

# FIRST YEAR IIT- DELHI MESHMERISE CONTEST DOCUMENTATION

**DECEMBER 2018**

---

COMPETITION: IIT BOMBAY TECHFEST  
MESHMERISE



## **PARTICIPANTS**

Neelabh Madan  
Mudit Garg  
Varsha Goalla  
Hima Dhruti

# CONTENTS

INTRODUCTION .....	4
ABOUT MESHMERISE.....	5
OUTLINE .....	5
ARENA.....	5
GAME PLAN .....	5
SAMPLE MAZE .....	6
BOT DESIGNING .....	7
ELECTRICAL DESIGN .....	8
INTIAL THOUGHTS AND APPROACHES .....	9
MAZE SCANNING .....	9
MAZE SOLVING.....	10
FINAL DECISION.....	11
COMPONENTS USED.....	12
APPROACH TOWARDS THE PROBLEM.....	13
DETECTING JUNCTION TYPES .....	14
CHALLENGE: DETECTING 45° JUNCTIONS WITH ONLY 1 LSA08 SENSOR.....	15
MAPPING THE MAZE.....	17
FINDING SHORTEST PATH.....	19
TACKLING CLOSED LOOP SITUATIONS.....	20
OPTIMISING .....	21
PROBLEMS FACED .....	22
PROBLEMS FACED AT TECHFEST-IIT BOMBAY .....	23

IMPROVEMENTS THAT CAN BE FURTHER DONE .....	24
ACKNOWLEDGEMENTS .....	25
REFERENCES.....	26

# INTRODUCTION

As part of the robotics club. We four freshers worked on the problem statement of IIT Bombay's TechFest's Meshmerise Competition. The competition was scheduled for December 2018. With about 1-1.5 months work and under Pranav Nagpure's mentorship we were ready with our maze solving bot.

# ABOUT MESHMERISE

## OUTLINE

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 its path in the dry run and has to go through the maze from the starting point to the ending point in shortest path in the actual run. Throughout the maze the bot must remember the junction it visited and its type(45 degree turns included). Also the distance between two such junctions was quantized. A random maze is given below.

## ARENA

Game field consists of an arena of dimensions 224cmx194cm(lxb). It is made of white vinyl strips of width 30mm. Angle between adjacent white lines in the path is 90deg or 135deg. White box is of dimensions 30cmx30cm.

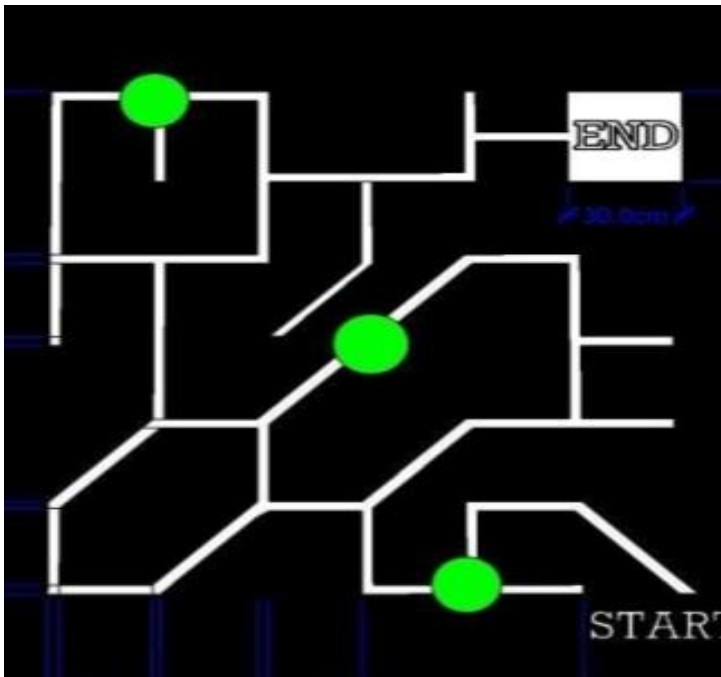
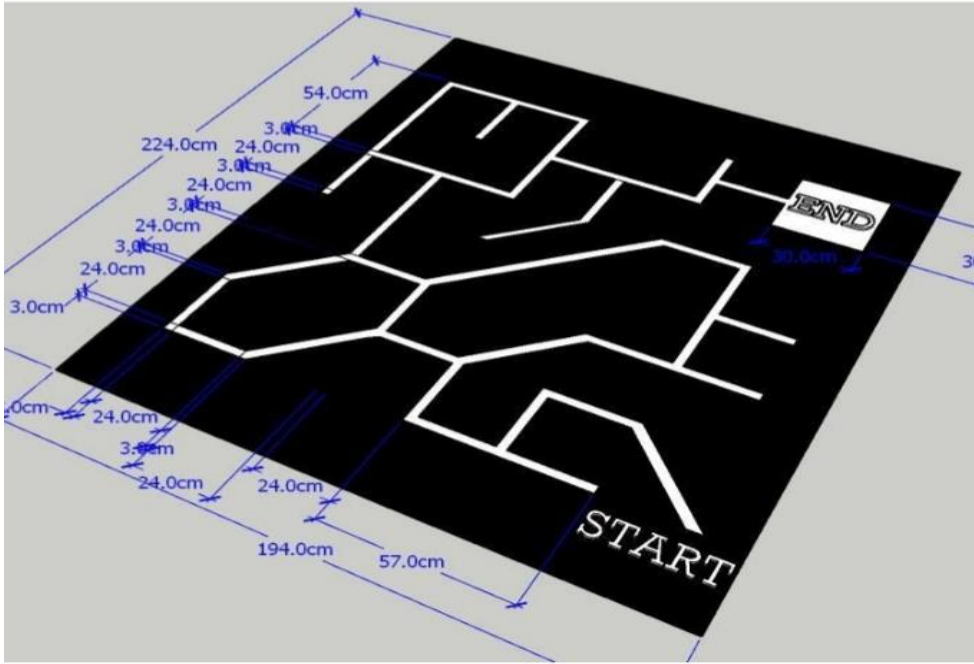
## GAME PLAN

