Mansoura University
Faculty of Engineering

Department of Computer
Engineering and Systems

Artificial Intelligence
CSE 3315

MAZE SOLVING ROBOT COMPETITION
2019-2018

Team:
هو قالك فين؟

Supervisor:
Dr. Mahmoud Saafan

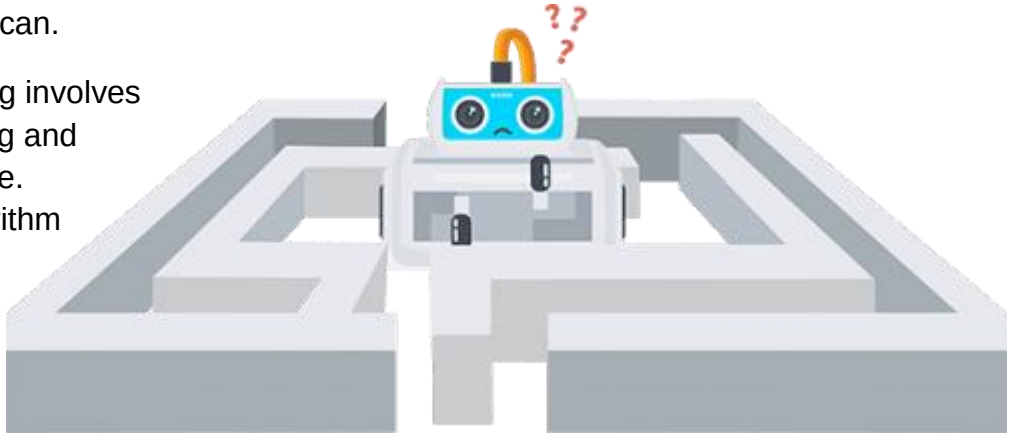
Members:
Khaled Abdulkader Madkour
Abdulrahman Issam AbouOuf
Mohamed Nasser Elfrash
Mohamed Hussin Salama
Omar Sameh Rehan
Ahmad Hamed Akef
Omar Raafat Elezaby
Abdulla Ashraf Abdelrahman

Abstract

ABSTRACT

The project deals with the development maze robot using simple circuits. A MSR (maze solving robot) is a clever little gadget with a silicon brain that finds its way through an arbitrary maze. It competes against other members of its species, racing as fast as it can.

Maze-solving involves Control Engineering and Artificial Intelligence. Using a good algorithm can achieve the high efficiency of finding the shortest path.



This exercise is a study about the ability to equip a small mobile robot with the ability to learn how to navigate in unknown environment based on its own decisions.

This report discusses the steps in detail to build a maze solving robot from the software and hardware perspective using (BFS DFS Algorithm) and (Left Wall Follower).

TABLE OF CONTENTS

LIST OF FIGURES

TABLES

1 CHAPTER ONE INTRODUCTION

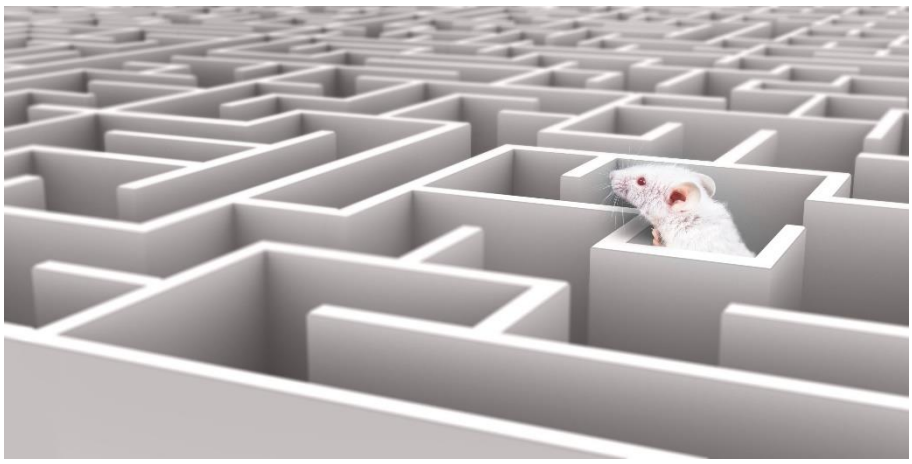
This Chapter aims to give a brief discussion about the project, define the problem statement and state the objectives, general information about the maze solving robot and a walkthrough for the report.

Autonomous navigation is an important feature that allows a mobile robot to independently move from a point to another without an intervention from a human operator. Autonomous navigation within an unknown area requires the robot to explore, localize and map its surrounding. By solving a maze, the pertaining algorithms and behavior of the robot can be studied and improved upon. This paper describes an implementation of a maze-solving robot designed to solve a maze using BFS and DFS from python and allowing the Arduino to use the shortest path to aim the robot through the end. Also, the report discusses using the “left wall follower” if the map is unknown and using PID control theory to control the movement of the robot.

1.1 What is a maze from (graph) point of view?

The maze-solving task is similar to the ones in the Micro Mouse competition where robots compete on solving a maze in the least time possible and using the most efficient way.

A maze is a network of paths (Graph), typically from an entrance to exit. The concept of Maze approximately thousand years old. Which was invented in Egypt. From then, many mathematicians made various algorithms to solve the maze.



Now, maze solving problem is an important field of robotics. It is based on one of the most important areas of robotics, which is “**Decision Making Algorithms**”. Cause, this robot will be placed in unknown place, and it requires to have a good decision making capability. There are many types of maze solving robot using various type of algorithms.

1.2 Problem Statement

Is to create a mechanical robot using Arduino, sensors and actuators and find the proper algorithm to solve the maze.

1.3 A brief walkthrough

To create a maze solving robot you need:

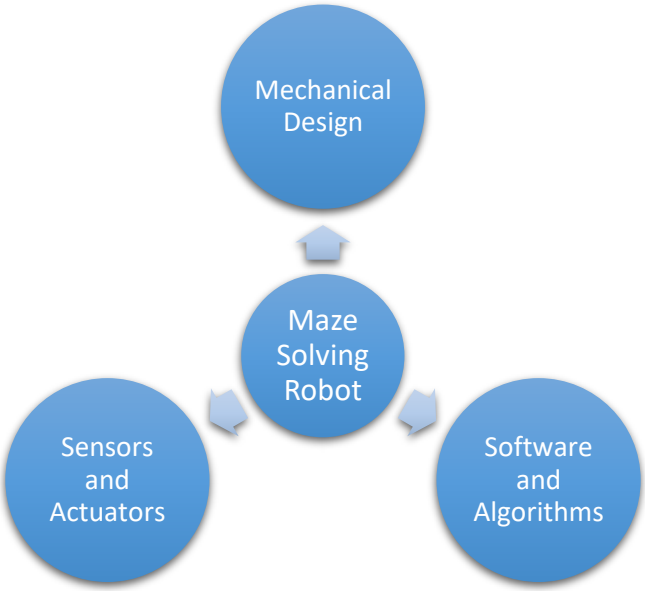


Figure 1-1 - Creating the robot process

1.3.1 Mechanical Design

Is to create a body that fits the requirements to walk through the maze. And to carry the Sensors, actuators, etc.…

1.3.2 Sensors and Actuators

Using Ultrasonic to read the walls and connecting them to the Arduino and depending on these readings the motors will decide which way to go.

1.3.3 Software and Algorithms

It is the brains of the robot, using python and BFS algorithm to send the shortest path and send it to Arduino to control the motors. Also using the left wall follower algorithm if the map is unknown.

In the next chapters, we will discuss each point of the process in detail.

2 CHAPTER TWO MECHANICAL DESIGN

This Chapter explains how to build a body of a robot using (SOLIDWORKS) and choosing one that fit, also showing the design of the robot.

Since each block of the maze is 20*20cm and a height of 15cm, it is required to design a robot body that is able to walk through this maze

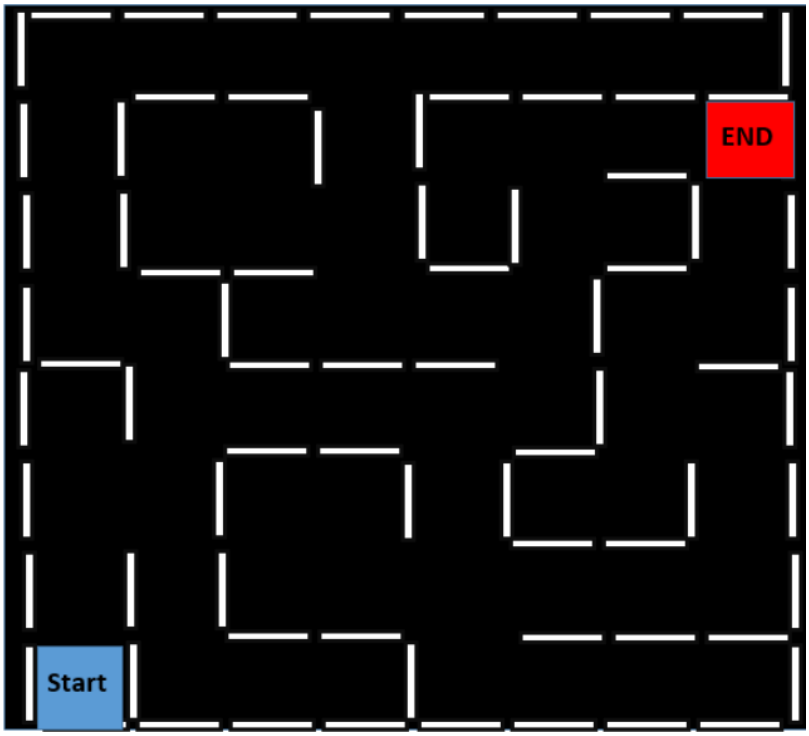


Figure 2-1 - Maze Description

2.1 Schematic design

Designing a robot body 13*13 cm and a height of almost 13cm so it can rotate around its axis easily in the maze.

