# Maze Robot Assignment CS26020

Joel Adams [joa38]
Student ID: 180005309

20th April, 2020

## Contents



## 1 Introduction

This document describes my implementation of the CS26020 maze robot assignment. This code was written using Finite State Machines, reactive and deliberative systems in mind.

# 2 How much of the controller was I able to design?

Due to the lockdown and the rush to leave Aber earlier than expected, I didn't have enough time to finalise my code. If I had been given another day to work on it then it would be completely working but as it stands I have completed/debugged scenarios 1, 2, 3 and about half of 4. This is because my robot cannot display the maze on the internal LCD as I had some issues figuring out the logic behind it.

# 3 Controller design

## 3.1 Overview

**Setup code during initialisation**
The program begins by asking the user to connect a Bluetooth device, followed by setting the default compass position to North as specified in the assignment. This is followed by a for loop that iterates over the 2D array of cells (which is used to represent the maze) and sets them all to unvisited by default. It then prints an empty grid on the robots LCD.

**Looping code until program is completed**
First the motor speed of the robot is set to a relatively slow speed which is then followed by an if statement which checks if a line is detected. If it is detected then it moves the robot forwards by 10cm and stops. It then increases the line counter followed by reading the node data, printing the node data via Bluetooth[1] and detecting the walls so that it can determine which direction to turn. After the if statement a function will always be called to check that the robot isn't too close to the left, right or front wall and adjust it accordingly. Then one final if statement is called to check if all the nodes in the 2D array have been visited. If this returns true then the speed is set to 0, the final table is printed to the LCD (please note that this function doesn't work correctly) and then a delay for 10 seconds to allow the user to read the final table before the robot restarts.

## 3.2 Behaviours and control strategies

### 3.2.1 Navigating and modelling the maze

For information on how the maze is modelled please see section 4.1 Structure

The behaviour **markAllNodesUnivisited()** sets the visited flag of each cell in the maze to false and the behaviour **checkAllNodesVisited()** checks if all the cells in the maze have been visited and returns true if correct resulting in the program finishing.

When the robot crosses a line the behaviour **currentPosition()** is called which uses the compass to update the coordinates of the robot in the maze.

### 3.2.2 Detecting and reading cells/nodes

For the maze to be explored properly the robot first needs to detect when it has moved into a cell. This is done by using the behaviour **detectLine()** which will check when a line is detected using the line sensor and the robot will update its position in the maze using the behaviour **currentPostion()**. This uses the compasses orientation to update the x and y coordinates so that the correct cell in the 2D array is accessed. Finally the robot will move forward by 100mm so that it moves into the centre of the next cell.

The robot can then read the cell using the behaviour **readNode()** which reads all four main IR sensors (front, left, right, rear) and updates a struct which contains information about that cell in a 2D array followed by checking

if the cell is a nesting area using the behaviour **checkNestingArea()**. This updates the cell's struct nesting area flag to true, followed by turning all the LED's on for 1 second and then switching them off to indicate that the cell has been visited.

### 3.2.3 Driving and realignment

The robot uses the behaviour **detectWall()** which will read the front, left and right IR sensors. It will always prioritise turning left (by 90°) when possible, otherwise it will go forwards or turn right (by 90°) if that is not possible. If none of these options are available then the robot will turn 180°. Once the robot has turned in a specific direction the compass will also be updated.

The robot can realign itself using the behaviour **avoidWalls()** which reads the front_left, front_right and front IR sensors. Using these sensor readings it detects whether the robot is too close to the left, right or front wall and either turns in the opposite direction by 5° or reverses by 10mm respectively.

### 3.2.4 Displaying and debugging the maze

For debugging the maze and displaying data about the current cell I used a behaviour called **printNode()** which sends data about the robots position in the maze, lines counted, compass orientation and cell information via Bluetooth to my laptop. The image below shows an example of the output.

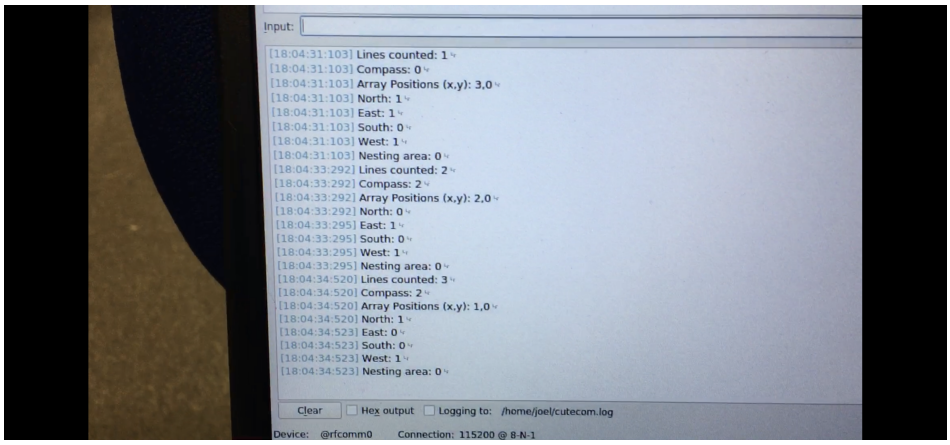


Figure 1: Bluetooth data displayed

The maze is displayed using the behaviour **printStartTable()** which prints out a 5x5 grid on the LCD display. The behaviour **printFinalTable()** displays the final table to the LCD once the maze is completed but this behaviour doesn't work properly or produce the correct output. I have left it in my code so you can see how I tried to model it.