First part is DRY RUN. The bot must start from START and find its way to END.

The bot has to give signal by glowing LED as soon as it senses white box. It has to follow a algorithm to find the path and store it. There is no restriction to cover all checkpoints. Second part is ACTUAL RUN(2.5 min). It has to start from START and reach END by the best possible path stored in memory.

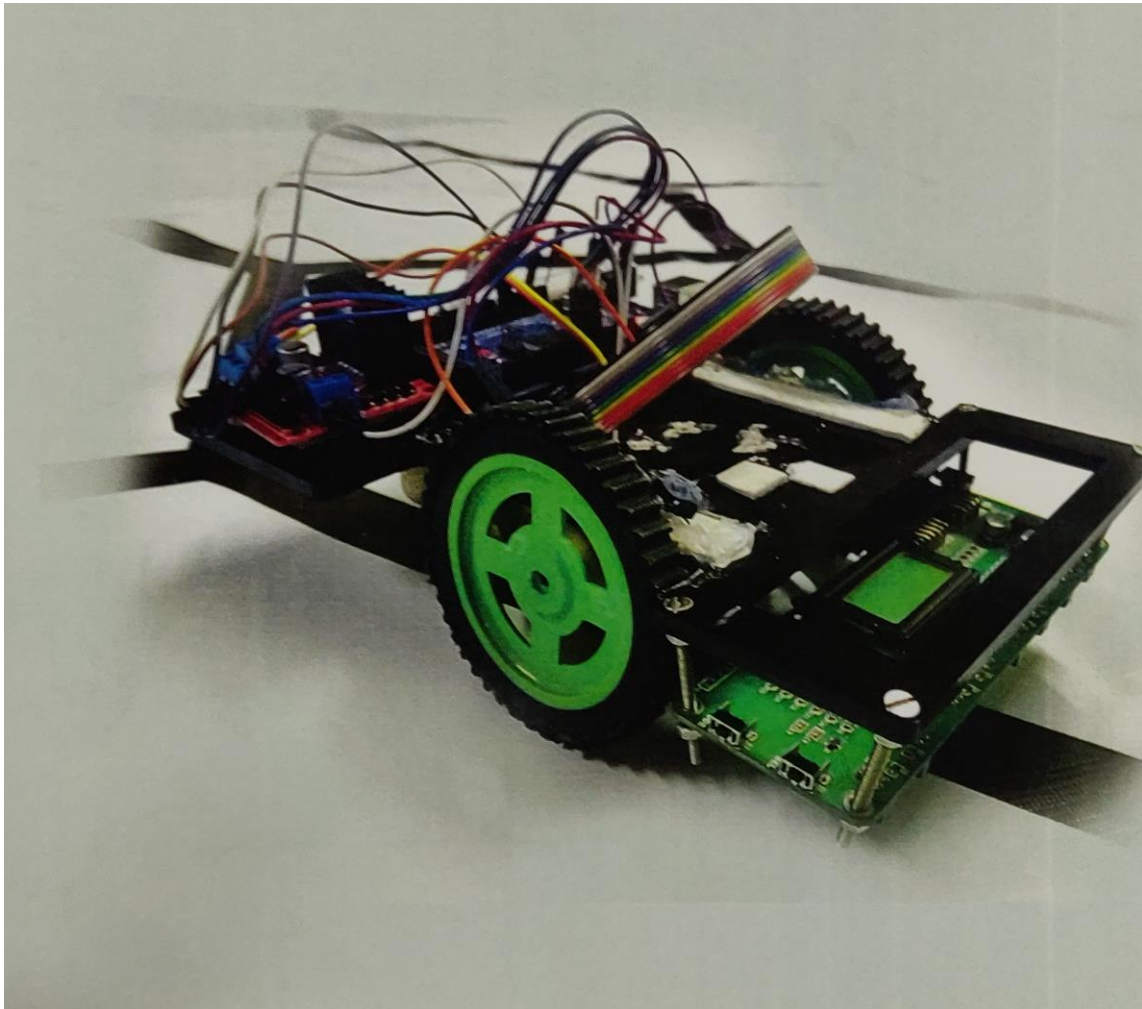
DRY RUN is given 3 minutes. Passing through flag has 50 points each. Also  $330 - (\text{time taken in sec})$  points will be awarded if completed in 5.5 min.