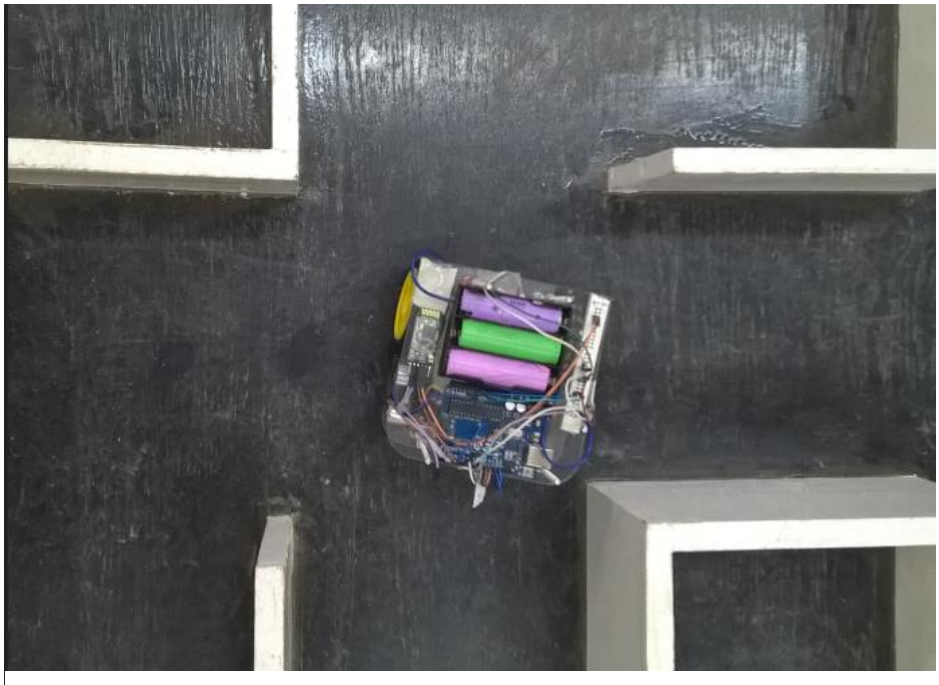


Figure 2-2 Our robot in the maze

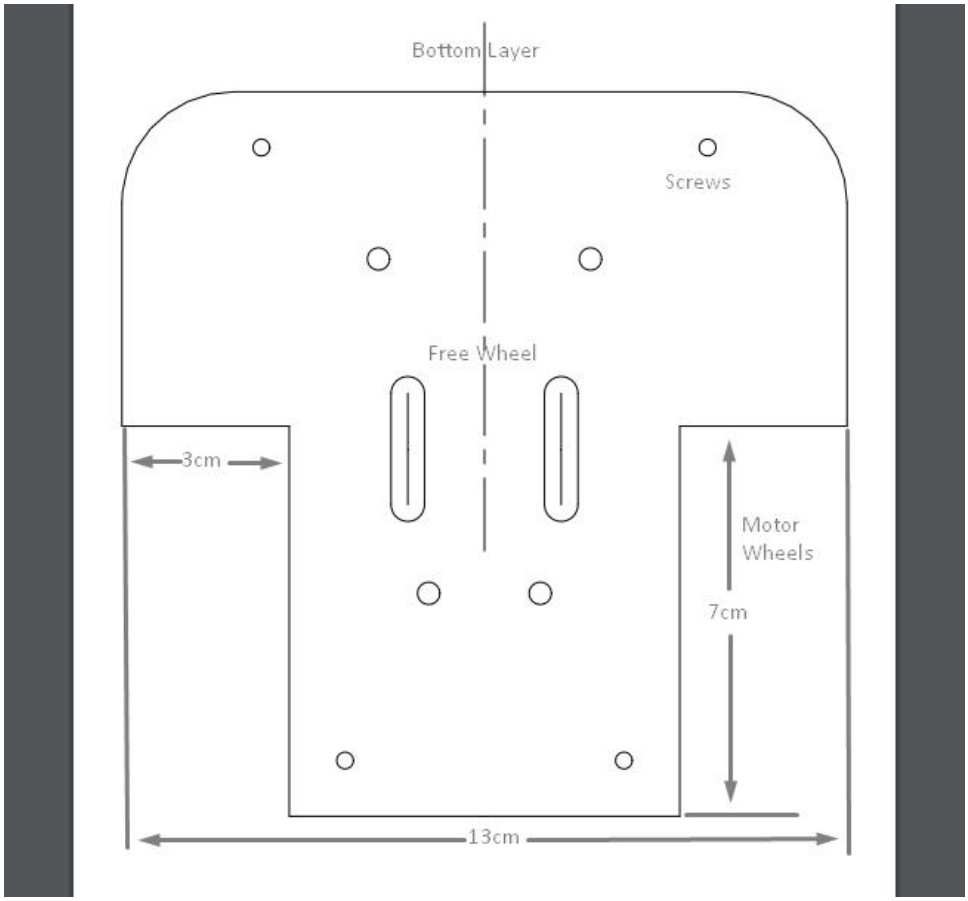


Figure 2-3 - Schematic Bottom layer

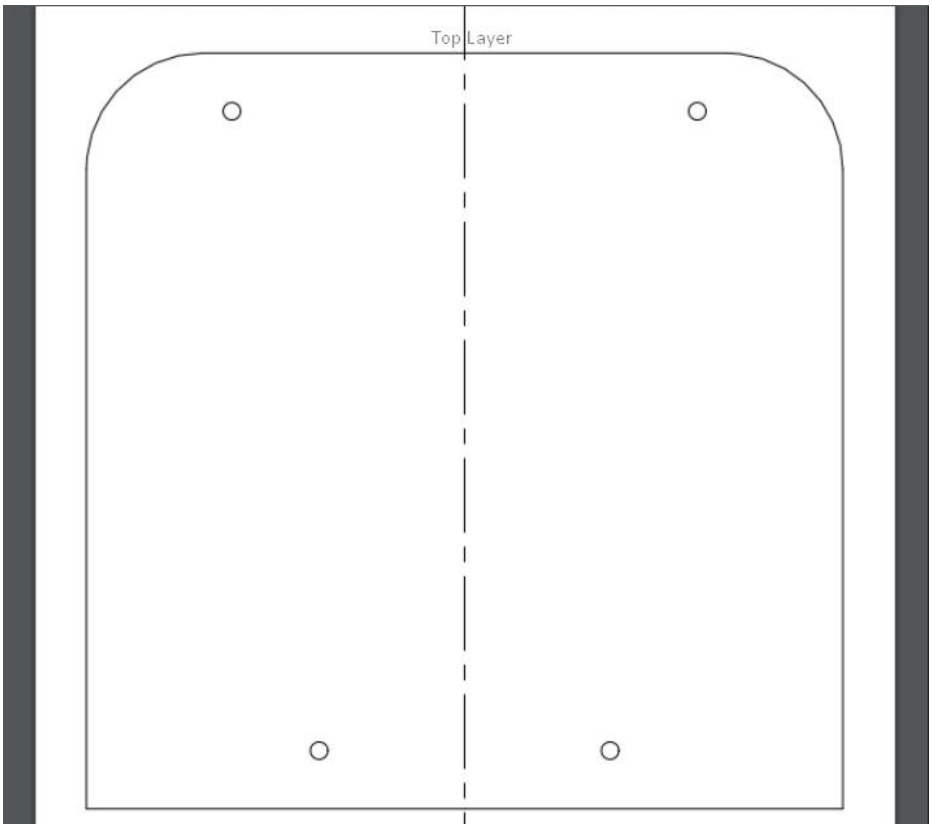


Figure 2-4 - Schematic Top layer

2.2SolidWorks and Printing the Design

SolidWorks is a solid modeling computer-aided design (CAD) and computer-aided engineering (CAE) computer program that runs on Microsoft Windows. SolidWorks is published by Dassault Systèmes.

We used SolidWorks to convert our schematic design to a printable vesion and SolidWorks was a good tools for doing that.

Then the next step was to go to the workshop and print the actual robot.

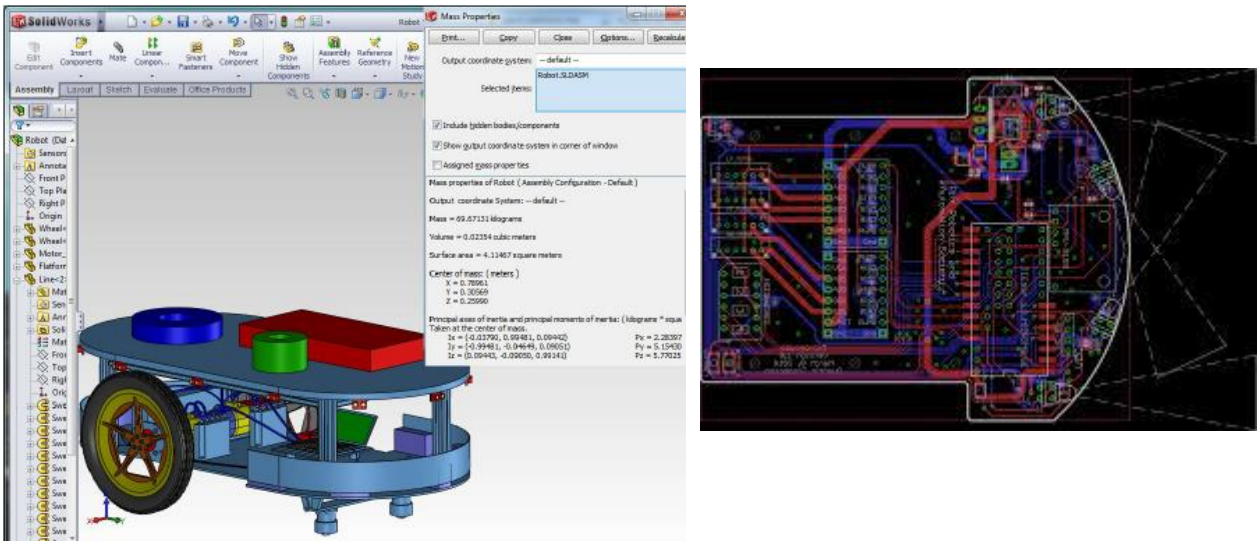


Figure 2-5 - SolidWorks Example

2.3 Final Robot form and old body versions

First we used a three-layer body robot, then optimizing it to be only two-layer robot as seen in figures below:

Figure 2-6 - Three-layer old body version

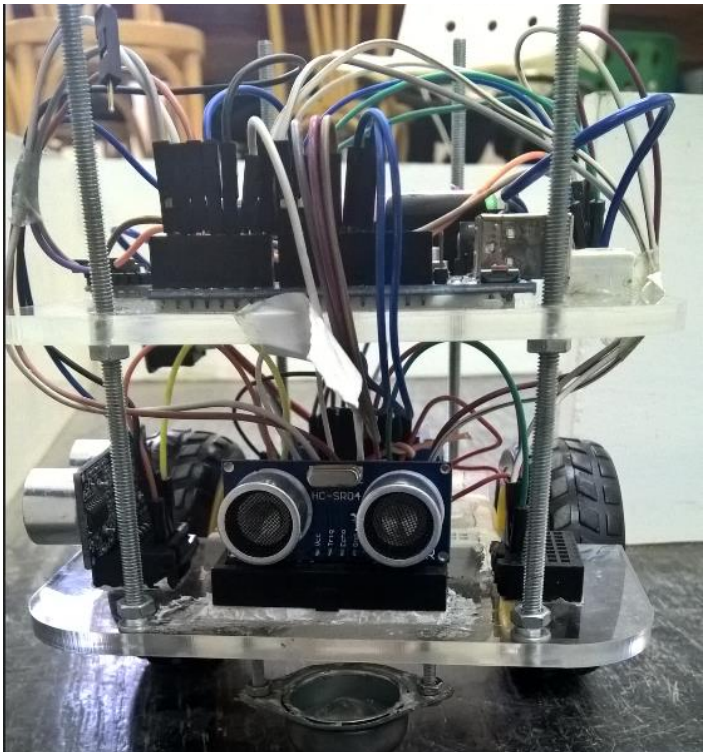


Figure 2-7 - Two-layer new body version

3 CHAPTER THREE SENSORS AND ACTUATORS

This Chapter deals with sensors of the robot which are the eyes of it that can read the surroundings, then how to use these readings to control the actuators which are the motors in this case to decide on which way to take.