## 3.3 How the behaviours are combined

The behaviour **readNode()** is combined with the compass behaviour so that the walls of the cell are correctly identified and in the right orientation. The behaviour **detectWall()** updates the compass depending upon which direction the robot turns. The behaviour **detectLine()** calls the behaviour **currentPosition()** to change the x and y coordinates.

# 4 Internal model of the maze

## 4.1 Structure

I structured the maze using a 5x5 2D array as it is basically just a simple grid that can be accessed using an x and y grid coordinate system. These coordinates are represented by 2 static global integers starting at x(horizontal) = 2 and y(vertical) = 0.

Each cell is represented by a struct called **mazeNode** which has 4 integers (North, East, South and West) to store the wall data (0 = no wall, 1 = wall), a boolean flag for if the cell has been visited and a second boolean flag for if the cell is a nesting area. A compass is also implemented using an enum that contains the variables North (0), East (1), South (2) and West (3). There are also two further static integers for keeping count of the number of lines crossed and the number of nesting areas discovered.

## 4.2 How the controller is updated by behaviours

When the robot crosses a line the behaviour **currentPosition()** is called which uses the compass to update the coordinates of itself in the maze. The compass on the other hand is updated when the robot turns.

# 5 Sensors

## 5.1 Summary of obstacle sensors in sensors practical

Below is a chart of IR sensor data I collected every 5mm using a total of 5 readings to calculate the average of each point. In blue are the readings collected when the IR sensor reflects off a white surface and in orange are the readings collected when the IR sensor reflects off a dark surface. Before conducting the experiment I hypothesised that a white surface would return a higher reading and a dark surface would return a lower reading as IR sensors gauge distance based on the intensity of the reflectance.

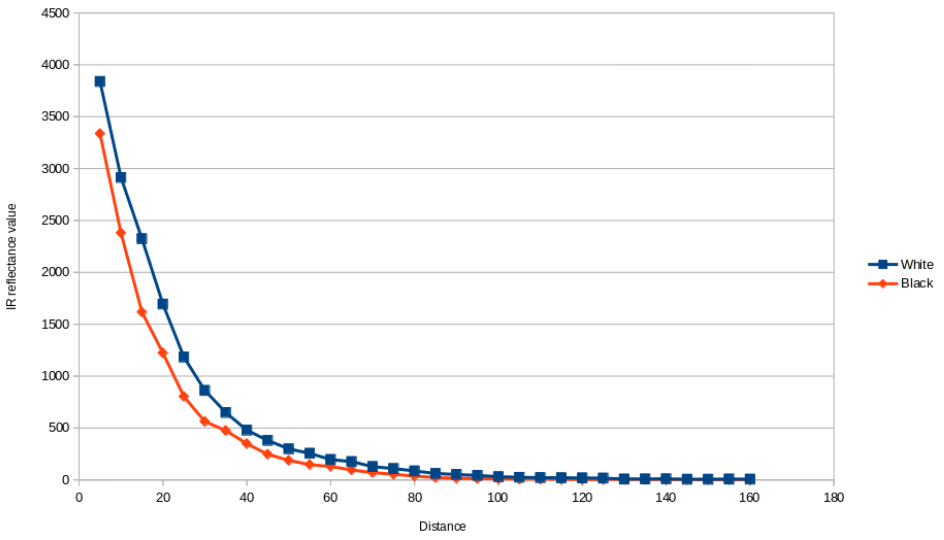


Figure 2: Rear IR sensor readings

Once I finished plotting the experiment I concluded that my hypothesis was correct, however I didn't consider the inaccuracy of the sensors after reaching 120mm. At this point the sensors started returning readings that would sometimes be higher than prior readings resulting in the sensors being considered useless after a distance of 120mm for both black and white surfaces.

### 5.2 How this information was used in the controller

I concluded that if any IR sensor read a value below 100 IR reflectance (about 55-60mm) then it would indicate that a wall is present. For the realignment behaviour, if the IR reflectance value was higher than 200 then the robot would readjust itself by 5°.

I also did a quick experiment with the light sensor and concluded that any readings lower than 450 would be considered appropriate to measure a nesting area.

## 6 Encountered problems

During the course of programming this robot I encountered 2 major problems. The first of which being that I found it difficult to get my head around the driving logic. I wanted to implement a simple behaviour that would cause the robot to always prioritise going left when it could so that the entirety of the maze would always get explored. My original implementation used to make the robot constantly attempt to turn the robot where possible, however this meant that when a left turn was found the robot would turn towards it and then realise that there was also another free turning on its left causing it to drive back in the direction it came from. I managed to solve this issue by only calling the turn behaviour one time once I had passed into a new square.

The second major issue that I had was I was having problems displaying the final data onto the LCD on the robot. I believe that my logic is correct however I cannot get it to work properly. Annoyingly I ran out of time to work on this behaviour as due to COVID-19 I had to return home to Bristol.

## 7 How could I improve or extend my controller

I believe that my controller is mostly quite good however I wish that I could've fixed the glaring issue described in the above section. The controller could do with some more debugging as the robot would sometimes stop believing that it has explored the entirety of the maze before having reached every cell.