# SAMPLE MAZE

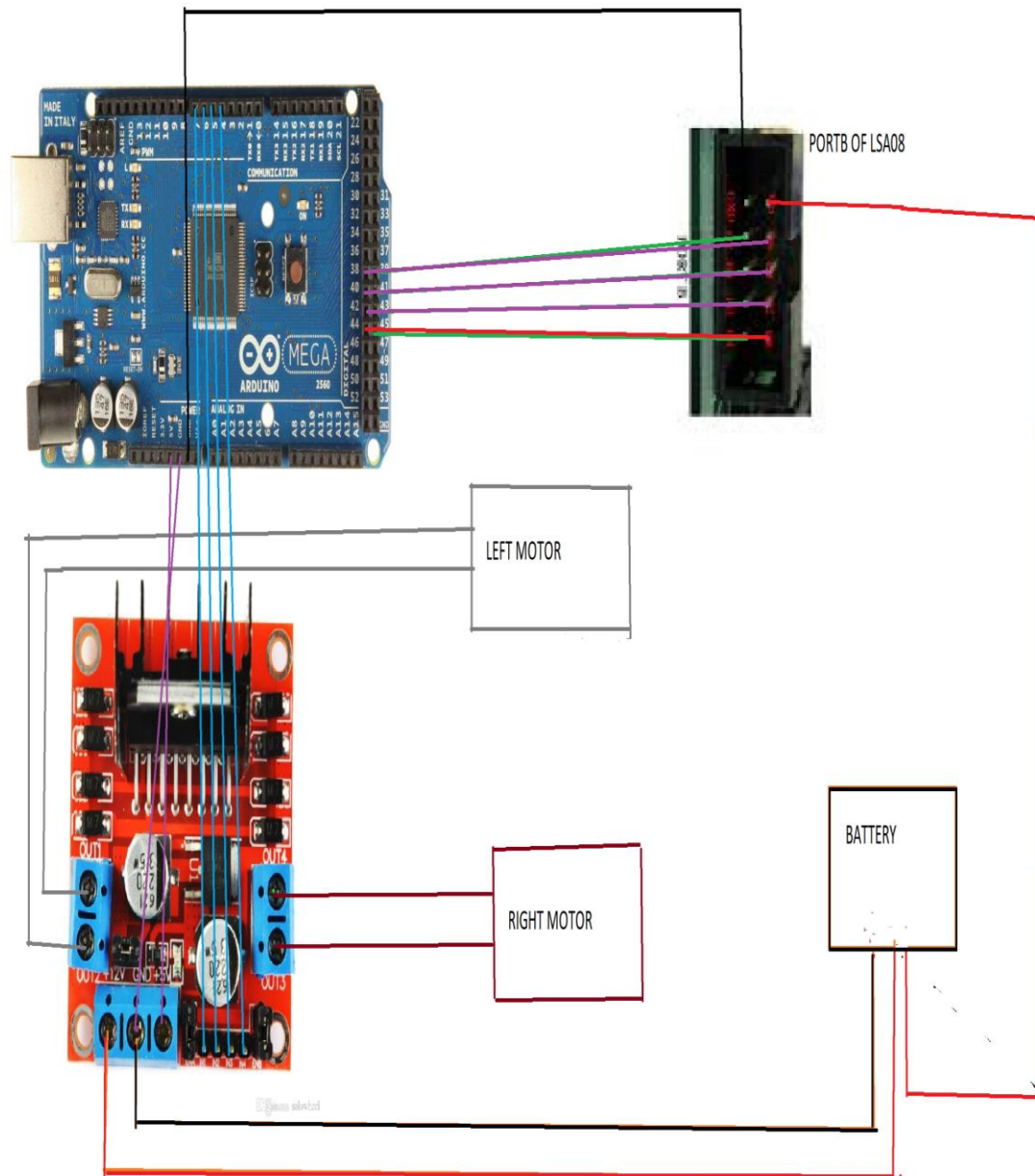


# BOT DESIGNING

We believed that the competition was not much dependent on the robot as compared to the code that goes into it. So we used the bot that was given to us in our Robotics workshop. Though we made some changes such as adding rubber grips on the wheel so that they don't slip when conditioned to a higher RPM.