3.1 Components

- Arduino UNO
- Ultrasonic sensors (3)
- Bluetooth module
- Motor Driver L298N
- Motors (2) and free Wheels
- BreadBoards
- On-Off switch
- 3.7 Li-On batteries (3) with battery box

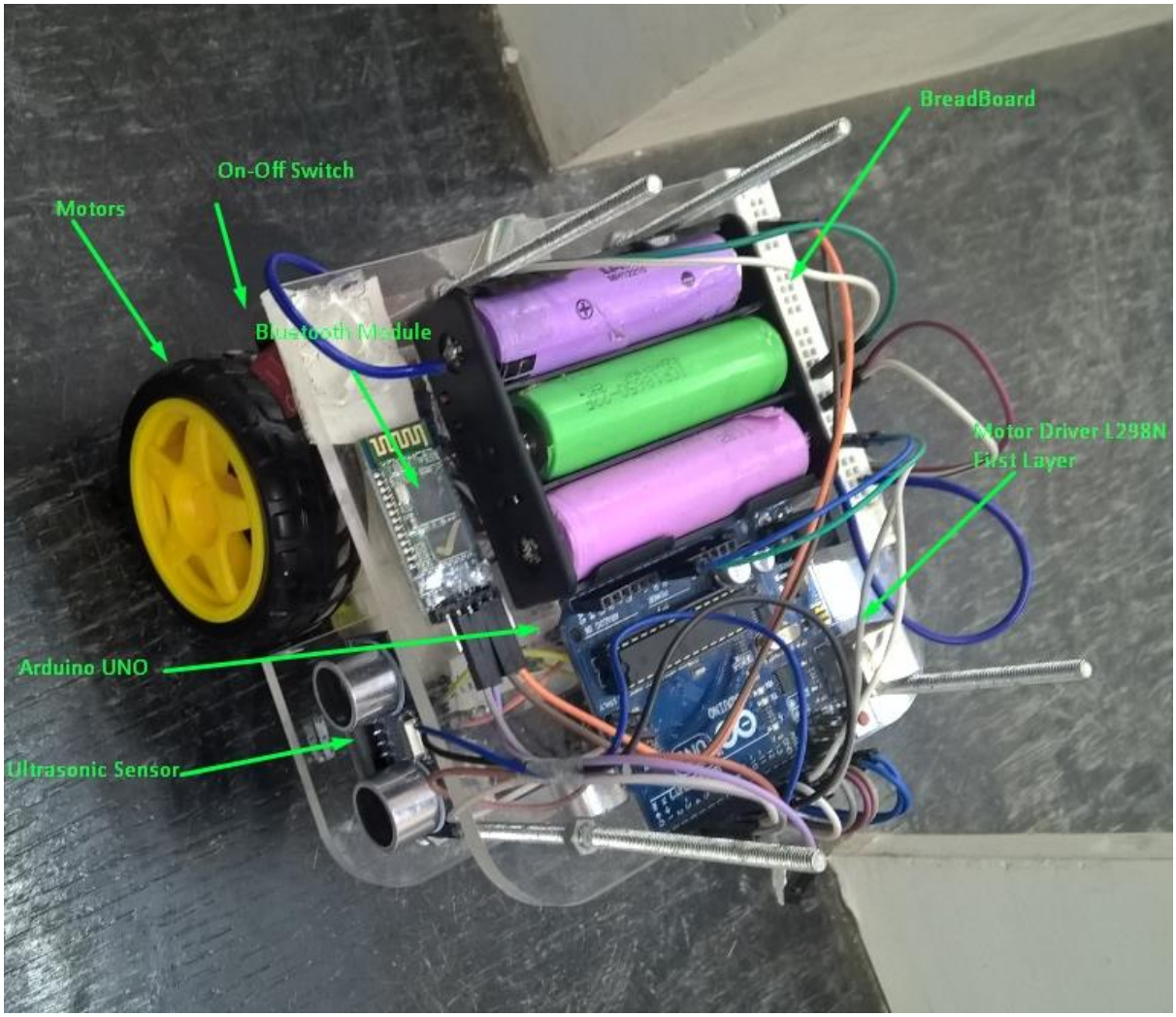


Figure 3-1 - Robot Components

In the next sections we will discuss each component in details and how did we use them to read the surroundings and make decisions upon the readings.

3.1.1Arduino UNO

Arduino is an open-source platform used for building electronic projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. The Arduino language (Arduino c) is merely a set of C/C++ functions that can be called from your code. Your sketch undergoes minor changes (e.g. automatic generation of function prototypes) and then is passed directly to a C/C++ compiler (avr-g++).

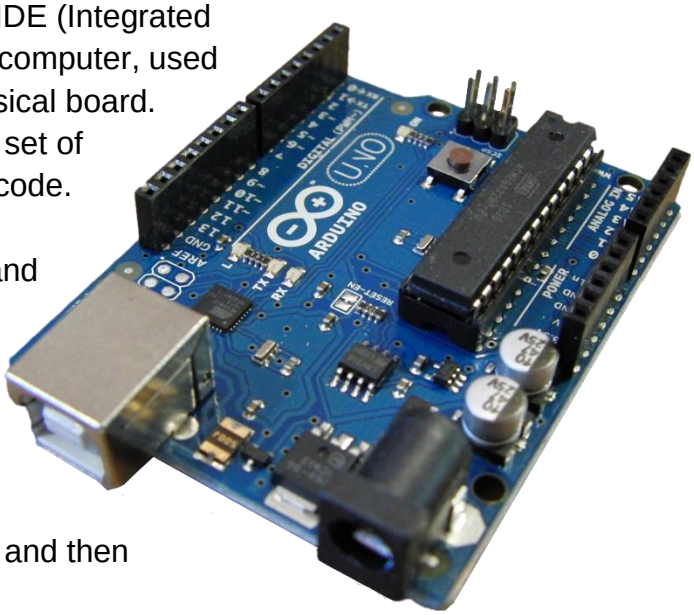


Figure 3-2 - Arduino UNO

We used Arduino in the first algorithm which will be discussed in the next chapter to read the shortest path generated from python (using BFS Algorithm) from Bluetooth module and then control the motors to take the path.

We also used a second algorithm (left wall follower Algorithm) which will be discussed in the next chapter so to control the motors to always follow the walls till it finds the end.

3.1.2 Ultrasonic Sensors

Ultrasonic HC-SR04 sensors measure distance by using ultrasonic waves. The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. Ultrasonic Sensors measure the distance to the target by measuring the time between the emission and reception.

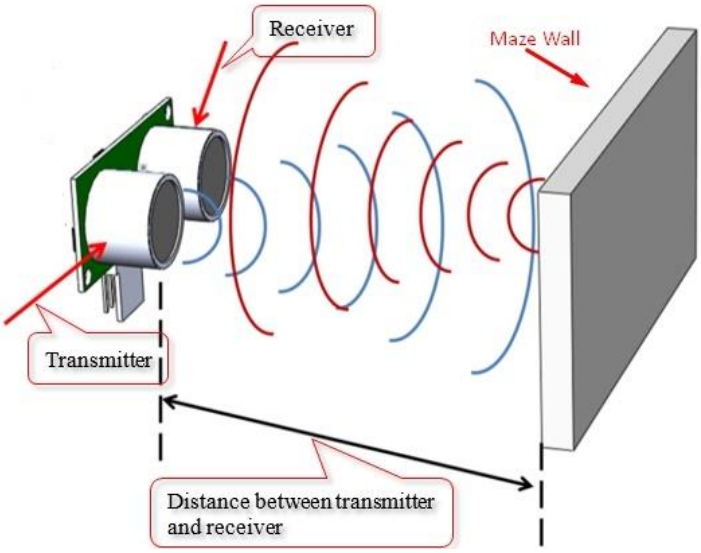


Figure 3-3 - Ultrasonic Configuration

By knowing the distance from each ultrasonic (right, left and front) we can detect each turn to take (reading more than 20cm from left or right) or keep moving forward otherwise.

See the code in the appendix.

3.1.3 Bluetooth module

HC-05 Bluetooth Module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. Its communication is via serial communication which makes an easy way to interface with controller or PC. Using COM to connect.

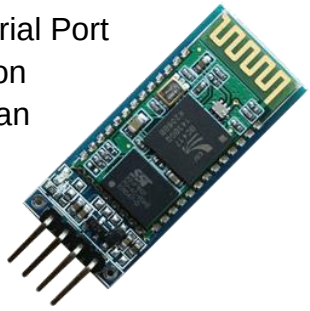


Figure 3-4 - Bluetooth Module

We used it to transmit the path generated from python code to the Arduino (first Algorithm), more on that on the next chapter.

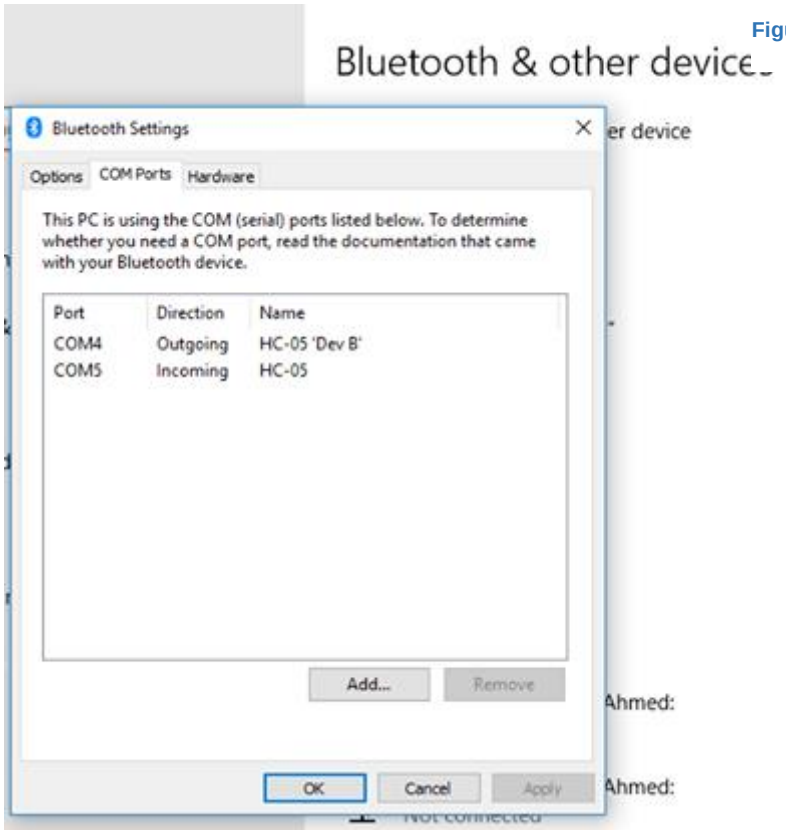


Figure 3-5 - Connecting PC to the robot

3.1.4 Motor Driver L298N

The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

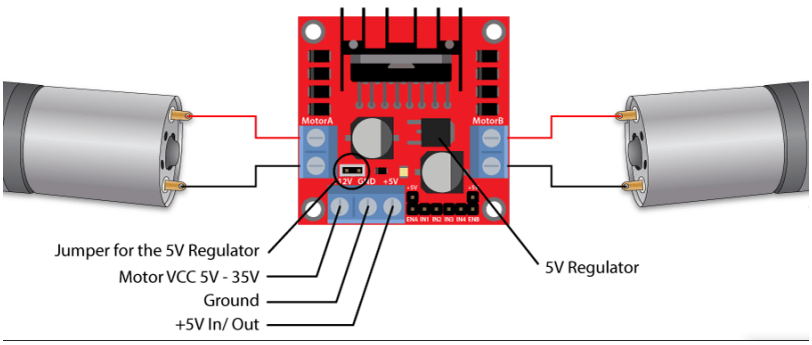


Figure 3-6 - Motor Driver Configuration

3.1.5 Motors

Using 12v DC motors to move the robot depending on the signals from Arduino.

