims, S.L.D.B. 1999. *Micromouse-Microprocessor Controlled Vehicle*, Bachelor of Engineering, University of East London.

[16] Bhawna Gupta, Smriti Sehgal. 2014. *Survey on Techniques used in Autonomous Maze Solving Robot*. 2014 5th International Conference – Confluence the Next Generation Information Technology Summit (Confluence).