# ELECTRICAL DESIGN





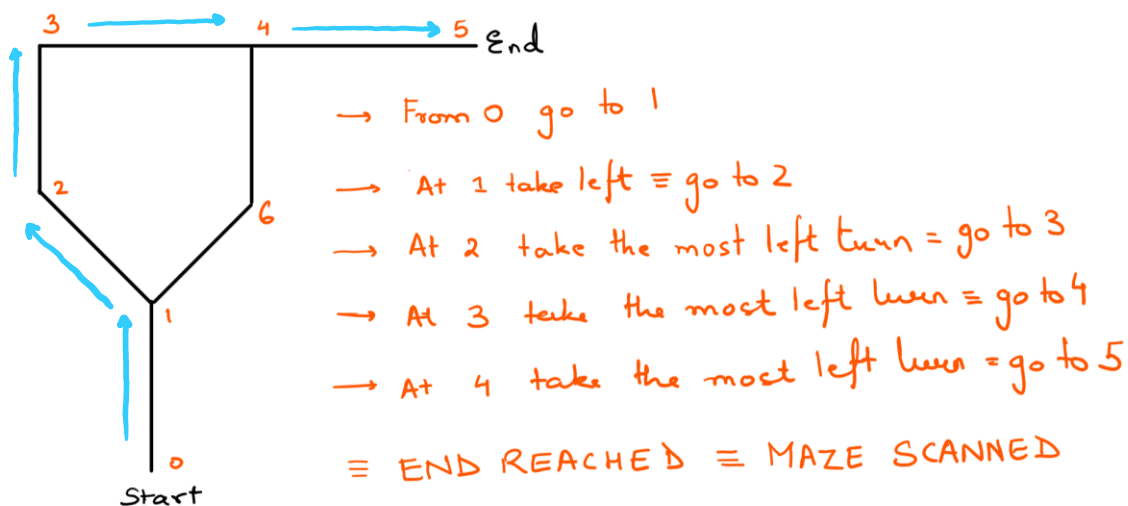
# INITIAL THOUGHTS AND APPROACHES

Assuming our hardware works perfectly well, the problem statement has 2 parts:

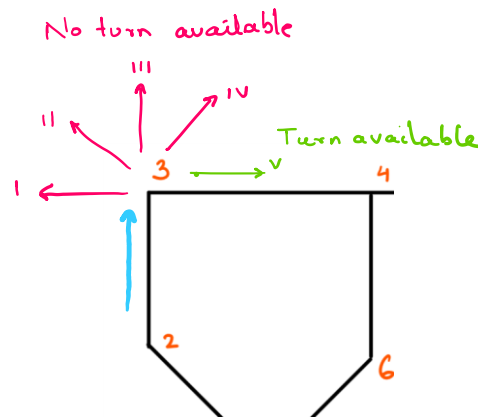
1. Maze Scanning
2. Maze Solving

## MAZE SCANNING

For maze scanning we started off with the most basic maze scanning algorithm called the always left/right algorithm. In this algorithm we take a left turn as we approach any junction. This algorithm is bound to scan the whole maze except when we encounter some tricky loops. Ex is given below

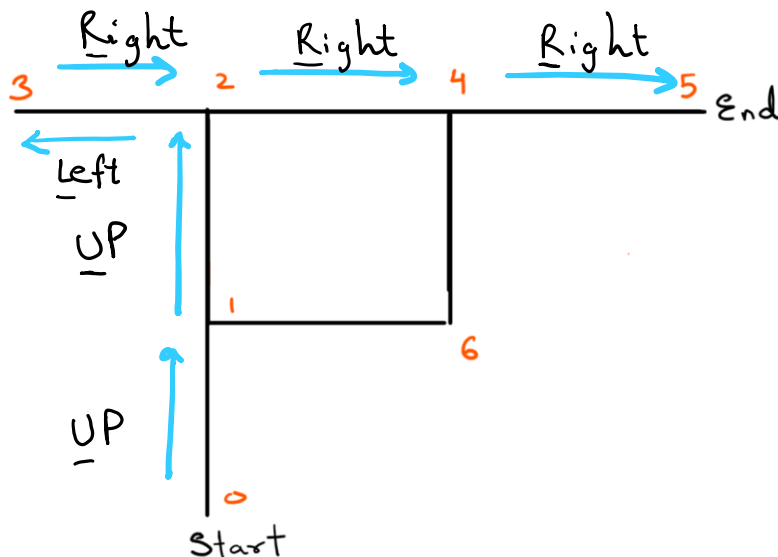


NOTE: A very important thing is to understand what always left means. Consider junction 3. When we approach this junction from junction 2, we move on to junction 4 because a left turn means scanning from the left till we find a turn and then take that turn. Scanning from the left path to junction 4 is our first left path.



## MAZE SOLVING

For the maze solving we went with the turn reduction method initially. It tells the shortest path from the path that we have travelled by eliminating unnecessary turns. Consider the example where we maintain a sequence of turns that we take.