Controlling the motors speeds ranging from 0 to 255 depending on the next move (forward, turning, rotating….etc.). and using delays to read from ultrasonic sensors and take actions.

Figure 3-7 - 12v DC Motor

3.2 Circuit diagram and Pins

The next table shows the connections and the Arduino configuration based on the Circuit Diagram. See the code in appendix.

Arduino Pin Number	Connected Device
0,1	Bluetooth Serial
2	Front ultrasonic Trigger
3	Front ultrasonic Echo
4	Right ultrasonic Trigger
5	Right ultrasonic Echo
6	Left Motor PWM
7	Left ultrasonic Trigger
8	Left ultrasonic Echo
9	Right Motor PWM
10	Right Motor backwards
11	Right Motor forward
12	Left Motor backwards
13	Left Motor forward

Table 3-1 - Arduino Pin Connections

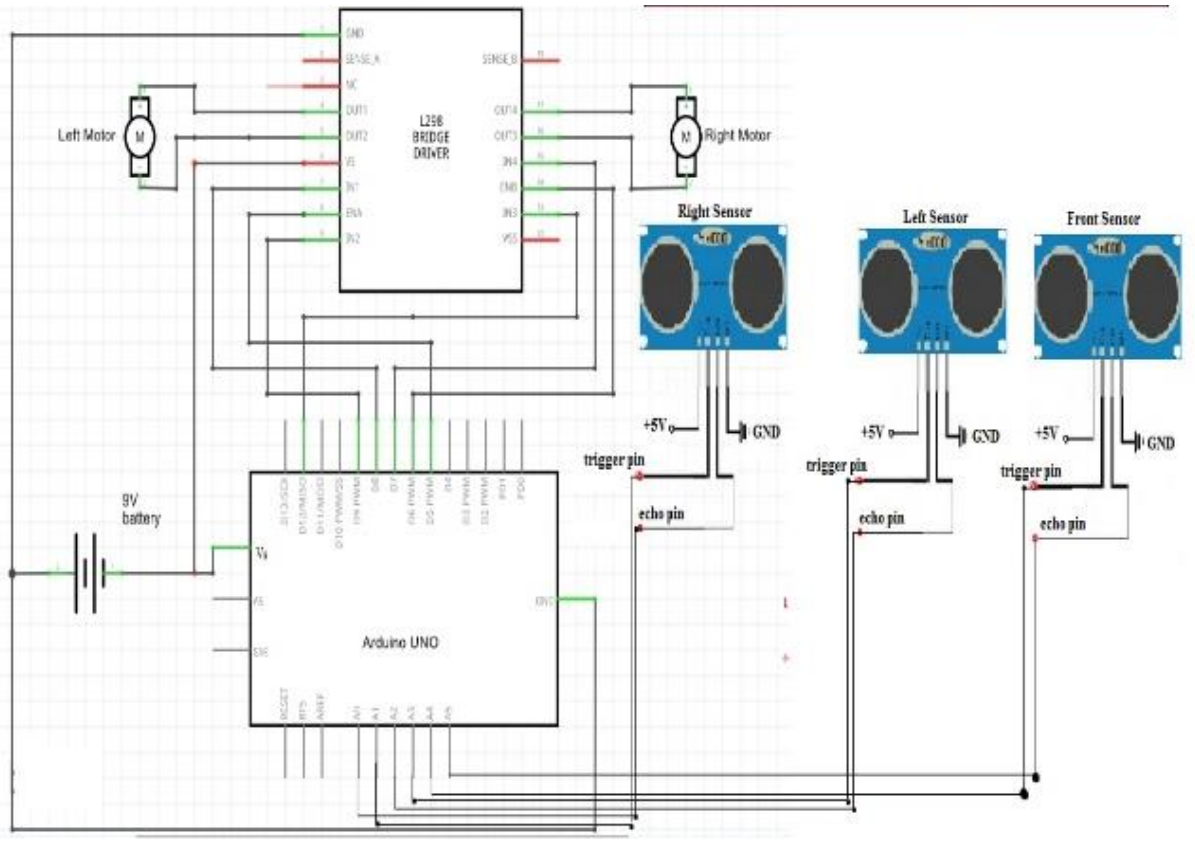


Figure 3-8 - Circuit Diagram

4 CHAPTER FOUR SOFTWARE AND ALGORITHMS

This chapter deals with the brains of the robot, Coding. In this chapter we will discuss the two algorithms used (BFS using Python) if the map is known, and (Left wall follower) if the map is unknown in details.

4.1 First Algorithm (Shortest path)

The steps needed to provide the robot to solve the maze wither the map is known or unknown is discussed in this chapter. These algorithms provide accuracy, efficiency and at the most optimized time possible.

4.1.1BFS, DFS and Python

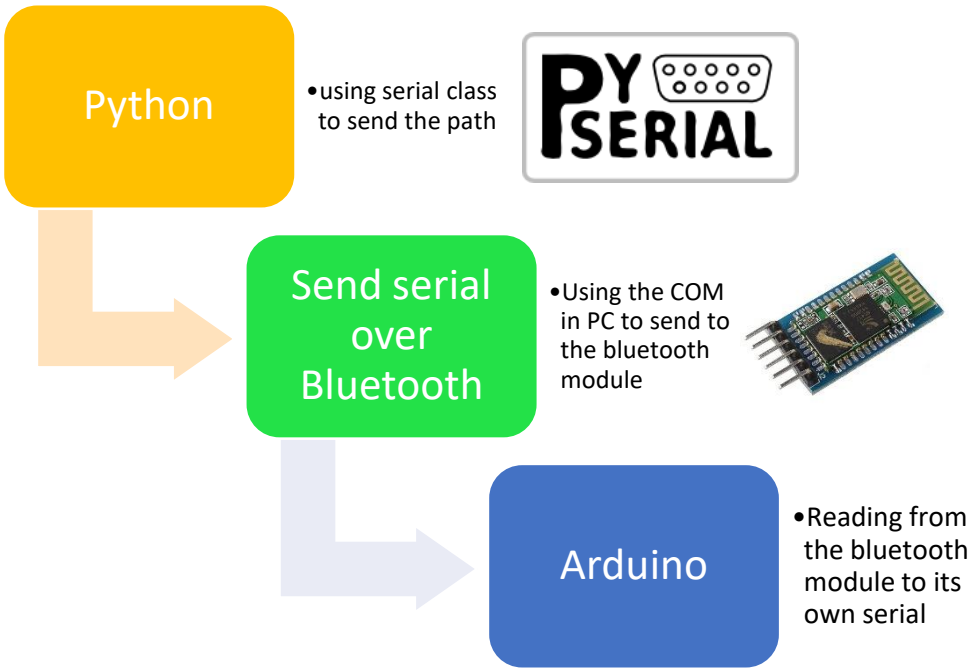


Figure 4-1 - BFS and DFS algorithms process

The python program uses data structures (Stack and Queue to hold possible paths), Graph Algorithms (BFS, DFS) to find paths from the start node to the goal and functions to map the structure of the maze. The Code Explanation in details:

4.1.1.1 Graph Algorithms

4.1.1.1.1 Depth First Search (DFS)

Depth-first search always follows a single path in the graph as long as it finds new nodes. After this, it returns to previous nodes and begins to explore other parts of the graph. The algorithm keeps track of visited nodes, so that it processes each node only once (explores as far as possible along each branch before backtracking. As stack pops the last in DFS uses stack to save a single path till no more nodes and pops the nodes if not reached and add other nodes children to the path and so on.

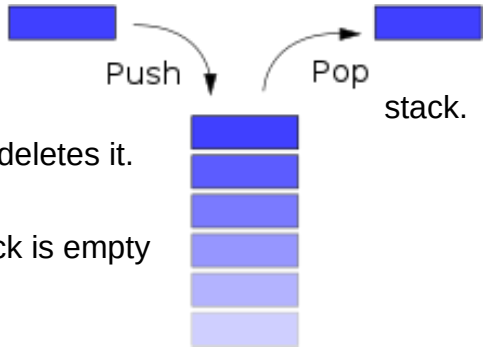
4.1.1.1.2 Breadth First Search (BFS)

Breadth-first search (BFS) visits the nodes of a graph in increasing order of their distance from the starting node. Breadth-first search goes through the nodes one level after another. First the search explores the nodes whose distance from the starting node is 1, then the nodes whose distance is 2, and so on. The queue q contains nodes to be processed in increasing order of their distance. New nodes are always added to the end of the queue, and the node at the beginning of the queue is the next node to be processed.

4.1.1.2 Data Structure

4.1.1.2.1 Class `Stack`

Creates (instance) Stack (LIFO) to use in DFS:

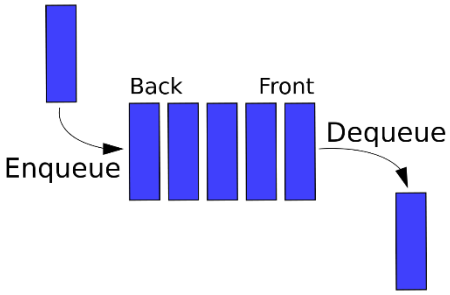


- **`Push`:** To insert element to the end of the stack.
- **`POP`:** Returns the last element added and deletes it.
- **`Count`:** Count elements in the stack
- **`isEmpty`:** Returns true or false if the stack is empty

4.1.1.2.2 Class `Queue`

Creates (instance) Queue (FIFO) to use in BFS:

- **`Enqueue`**: To insert element to a Queue
- **`Dequeue`**: Returns the First element added and deletes it.



4.1.1.3 Read and Process the Maze

4.1.1.3.1 Class MazeNode

Takes all the properties of a node (the position x,y and checks the walls around it).

4.1.1.3.2 Class `Maze`

To set the maze by knowing the coordinates, the start node and the goal node.

4.1.1.3.3 initializeMaze

To set the maze by creating a 2d empty array with given rows and columns.

4.1.1.3.4 setStartNode setGoalNode

know the start position and the goal by its coordinates.

4.1.1.3.5 addMazeSquare

Creates a new node with its proprieties using Class MazeNode and adds it to the maze.

4.1.1.3.6 getMazeNodeByXY

Calling a node by its x,y

4.1.1.3.7 getMazeNodeChildren

gets the nodes children and checks out of boundaries or it hits a wall.

4.1.1.3.8 BFS DFS

Search for paths and puts them on a list.

