Sukritee Sharma - MM-211659

Meshmerize - Maze Solver Robot

Abstract

Problem Statement

Teams have to build an autonomous robot which can follow a white line and keep track of directions while going through the maze. The bot has to analyze the path in the Dry Run and use this information in the Actual Run to go through the maze from the starting point to the ending point in minimum possible time.

Objectives

- Developing a robot to find the shortest route in a maze from the starting to the end point.
- Programming the microcontroller to make decisions on when to turn.
- Programming the microcontroller to choose which direction to turn if more than one option is available.
- Developing an algorithm for finding the shortest route through the maze after the *dry run*.
- Incorporating the shortest route found for the *actual run*.
- Programming the microcontroller for stopping when the end point is reached.

Introduction

Maze Solving Robot

The purpose of this autonomous robot is to find the shortest route possible in a maze while following white lines. It is designed to find its path without assistance of any kind.

Applications

The maze solver robot has many applications, including self-navigation and automating everyday tasks. The algorithms utilized in creating this bot can be incorporated into self-driving cars and automation industry for obstacle avoidance and increasing efficiency in repetitive tasks.

Components

S.No.	Components	Number of Units	Description / Utility / Purpose
1.	Chassis	1	Assembling and carrying all components
2.	Rear wheel	1	Support in the rear
3.	Front wheels	2	Direction controlling wheels
4.	L298n Motor driver	1	Controlling motor speed and direction
5.	DC motors	2	Rotating wheels
6.	Arduino Uno	1	Microcontroller
7.	IR sensors	3	For detecting white lines contrasting against black background
8.	11.1V Battery	1	Lipo battery for larger current supply

Setup

- The robot has two driving wheels in the front controlled by a motor driver connected to the microcontroller (Arduino Uno).
- There are three IR sensors (left, centre and right) attached to the front of the chassis, feeding back HIGH/LOW values to the Arduino.
- The whole setup is powered by a battery kept on top of the chassis.

Algorithm

Following along a straight line

The robot is programmed in such a manner that it moves along the white line in forward direction as long as the IR sensor in the centre keeps feeding back a HIGH signal to the Arduino.

Note: *A* HIGH *signal from the IR sensor implies that white color is detected.*

Turning at an intersection

When only one direction is available

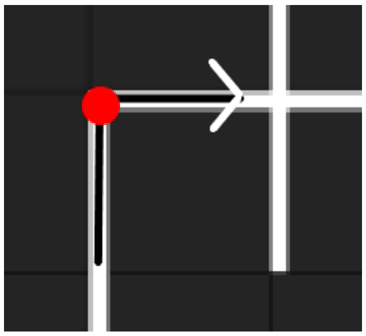
Fig. 1

For example, if we can only turn right as shown in the figure.

At the intersection (denoted by the red dot), when the IR sensor on the right returns a HIGH value, the robot starts turning right.

The right motor stops moving while the left motor assists in turning.

The turning stops once the centre IR sensor once again returns a HIGH signal after detecting the white line. The motors start moving to steer the robot in the forward direction.

The path is stored as *Straight, Right (SR)* for use while finding the shortest path in the Actual Run.

When more than one directions are available

When two directions for turning are available for the robot at an intersection, as shown in the figure, the **LSRB algorithm** is followed.

LSRB Algorithm

According to the LSRB algorithm, when more than one option for turning is available, the priority for which direction the robot will turn is given in the order: L>S>R>B.

Here, L stands for Left, S for straight, R for right and B for back.

It means:

- If given an option for turning left, the robot will definitely turn left even when straight or right paths are available.
- If there is no option for turning left, the robot will proceed in the forward direction without turning at the intersection.
- If there is only one turning option available, the robot will proceed according to the steps described above in the section of "*When only one direction is available"*
- If the robot reaches a dead end, then it will have no option except to turn back,

Example

Fig. 2 shows an intersection (marked with red) with two options available: either to head straight ahead, or turn right. According to the LSRB algorithm, the robot will choose to move straight ahead as S

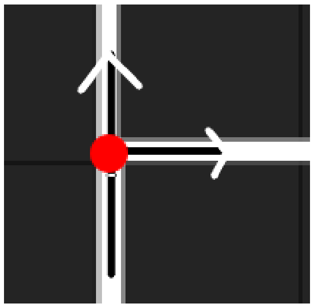
Fig. 2

precedes R. There will be no change in directions so both the motors will keep running at the same speed and the robot will follow the white line with the centre IR sensor returning a HIGH value.

The path is stored as *Straight(S)* for use while finding the shortest path in the Actual Run.

Turning at a Dead End

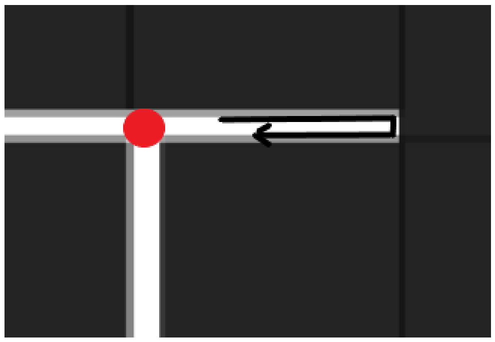
Fig. 3

When a dead end is reached, as shown in *Fig. 3*, there is no option left except to turn back.

For this, the robot will do a clockwise 180° turn with the right motor coming to a complete stop and the left motor at HIGH. Both the motors resume at equal speeds when the centre IR sensor once again detects a white line.

The path is stored as *Back(B)* for use while finding the shortest path in the Actual Run.

Actual Run Algorithm

Along the course of the dry run, the robot will encounter many intersections on which decisions on turning will have to be made. All the turns taken at any intersection will be registered in the microcontroller as:

Left, Back, Straight, Left, Right, Left… (this is only an example path followed by the robot)
Denoted in abbreviated form as: *LBSLRL…*

Now, in the actual run, the robot has to find the *shortest* or *most direct* path from start to end. Therefore, it cannot follow this registered path stored in the Dry Run. So, the microcontroller is programmed to find the 'equivalent path' which the robot travels.

For example, if the robot has turned *Left, Back, Right* in the Dry Run then the robot will just turn *Back* instead of taking all these extra turns. Because *Back* is the 'equivalent path' of *LBR.*

Similarly, all the equivalent paths for other set of turns are listed below:

1. LBR = B
2. LBS = R
3. LBL = S
4. RBL = B
5. RBR = S
6. SBL = R
7. SBS = B

Therefore, the Dry Run *LBSLRL* will become *RLRL* in the Actual Run.