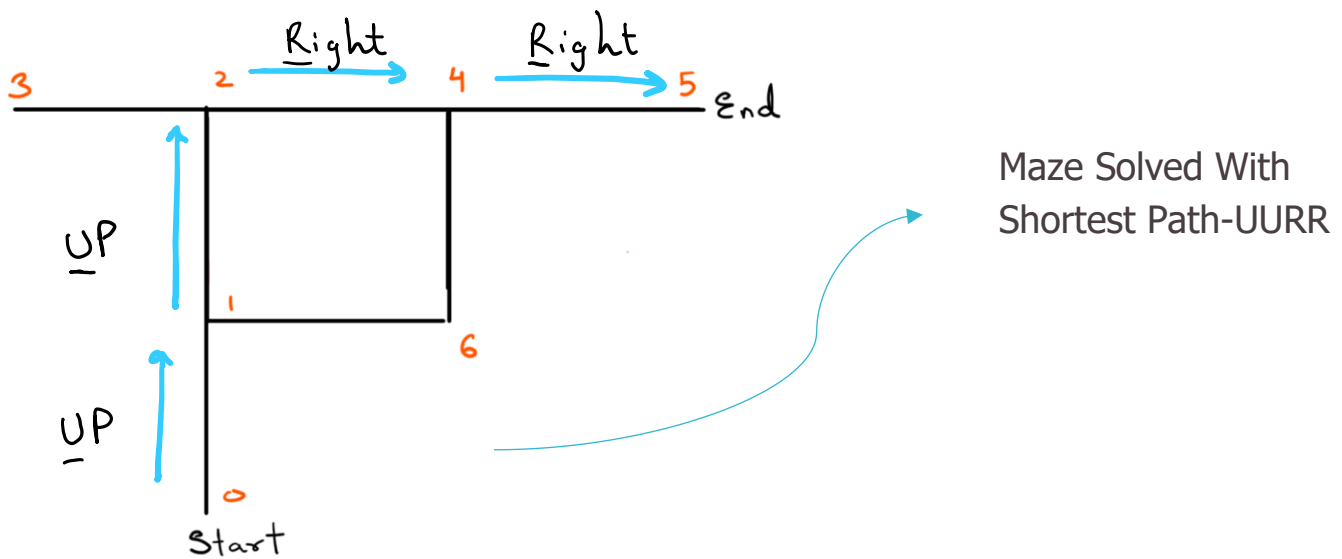


1-U  
2-U  
3-UUL  
2-UULR  
4-UULRR  
5-UULRRR

Unnecessary  
sequence

≡ Remove it

Final Sequence to scan  
maze=UULRRR  
Final Sequence to solve  
maze= UULRR



These reductions contain LRX to X, UDX to X, LUDRX to LRX then to X and more.

### FINAL DECISION

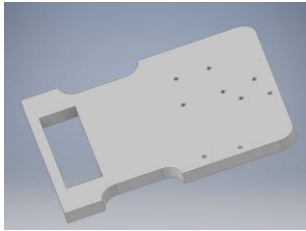



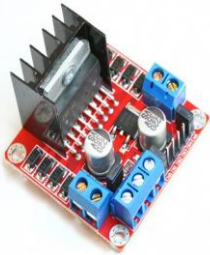


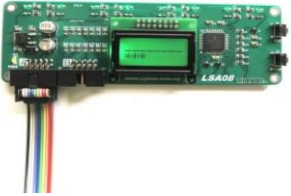
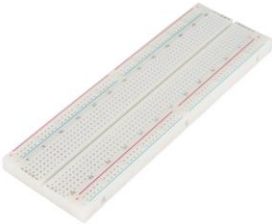



We kept the maze scanning algorithm as it was the most trivial way to scan the maze.

But the maze solving algorithm was not fit for the problem statement. The reduction algorithm works only when the distance between 2 junctions is constant. Drawing conclusion we moved on towards an Object Oriented Programming approach which will be discussed ahead.

# COMPONENTS USED

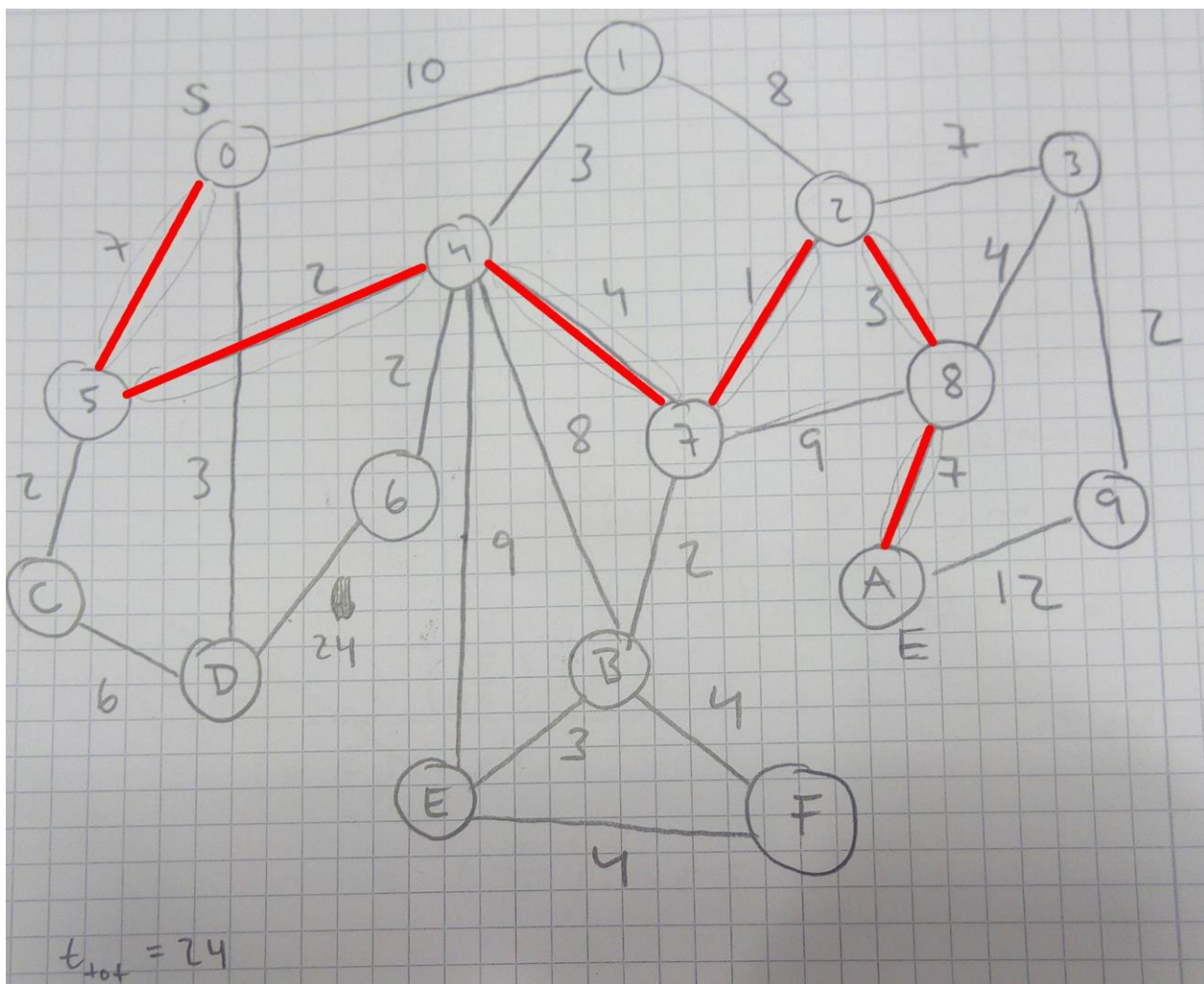
Our bot has dimensions around 21cmx19cm. It contains a switch. It has a red LED which glows at end.