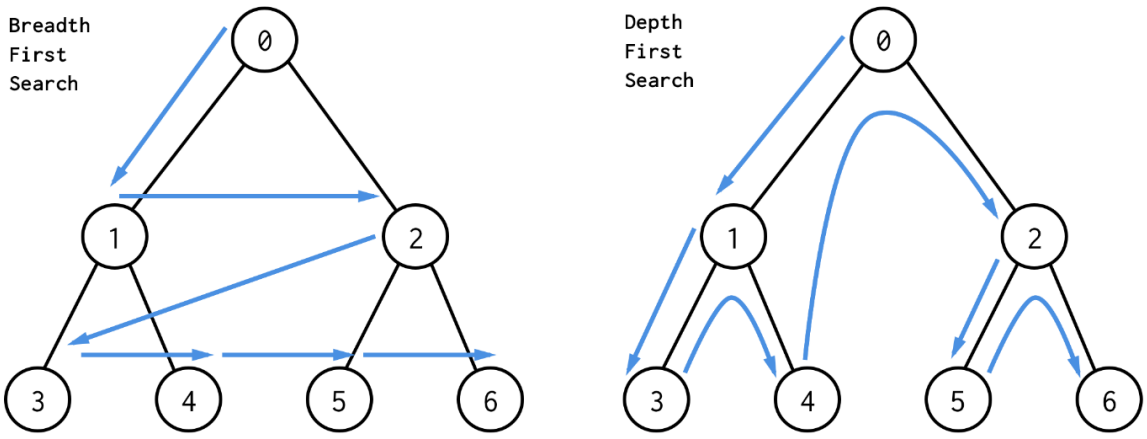


Figure 4-2 - BFS and DFS Techniques

4.1.1.4 `SerialTransfer` Class

By using PySerial and importing serial, This module encapsulates the access for the serial port. It provides backends for Python. The module named “serial” automatically selects the appropriate backend. So we can connect to the Bluetooth connected with the PC “using the PC Bluetooth driver” to send serial over to the Arduino.

The class creates an object that takes the port number that the Bluetooth is connected in, bundTime and the timeout.

This class has the functions **`send`**: which sends (encodes) to the port a string. And function **`receive`** which receives (decodes) data from the port.

4.1.1.4.1 Transferring path to the serial

- By adding the BFS and DFS solutions to the paths array and then
- choosing the optimal one to send over then
- sending over COM the path by its coordinates to the Bluetooth Module and then to the Arduino serial.

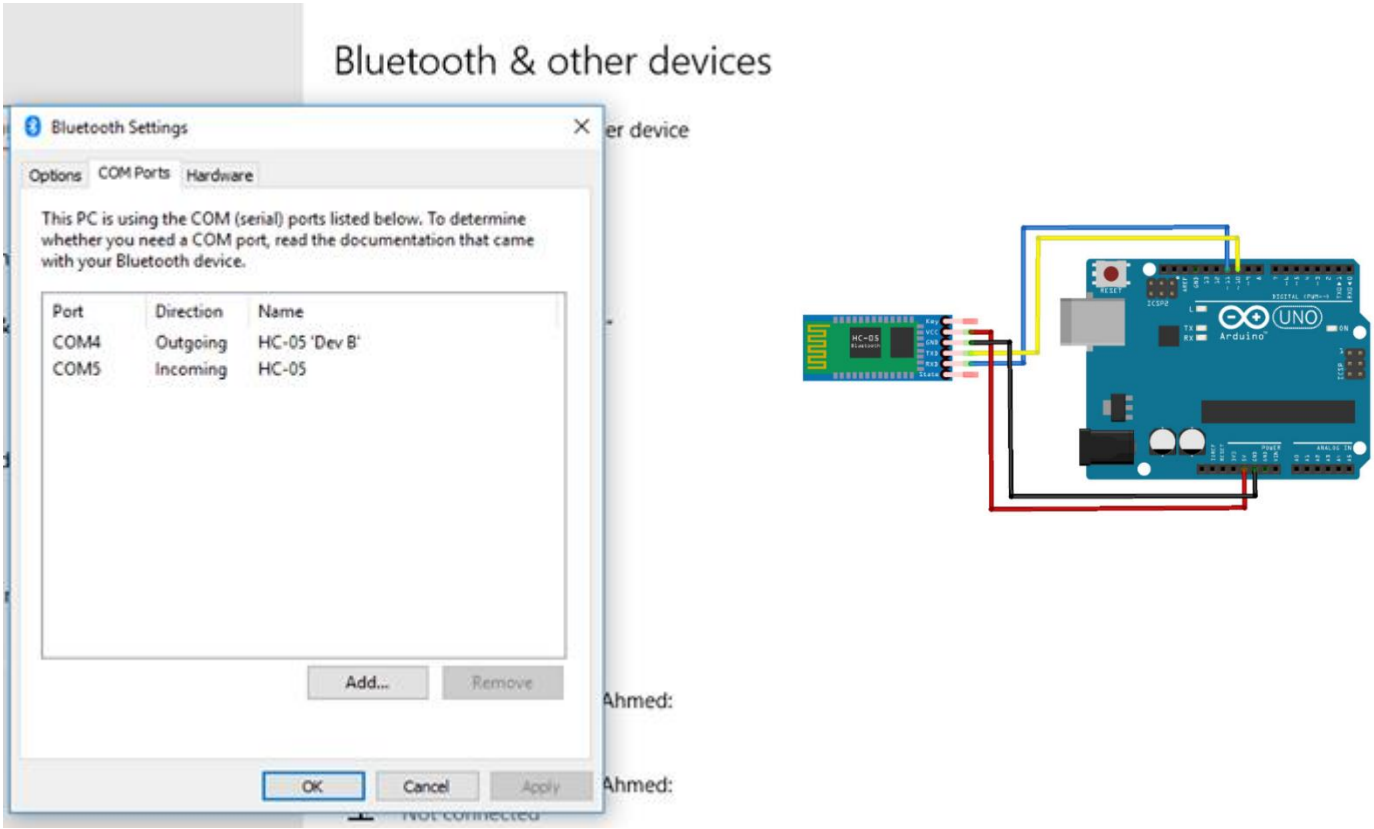


Figure 4-3 - Transfer from PC to Arduino using Bluetooth module

4.1.2 Arduino actions based on the path

By Downloading the Arduino code on the board to exchange data. And sending the result over Bluetooth the result will be as follow:

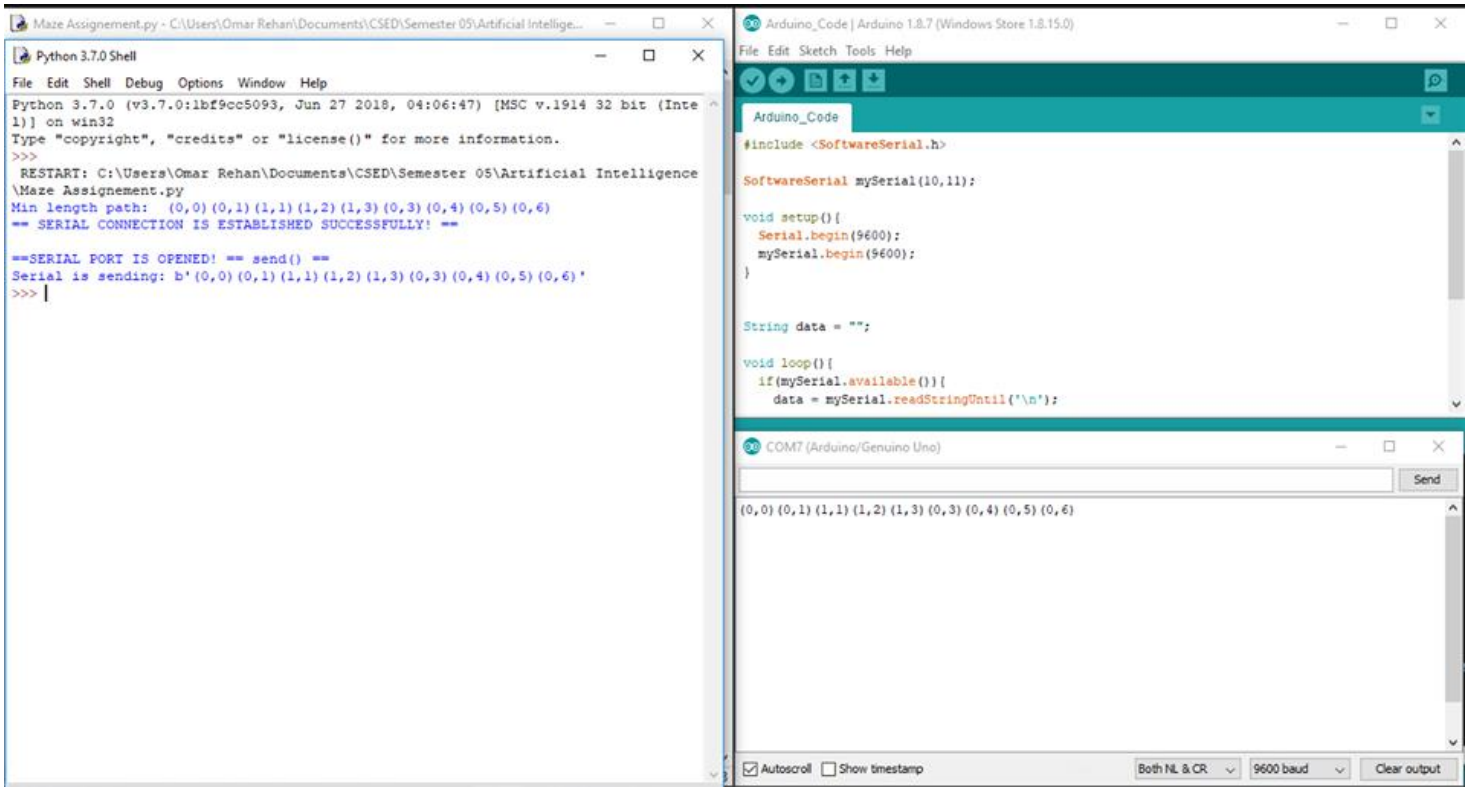


Figure 4-4 - Path transferred to Arduino

Then optimizing the path to only critical blocks of changing the robot either left or right till it finds the end block. Using PID to control motors from ultrasonic.

This method is used if the map, start and end are provided. For the Arduino and python codes see the appendix.

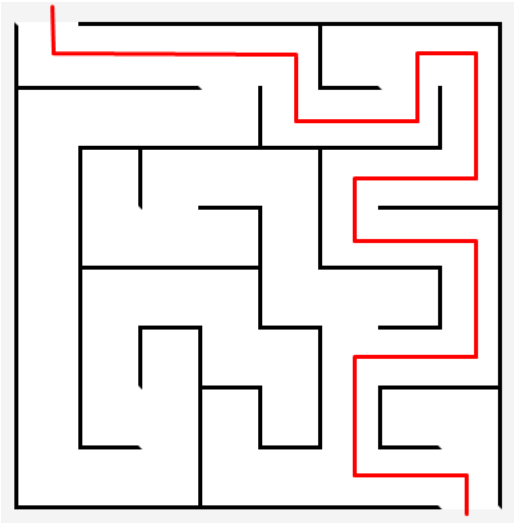


Figure 4-5 - Example of using the first Algorithm

4.2Second Algorithm (Left Wall Follower)

This algorithm is easy to implement and doesn't need any other interferers as from its name the algorithm follows the left wall, The left-hand rule works in such a way that the robot focuses more on its left-side and front-side while it has options for turns. The robot will turn right only if there are no other possibilities while it always turns to the left if there is an option.

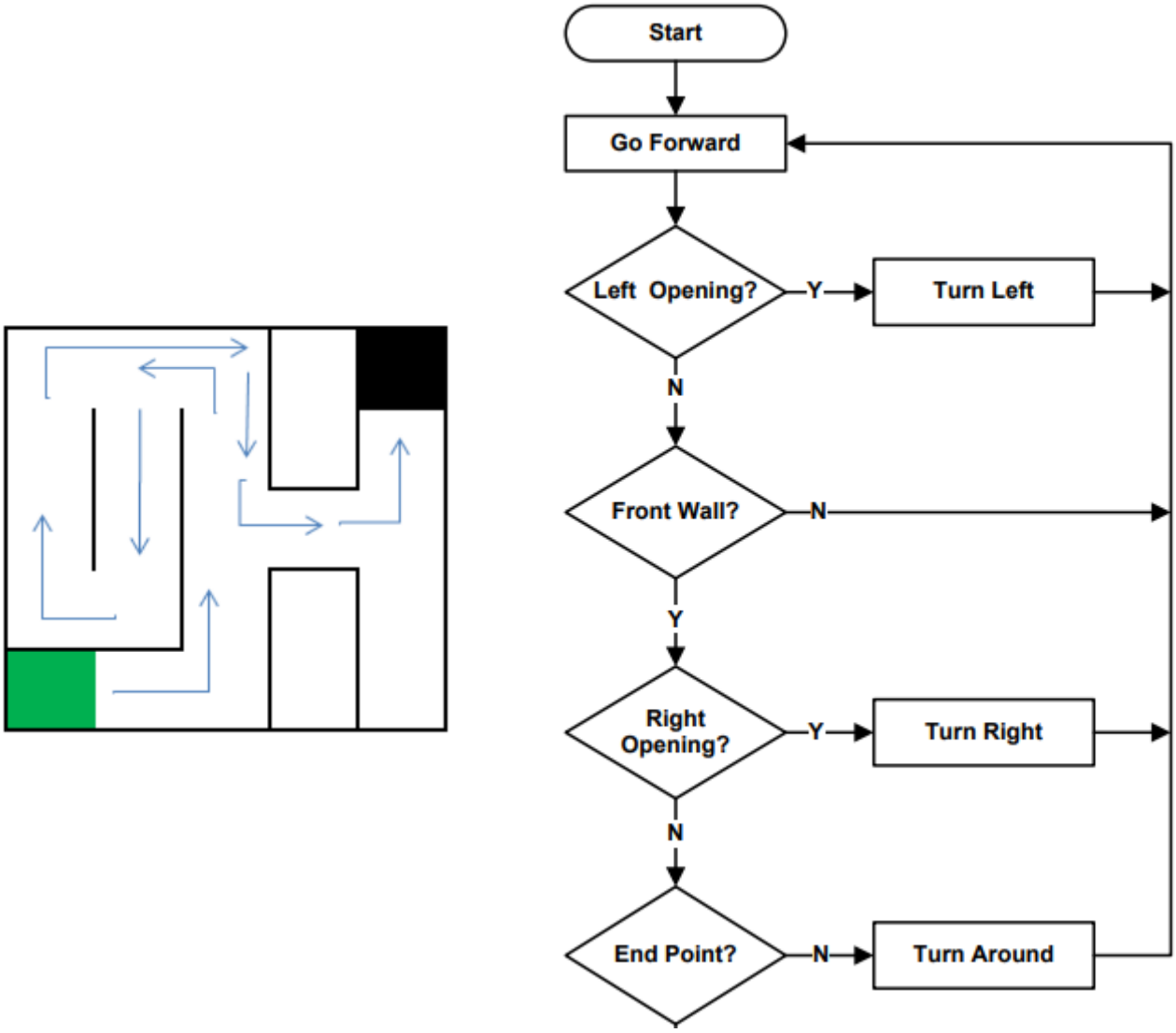


Figure 4-6 - Left wall follower Algorithm

For the Arduino code, see the appendix.

5 CHAPTER FIVE CONCLUSION AND FINAL RESULTS

At the end of the game play it could be concluded that it was successful in achieving the goal of Artificial Intelligence in the prototype robot, that is giving a own brain-like feature to the robot so that it can work accordingly in any favorable condition and the arena provided. Since, we were successful in providing the robot with its own sense of judgment, so it is probable that the robots in the industry could be developed with such a brain- like feature, so that they can have their own intellect to judge the path they follow, making it a helping hand in the textile industry.

5.1 Solving the maze

After Summing up all the above chapter and Creating a Maze Solving Robot. Here is our robot competing in the maze. For more media Visit:

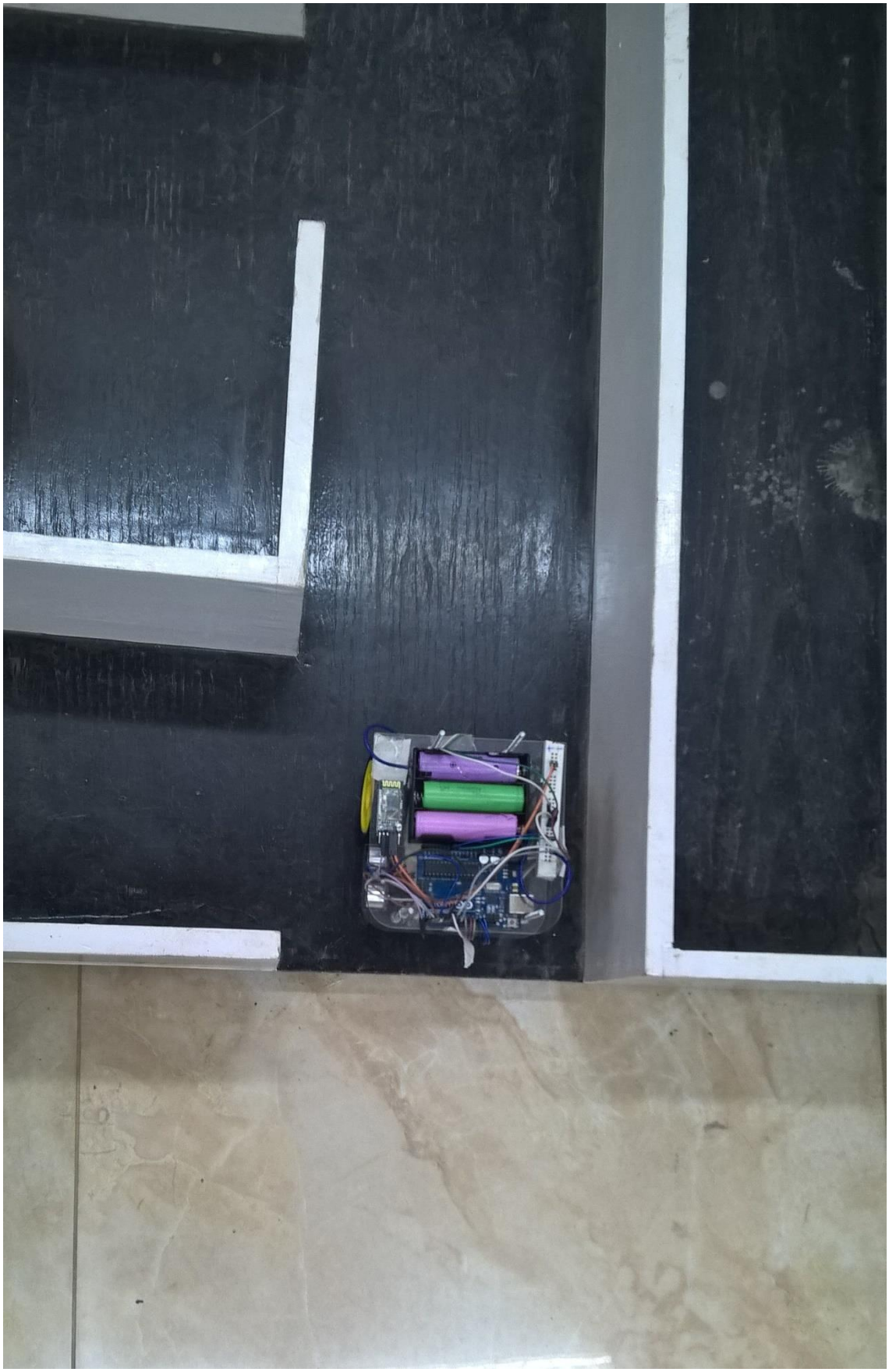


Figure 5-1 - Finishing the maze

6 REFERENCES

References

Googe SearchGate, AUTONOMOUS MAZE SOLVING ROBOT

Design and Implementation of a Robot for Maze-Solving using Flood-Fill Algorithm, International Journal of Computer Applications

SHORTEST DISTANCE MAZE SOLVING ROBOT, International Journal of Research in Engineering and Technology.

Jerry: An Intelligent maze solving robot, Shivaji University.

ARTIFICIALLY INTELLIGENT MAZE SOLVER ROBOT, Priyadarshini College of Engineering, Nagpur, India

https://www.solidworks.com/ , https://en.wikipedia.org/wiki/SolidWorks

Ultrasonic,
https://www.keyence.com/ss/products/sensor/sensorbasics/ultrasonic/info/