Materials used:

Chassis 	Driving Wheels 	Arduino Mega 2560 	Castor Wheel 
L298 Motor Driver 	300 RPM BO Motor 	LiPo 11.1V Battery 	LSA08 sensor 
Breadboard 	Switch 	Jumper Wires 	Nuts and Bolts 

# APPROACH TOWARDS THE PROBLEM

We found out a solution by considering each junction as a node. With nearby nodes connected to each other the structure we got was of a graph. Then we used Dijkstra and flood fill algorithm to find the shortest path. We also came up with a nice little algorithm to tackle closed loops in the problem statement.



A sample implementation of Dijkstra's algorithm

# DETECTING JUNCTION TYPES

The following are the 12 types of junctions

Type-1



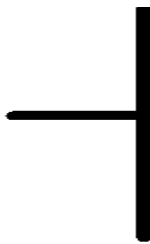
Type-2



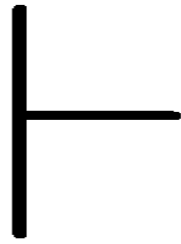
Type-3



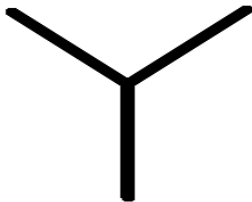
Type-5



Type-6



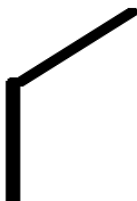
Type-4



Type-7



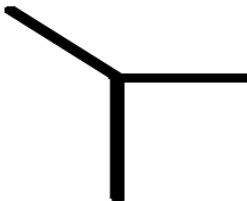
Type-8



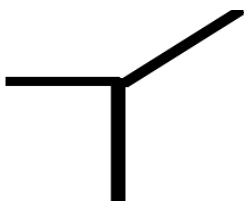
Type-9



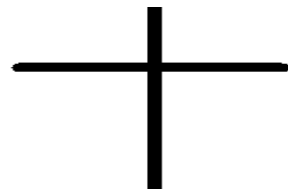
Type-10



Type-11



Type-12



# CHALLENGE: DETECTING 45° JUNCTIONS WITH ONLY 1 LSA08 SENSOR

Determining the nature of a junction was not easy with only 1 LSA08 sensor due to the fact that it has 8 linearly aligned sensors.

In order to overcome this problem we went with a little tricky approach.

We took a set of readings from the sensor while the bot would be traveling over the junction. This was a very small time period in which it would cover not more than 5 cm This generated a matrix of readings to be processed.

We further pattern matched the readings to what we would expect the readings to be.

Lets see one such example.

Stopped after moving a little ahead

Reading=[0,0,0,0,0,0,1,1]

T3



T2



Junction Detected==Moving a little ahead

Reading=[1,1,1,1,1,1,0,0]

T1



No junction detected==  
Moving straight  
Reading=[0,0,0,1,1,0,0,0]

Matrix generated in between the state T2 and T3 is

[1,1,1,1,1,1,0,0]  
[1,1,1,1,1,1,0,0]  
[1,0,0,1,1,1,0,0]  
[0,0,0,1,1,1,0,0]  
[0,0,0,0,1,1,0,0]  
[0,0,0,0,0,1,1,0]  
[0,0,0,0,0,1,1,0]  
[0,0,0,0,0,1,1,1]  
[0,0,0,0,0,0,1,1]  
[0,0,0,0,0,0,1,1]  
[0,0,0,0,0,0,1,1]

This matrix shows the behavior of the junction as junction type 11 so we store it as junction 11