https://components101.com/ultrasonic-sensor-working-pinout-datasheet

Bluetooth Module,

https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/

https://pdfs.semanticscholar.org/1466/06c92916071ed6f6ae98fb27229660570bd3.pdf

7 APPENDICES

7.1 Python Code and Serial Path

```
1. import serial
2. from time import sleep
3.
4. class SerialTransfer(object):
5. def __init__(self,port,baudRate = 9600):
6. self._port = port
7. self._baudRate = baudRate
8. self._serial =
   serial.Serial(self._port,self._baudRate,timeout = 5)
9. self._serial.flushInput()
10.     self._serial.flushOutput()
11.     print("== SERIAL CONNECTION IS ESTABLISHED
   SUCCESSFULLY! ==")
12.
13.     def __del__(self):
14.     self.close()
15.
16.     def close(self):
17.     self._serial.close()
18.
19.     def send(self,statement):
20.     if(self._serial.isOpen()):
21.     print("\n==SERIAL PORT IS OPENED! == send() ==")
22.     statement += '\r\n'
23.     encoded = statement.strip().encode()
24.     print("Serial is sending:" , encoded)
25.     self._serial.write(encoded)
26.     self._serial.flush()
27.     sleep(1)
28.     else:
29.     print("\n== SERIAL PORT IS NOT OPENED!==")
30.
31.
32.     def recieve(self):
33.     if(self._serial.isOpen()):
34.     print("\n==SERIAL PORT IS OPENED! == recieve() ==")
35.     recieveData = self._serial.readline()
36.     stmt = recieveData.decode().strip()
37.     return stmt
38.     else:
39.     print("\n== SERIAL PORT IS NOT OPENED!==")
40.     return None
41.
```

```
42.
43.     class Stack:
44.     def __init__(self):
45.     self._L = []
46.     def push(self, item):
47.     self._L.append(item)
48.     def pop(self):
49.     if (self.count() > 0):
50.     return self._L.pop()
51.     else:
52.     return None
53.     def count(self):
54.     return len(self._L)
55.     def isEmpty(self):
56.     return self.count() == 0
57.
58.
59.     class Queue:
60.     def __init__(self):
61.     self._L = []
62.     def enqueue(self, item):
63.     self._L.append(item)
64.     def dequeue(self):
65.     if (self.count() > 0):
66.     return self._L.pop(0)
67.     else:
68.     return None
69.     def count(self):
70.     return len(self._L)
71.     def isEmpty(self):
72.     return self.count() == 0
73.
74.
75.     class MazeNode:
76.     def __init__(self, r, c, openLeft, openRight,
  openUp, openDown):
77.     self._r = r
78.     self._c = c
79.     self._openLeft = openLeft
80.     self._openRight = openRight
81.     self._openUp = openUp
82.     self._openDown = openDown
83.
84.
85.     class Maze:
86.     def __init__(self, numOfRows, numOfCols):
```

```
87.     self._numOfRows = numOfRows
88.     self._numOfCols = numOfCols
89.     self._maze = []
90.     self._startNode = None
91.     self._goalNode = None
92.     self.initializeMaze()
93.
94.     def initializeMaze(self):
95.     for r in range(self._numOfRows):
96.     row = []
97.     for c in range(self._numOfCols):
98.     row.append(None)
99.     self._maze.append(row)
100.
101.    def setStartNode(self, r, c):
102.    self._startNode = (r, c)
103.    def setGoalNode(self, r, c):
104.    self._goalNode = (r, c)
105.
106.    def addMazeSquare(self, r, c, openUp, openDown,
  openLeft, openRight):
107.    newMazeSquare = MazeNode(r, c, openLeft, openRight,
  openUp, openDown)
108.    self._maze[r][c] = newMazeSquare
109.
110.    def getMazeNodeByRC(self, r, c):
111.    return self._maze[r][c]
112.
113.    def getMazeNodeChildren(self, r, c):
114.    children = []
115.    if (r > 0 and self._maze[r][c]._openUp):
116.    children.append((r-1, c))
117.    if (r < self._numOfRows-1 and
  self._maze[r][c]._openDown):
118.    children.append((r+1, c))
119.    if (c > 0 and self._maze[r][c]._openLeft):
120.    children.append((r, c-1))
121.    if (c < self._numOfCols-1 and
  self._maze[r][c]._openRight):
122.    children.append((r, c+1))
123.    return children
124.
125.    def DFS(self):
126.    if (self._startNode is None or self._goalNode is
  None):
127.    return None
```

```
128.
129.    stack = Stack()
130.    stack.push([self._startNode])
131.    paths = []
132.
133.    while(not stack.isEmpty()):
134.    currentPath = stack.pop()
135.    coordinates = currentPath[-1]
136.    if (self._goalNode == coordinates):
137.    paths.append(currentPath)
138.
139.    children = self.getMazeNodeChildren(coordinates[0],
  coordinates[1]) [::-1]
140.    for child in children:
141.    if (not (child in currentPath)):
142.    stack.push(currentPath + [child])
143.
144.    return paths
145.
146.    def BFS(self):
147.    if (self._startNode is None or self._goalNode is
  None):
148.    return None
149.
150.    queue = Queue()
151.    queue.enqueue([self._startNode])
152.    paths = []
153.
154.    while(not queue.isEmpty()):
155.    currentPath = queue.dequeue()
156.    coordinates = currentPath[-1]
157.    if (self._goalNode == coordinates):
158.    paths.append(currentPath)
159.
160.    children = self.getMazeNodeChildren(coordinates[0],
  coordinates[1]) [::-1]
161.    for child in children:
162.    if (not child in currentPath):
163.    queue.enqueue(currentPath + [child])
164.
165.    return paths
166.
167.
168.
169.    game = Maze(8 ,8)
170.
```

```
171.    #First Row
172.    game.addMazeSquare(0, 0, False, True, False, True) #
   up - down - left - right
173.    game.addMazeSquare(0, 1, False, False, True, True)
174.    game.addMazeSquare(0, 2, False, False, True, True)
175.    game.addMazeSquare(0, 3, False, True, True, True)
176.    game.addMazeSquare(0, 4, False, False, True, True)
177.    game.addMazeSquare(0, 5, False, False, True, True)
178.    game.addMazeSquare(0, 6, False, False, True, True)
179.    game.addMazeSquare(0, 7, False, False, True, False)
180.
181.    #Second Row
182.    game.addMazeSquare(1, 0, True, True, False, False)
183.    game.addMazeSquare(1, 1, False, True, False, True)
184.    game.addMazeSquare(1, 2, False, True, True, False)
185.    game.addMazeSquare(1, 3, True, True, False, False)
186.    game.addMazeSquare(1, 4, False, True, False, True)
187.    game.addMazeSquare(1, 5, False, True, True, True)
188.    game.addMazeSquare(1, 6, False, False, True, True)
189.    game.addMazeSquare(1, 7, False, True, True, True)
190.
191.    #Third Row
192.    game.addMazeSquare(2, 0, True, True, False, False)
193.    game.addMazeSquare(2, 1, True, False, False, True)
194.    game.addMazeSquare(2, 2, True, False, True, True)
195.    game.addMazeSquare(2, 3, True, True, True, False)
196.    game.addMazeSquare(2, 4, True, False, False, False)
197.    game.addMazeSquare(2, 5, True, True, False, True)
198.    game.addMazeSquare(2, 6, False, False, True, False)
199.    game.addMazeSquare(2, 7, True, True, False, False)
200.
201.    #Fourth Row
202.    game.addMazeSquare(3, 0, True, False, False, True)
203.    game.addMazeSquare(3, 1, False, True, True, False)
204.    game.addMazeSquare(3, 2, False, False, False, True)
205.    game.addMazeSquare(3, 3, True, False, True, True)
206.    game.addMazeSquare(3, 4, False, False, True, True)
207.    game.addMazeSquare(3, 5, True, True, True, False)
208.    game.addMazeSquare(3, 6, False, True, False, True)
209.    game.addMazeSquare(3, 7, True, False, True, False)
210.
211.    #Fifth Row
212.    game.addMazeSquare(4, 0, False, True, False, False)
213.    game.addMazeSquare(4, 1, True, True, False, True)
214.    game.addMazeSquare(4, 2, False, False, True, True)
215.    game.addMazeSquare(4, 3, False, False, True, True)
```

```
216.    game.addMazeSquare(4, 4, False, True, True, True)
217.    game.addMazeSquare(4, 5, True, False, True, False)
218.    game.addMazeSquare(4, 6, True, True, False, True)
219.    game.addMazeSquare(4, 7, False, True, True, False)
220.
221.    #Sixth Row
222.    game.addMazeSquare(5, 0, True, True, False, True)
223.    game.addMazeSquare(5, 1, True, True, True, False)
224.    game.addMazeSquare(5, 2, False, True, False, True)
225.    game.addMazeSquare(5, 3, False, True, True, False)
226.    game.addMazeSquare(5, 4, True, True, False, False)
227.    game.addMazeSquare(5, 5, False, False, False, True)
228.    game.addMazeSquare(5, 6, True, False, True, False)
229.    game.addMazeSquare(5, 7, True, True, False, False)
230.
231.    #Seventh Row
232.    game.addMazeSquare(6, 0, True, True, False, False)
233.    game.addMazeSquare(6, 1, True, True, False, False)
234.    game.addMazeSquare(6, 2, True, False, False, True)
235.    game.addMazeSquare(6, 3, True, False, True, True)
236.    game.addMazeSquare(6, 4, True, True, True, True)
237.    game.addMazeSquare(6, 5, False, False, True, True)
238.    game.addMazeSquare(6, 6, False, False, True, True)
239.    game.addMazeSquare(6, 7, True, False, True, False)
240.
241.    #Eighth Row
242.    game.addMazeSquare(7, 0, True, False, False, False)
243.    game.addMazeSquare(7, 1, True, False, False, True)
244.    game.addMazeSquare(7, 2, False, False, True, True)
245.    game.addMazeSquare(7, 3, False, False, True, False)
246.    game.addMazeSquare(7, 4, True, False, False, True)
247.    game.addMazeSquare(7, 5, False, False, True, True)
248.    game.addMazeSquare(7, 6, False, False, True, True)
249.    game.addMazeSquare(7, 7, False, False, True, False)
250.
251.    strt = "U"
252.    game.setStartNode(7, 0)
253.
254.    game.setGoalNode(1, 7)
255.
256.
257.    paths = []
258.    paths.extend(game.DFS())
259.    paths.extend(game.BFS())
260.
261.    if (len(paths) > 0):
```

```
262.    minPath = paths[0]
263.    for path in paths:
264.    if (len(path) < len(minPath)):
265.    minPath = path
266.
267.
268.
269.    crit = []
270.    for idx in range(1, len(minPath)):
271.    crnt = minPath[idx]
272.    node = game.getMazeNodeByRC(crnt[0],crnt[1])
273.    if(node._openUp + node._openDown + node._openLeft +
  node._openRight > 2) : crit.append(idx)
274.
275.    print(crit)
276.
277.    res = ""
278.    #print(minPath[0][0], minPath[0][1])
279.    for idx in range(1, len(minPath)):
280.    crnt = minPath[idx]
281.    prev = minPath[idx-1]
282.    print(prev[0], prev[1], '->', crnt[0], crnt[1])
283.
284.    if (crnt[0] > prev[0]): #Down
285.    res += "D"
286.    elif (crnt[0] < prev[0]): #Up
287.    res += "U"
288.    elif (crnt[1] > prev[1]): #Right
289.    res += "R"
290.    crit.append(idx)
291.    elif (crnt[1] < prev[1]): #Left
292.    res += "L"
293.    crit.append(idx)
294.
295.    print(res)
296.    data = ""
297.
298.    res = strt + res;
299.    crit.append(0);
300.
301.    for idx in range(1, len(res)):
302.    if((idx-1) not in crit) : continue
303.    if (res[idx] == "U"): # should move up globally
304.    if (idx == 0 or res[idx-1] == "U"): # currently
  facing up
305.    data += "U"
```

```
306.    elif (res[idx-1] == "D"): # currently facing down
307.    data += "D"
308.    elif (res[idx-1] == "L"): # currently facing left
309.    data += "R"
310.    elif (res[idx-1] == "R"): # currently facing right
311.    data += "L"
312.
313.
314.    elif (res[idx] == "D"): # should move down globally
315.    if (idx == 0 or res[idx-1] == "U"): # currently
   facing up
316.    data += "D"
317.    elif (res[idx-1] == "D"): # currently facing down
318.    data += "U"
319.    elif (res[idx-1] == "L"): # currently facing left
320.    data += "L"
321.    elif (res[idx-1] == "R"): # currently facing right
322.    data += "R"
323.
324.
325.    elif (res[idx] == "R"): # should move right globally
326.    if (idx == 0 or res[idx-1] == "U"): # currently
   facing up
327.    data += "R"
328.    elif (res[idx-1] == "D"): # currently facing down
329.    data += "L"
330.    elif (res[idx-1] == "L"): # currently facing left
331.    data += "D"
332.    elif (res[idx-1] == "R"): # currently facing right
333.    data += "U"
334.
335.
336.    elif (res[idx] == "L"): # should move left globally
337.    if (idx == 0 or res[idx-1] == "U"): # currently
   facing up
338.    data += "L"
339.    elif (res[idx-1] == "D"): # currently facing down
340.    data += "R"
341.    elif (res[idx-1] == "L"): # currently facing left
342.    data += "U"
343.    elif (res[idx-1] == "R"): # currently facing right
344.    data += "D"
345.
346.
347.    print(data)
348.
```

```
349.    minPathStr = ""
350.
351.    for coord in minPath:
352.    minPathStr += "(" + str(coord[0]) + "," +
  str(coord[1]) + ")"
353.    print("Min length path: ", minPathStr)
354.
355.
356.    t = SerialTransfer('COM4')
357.    t.send(data)
358.    t.close()
359.
360.
361.
362.
363.
364.
365.    #try:
366.    # t.send(minPathStr)
367.    # t.close()
368.    #except Exception as ex:
369.    # t.close()
370.    # print("Exception: ", ex)
371.
372.    #else:
373.    # print("There are no paths")
```

7.2 Arduino Code Using PID

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(0, 1);

// defines pins numbers
// BM 10 , 11

const int trigPinN = 2;
const int echoPinN = 3;
    const int trigPinR = 4;
    const int echoPinR = 5;
    const int trigPinL = 7;
    const int echoPinL = 8;


    // motor one
    int en1 = 6;
    int in1 = 12;
    int in2 = 13;
    // motor two
    int en2 = 9;
    int in3 = 11;
    int in4 = 10;

    String data = "";

    void setup() {

    pinMode(trigPinN, OUTPUT); // Sets the trigPin as an Output
    pinMode(echoPinN, INPUT); // Sets the echoPin as an Input
    pinMode(trigPinR, OUTPUT); // Sets the trigPin as an Output
    pinMode(echoPinR, INPUT); // Sets the echoPin as an Input
    pinMode(trigPinL, OUTPUT); // Sets the trigPin as an Output
    pinMode(echoPinL, INPUT); // Sets the echoPin as an Input

    // set all the motor control pins to outputs
    pinMode(in1, OUTPUT);
```

```
38.     pinMode(in2, OUTPUT);
39.     pinMode(in3, OUTPUT);
40.     pinMode(in4, OUTPUT);
41.
42.     //pinMode(en1, OUTPUT);
43.     //pinMode(en2, OUTPUT);
44.
45.     Serial.begin(9600); // Starts the serial
  communication
46.     mySerial.begin(9600);
47.
48.     //moveForward();
49.     /*
50.
51.       turnLeft();
52.       stopMotor();
53.       delay(1000);
54.
55.       turnRight();
56.       stopMotor();
57.       delay(1000);
58.
59.       moveForward();
60.       delay(1000);
61.       stopMotor();
62.       delay(1000);
63.
64.       moveBackward();
65.       delay(1000);
66.       stopMotor();
67.       */
68.
69.     delay(1000);
70.     }
71.
72.     int leftPower = 100;
73.     int rightPower = 76;
74.
75.     void stopMotor() {
76.     analogWrite(en1, 0);
77.     analogWrite(en2, 0);
78.     }
79.
80.     void motorLeftForward() { // Left motor
81.     digitalWrite(in1, LOW);
82.     digitalWrite(in2, HIGH);
```

```
83.     analogWrite(en1, leftPower);
84.     }
85.
86.     void motorRightForward() { // Right motor
87.     digitalWrite(in3, HIGH);
88.     digitalWrite(in4, LOW);
89.     analogWrite(en2, rightPower);
90.     }
91.
92.     void motorLeftBackward() { // Left motor
93.     digitalWrite(in2, LOW);
94.     digitalWrite(in1, HIGH);
95.     analogWrite(en1, leftPower);
96.     }
97.
98.     void motorRightBackward() { // Right motor
99.     digitalWrite(in4, HIGH);
100.    digitalWrite(in3, LOW);
101.    analogWrite(en2, rightPower);
102.    }
103.
104.    void moveForward() {
105.    motorLeftForward();
106.    motorRightForward();
107.    }
108.
109.    void moveBackward() {
110.    motorLeftBackward();
111.    motorRightBackward();
112.    }
113.
114.    int dly = 400;
115.
116.    void turnRight() {
117.    motorRightBackward();
118.    motorLeftForward();
119.    delay(dly);
120.    }
121.
122.    void turnLeft() {
123.    motorLeftBackward();
124.    motorRightForward();
125.    delay(dly);
126.    }
127.
128.    int dir = 0;
```

```
129.    void turnBackward() {
130.    dir = 1 - dir;
131.    if (dir) {
132.    turnRight();
133.    turnRight();
134.    }
135.    else {
136.    turnLeft();
137.    turnLeft();
138.    }
139.
140.    }
141.
142.
143.    // defines variables
144.    long durationN;
145.    int distanceN;
146.    long durationR;
147.    int distanceR;
148.    long durationL;
149.    int distanceL;
150.
151.    void plusUltra() {
152.
153.    digitalWrite(trigPinR, LOW);
154.    delayMicroseconds(2);
155.    digitalWrite(trigPinR, HIGH);
156.    delayMicroseconds(10);
157.    digitalWrite(trigPinR, LOW);
158.    durationR = pulseIn(echoPinR, HIGH);
159.
160.    // Clears the trigPin
161.    digitalWrite(trigPinN, LOW);
162.    delayMicroseconds(2);
163.    digitalWrite(trigPinN, HIGH);
164.    delayMicroseconds(10);
165.    digitalWrite(trigPinN, LOW);
166.    durationN = pulseIn(echoPinN, HIGH);
167.
168.    digitalWrite(trigPinL, LOW);
169.    delayMicroseconds(2);
170.    digitalWrite(trigPinL, HIGH);
171.    delayMicroseconds(10);
172.    digitalWrite(trigPinL, LOW);
173.    durationL = pulseIn(echoPinL, HIGH);
174.
```

```
175.    // Calculating the distance
176.    distanceN = durationN * 0.034 / 2;
177.    distanceR = durationR * 0.034 / 2;
178.    distanceL = durationL * 0.034 / 2;
179.    // Prints the distance on the Serial Monitor
180.    /*
181.      Serial.print("DistanceN: ");
182.      Serial.println(distanceN);
183.      Serial.print("DistanceR: ");
184.      Serial.println(distanceR);
185.      Serial.print("DistanceL: ");
186.      Serial.println(distanceL);
187.      delay(1000);
188.      */
189.    }
190.
191.    int threshold = 20;
192.    int threshold2 = 3;
193.    int idx = 0;
194.    int dlyF = 520;
195.    int dlyR = 200;
196.
197.    void leftWallFollower() {
198.    if (distanceL > threshold) {
199.    delay(dlyR);
200.    turnLeft();
201.    moveForward();
202.    delay(dlyF - 200);
203.    } else if (distanceN > 10) {
204.    moveForward();
205.    delay(50);
206.    } else if (distanceR > threshold) {
207.    delay(dlyR);
208.    turnRight();
209.    moveForward();
210.    delay(dlyF - 200);
211.    } else {
212.    turnBackward();
213.    }
214.    }
215.
216.    void loop() {
217.    if (mySerial.available()) {
218.    data = mySerial.readStringUntil('\n');
219.    Serial.println(data);
220.    }
```

```
221.    /*
222.      if (Serial.available()) {
223.      data = Serial.readStringUntil('\n');
224.      mySerial.println(data);
225.      }
226.      */
227.    plusUltra();
228.    //pid_calculation();
229.    //leftwall_follow();
230.    leftWallFollower();
231.    return;
232.
233.    if (idx > data.length() || data == "") return;
234.
235.
236.    int cnt = 0;
237.    if (distanceL <= threshold) cnt++;
238.    if (distanceR <= threshold) cnt++;
239.    if (distanceN <= threshold) cnt++;
240.    //Serial.print("cnt: ");
241.    //Serial.println(cnt);
242.
243.    int temp1 = leftPower;
244.    int temp2 = rightPower;
245.
246.    if (cnt >= 2 && distanceN > threshold) {
247.    if (distanceL <= threshold2) leftPower += 20;
248.    if (distanceR <= threshold2) rightPower += 20;
249.    moveForward();
250.    delay(50);
251.    leftPower = temp1;
252.    rightPower = temp2;
253.    /*
254.      if (distanceL > threshold) {
255.      //Serial.println("go fken left");
256.      turnLeft();
257.      moveForward();
258.      delay(dlyF);
259.      }
260.      else if (distanceR > threshold) {
261.      //Serial.println("go fken right");
262.      turnRight();
263.      moveForward();
264.      delay(dlyF);
265.      }
266.
```

```
267.      }
268.      */
269.    return;
270.
271.    }
272.
273.
274.
275.    if (data[idx] == 'U') {
276.    if (distanceL <= threshold2) leftPower += 20;
277.    if (distanceR <= threshold2) rightPower += 20;
278.    moveForward();
279.    if (idx == 0)
280.    delay(100);
281.    else
282.    delay(300);
283.    leftPower = temp1;
284.    rightPower = temp2;
285.    } else if (data[idx] == 'D') {
286.    turnBackward();
287.    } else if (data[idx] == 'R') {
288.    delay(90);
289.    turnRight();
290.    moveForward();
291.    delay(550);
292.    } else if (data[idx] == 'L') {
293.    delay(90);
294.    turnLeft();
295.    moveForward();
296.    delay(550);
297.    }
298.
299.    leftPower = temp1;
300.    rightPower = temp2;
301.
302.    if (idx++ > data.length()) Serial.end();
303.
304.    }
305.    float error = 0;
306.    //PID parameters
307.    float integral = 0;
308.    float derivative = 0;
309.    float last_error = 0;
310.    float PID_value = 0;
311.    //tuning these parameters is pain in the ass.
```

```
312.    //probably could use some optimisation to reach best
  option but too much work
313.    float Kp = 0.7;
314.    float Ki = 0.005;
315.    float Kd = 1.3;
316.    //These correction speed are purely experimental
317.    //Needs to manually checked to get the appropriate
  correction speed
318.    int correctionSpeed = 54;
319.    int base_speedLeft = 100;
320.    int base_speedRight = 78;
321.
322.    int choose_setpoint = 10;
323.
324.    void pid_calculation() // does the pid calculation,
  setpoint can be changed either for left or right sensor
325.    {
326.    //taking the difference in reading
327.    int setpoint = choose_setpoint;
328.    //IrDistance could be left or right depending on the
  wall follwing algorithm..
329.    error = distanceL - setpoint;
330.    //calculating integral
331.    if (abs(error) < threshold)
332.    {
333.    integral = integral + error;
334.    }
335.    else
336.    {
337.    integral = 0;
338.    }
339.    //taking the change in difference in reading
340.    derivative = error - last_error;
341.    last_error = error;
342.    //calulating PID value which is also an error value
343.    PID_value = int(error * Kp + integral * Ki +
  derivative * Kd);
344.    //return PID_value;
345.    // Serial.println(PID_value);
346.    //delay(1000);
347.    }
348.
349.
350.
351.    void leftwall_follow() {
```

```
352.    //sets the PID_value to be lowest to -45 and highest
  to 45
353.    if (PID_value < -correctionSpeed) {
354.    PID_value = -correctionSpeed;
355.    //Serial.println(PID_value);
356.    //delay(1000);
357.    }
358.    if (PID_value > correctionSpeed) {
359.    PID_value = correctionSpeed;
360.    //Serial.println(PID_value);
361.    // delay(1000);
362.    }
363.
364.    PID_value += 7;
365.
366.
367.    Serial.print("distance:");
368.    Serial.println(distanceL);
369.    Serial.print("value:");
370.    Serial.println(PID_value);
371.
372.    leftPower = base_speedLeft - PID_value;
373.    rightPower = base_speedRight - PID_value;
374.
375.    //moveForward();
376.
377.    //sharp right turn(90 degree turn) when theres a
  wall on the left and wall at the front as well
378.
379.    if (distanceN <= threshold && distanceL <= threshold
  && distanceR > threshold)
380.    turnRight();
381.    else if (distanceN <= threshold && distanceL <=
  threshold && distanceR <= threshold)
382.    turnBackward();
383.    }
```

7.3 Media and Design

For more media, seeing our robot attempts and designs visit

https://drive.google.com/drive/folders/1ERo0OfPcFGQ9jkGrFE-2-J-Rt2eJJtnp?usp=sharing