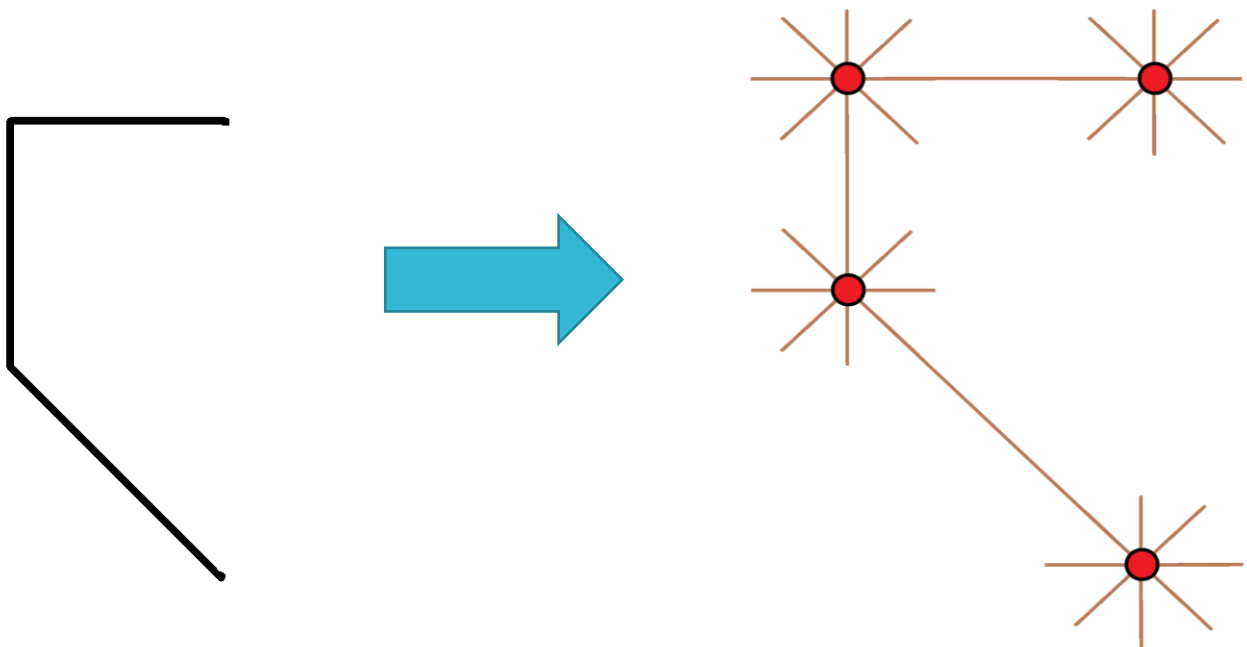
# MAPPING THE MAZE

For this we took the help of object oriented programming from C++. We made each node an object with certain properties and paths that will be shortly discussed. This helped us generate a graph like mapping of the maze in which we could incorporate the maze solving algorithms.

Initially we were afraid that all this data couldn't be saved on the Arduino board but we were quite wrong. It was easily able to save such data and solve it in no time.

To start of with the mapping we made our own rectangular coordinate system with quantized lengths between any 2 points. We used the time taken to travel between two nodes as the distance between two nodes.

Following is the implementation of our nodes and paths





## NODE

This is a node object that has connections to eight paths and has a identity depending upon the type of junction this node this is and its coordinates.



## PATH

This is a path object that has 2 end points on which it saves the two connecting nodes. It also contains the important data such as is it being traversed by the bot for the first time, length of the path, is it blocked(in case of obstacles),etc.

We also maintained a bot orientation which was changed according to the turns it took. Also we maintained a global orientation i.e. the orientation of the bot when we start.

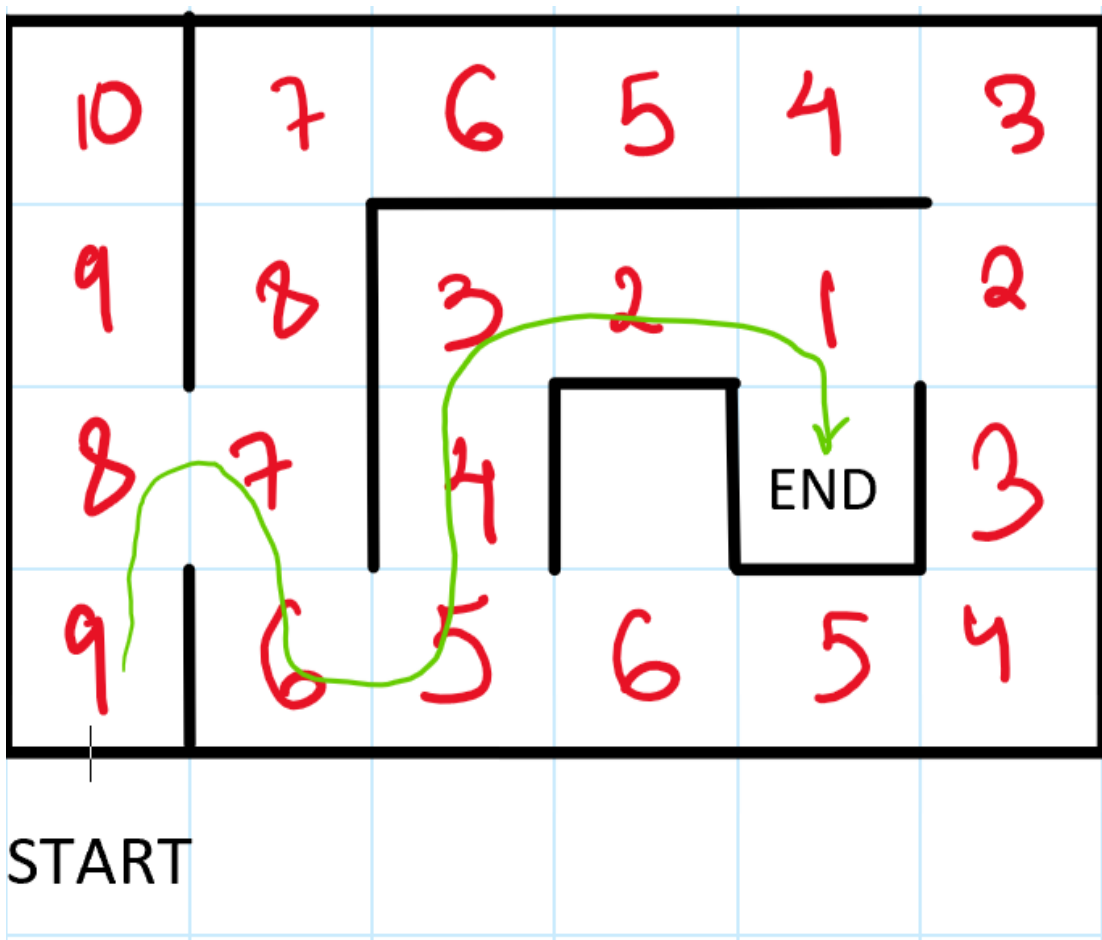
# FINDING SHORTEST PATH

With our own coordinate system we were able to apply a mix of Flood-Fill and Dijkstra's algorithm to solve the maze.

The rectangular coordinate system plays an important role as it enables us to make use of the Flood-Fill algorithm

We ranked each node with the help of the distance and Flood-Fill then applied Dijkstra's algorithm to find the shortest path.

This newly generated shortest path was then given to the bot and it would follow the path and complete the task.



# TACKLING CLOSED LOOP SITUATIONS

A lot of times we would encounter mazes which had closed loops, ie they were built such that our only left taking algorithm would fail miserably in solving such mazes. Our bot would continue to go round and round that loop. If this happens then we cannot even scan the maze properly.

To tackle this we came up with a smart algorithm with the help of stacks. Whenever a junction with 3 or more exit points was scanned we would put it in a stack. Since the bot had to eventually return back to that junction, when it returned we would remove it from the stack and deal with the rest of the stack.

Now at any point during the run if our bot has nowhere to go, we would make it go to the topmost element in the stack. This ensured that we wouldn't be stuck in closed loops and will traverse the whole maze.

# OPTIMISING

With our theoretical aspect clear we went ahead with a competitive mind.

To win the competition we required speed. That's why we went on with increasing the speed of the bot, checking its stability with each iteration, perfecting the turns.

We connected our motors to the chassis more firmly and increased the stability of the castor when so as to maintain a constant load on the motors for predictable results.

We also tried including extra switches so that we could change the always left to always right algorithm as soon as we will see the arena so as to maximize our points. But later it was told that flags equally favored always left and always right follower algorithms.

# PROBLEMS FACED

There were some issues that were keeping us from performing at our maximum potential.

Our BO motors were not of good quality and had different RPMS. Thus we had to hardcode our turns and PID control again and again. The difference in their RPMs was also dependent on the voltage that the battery could provide. So after fine tuning the bot for 5 minutes it would work correctly one time and then again it would not move as it should.

Also sometimes our Arduino wouldn't connect with the laptop leading large chunks of time wasted in diagnosing that issue.

Plus sometimes the bot wouldn't read the junction correctly leading to very unwanted behavior by the bot.

# PROBLEMS FACED AT TECHFEST-IIT BOMBAY

Our robot was working good when we tested in college. But on the day of competition motor driver of our robot did not work. So we took a motor driver from other team and used it. Then our robot was not moving properly. Then we changed our slot timing and tried to change the speeds so that it moves properly. Then it was fine. Our robot completed dry run successfully. It covered whole maze. But after dry run when we kept our robot at start position it started to revolve and went out of the maze. So robot did not complete the actual run.

# IMPROVEMENTS THAT CAN BE FURTHER DONE

We believe that we could improve on the following 3 fronts:

1. Using more than 1 linear sensors to analyze the junctions. Or maybe even make our own system using elementary IR sensors to suit the problem statement
2. Using better motors would have been much better due to their guaranteed performance.
3. Using wider wheel base.(Learnt from other competitors at TECHFEST). It provides very high stability and good control.



# ACKNOWLEDGEMENTS

We would like to express our profound gratitude and deep regards to our mentor Pranav Nagpure and Robotics Club for their constant support and valuable inputs and encouragement.

We would also like to thank Professor Subir Kumar Saha, who gave us the opportunity to work for this competition, which gave us a platform to think differently and helped us in doing a lot of research and explore new areas because of which we came to know about many new things.

We are also thankful to our Coordinators Pranav Nagpure and Dinesh Singamsetty for constantly guiding us to create the working prototypes.

# REFERENCES

<https://techfest.org/Meshmerize/competition>

[https://en.wikipedia.org/wiki/Flood\\_fill](https://en.wikipedia.org/wiki/Flood_fill)

<https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7>