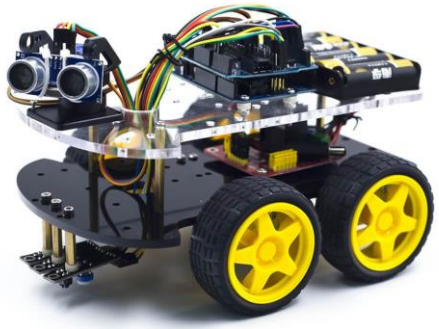
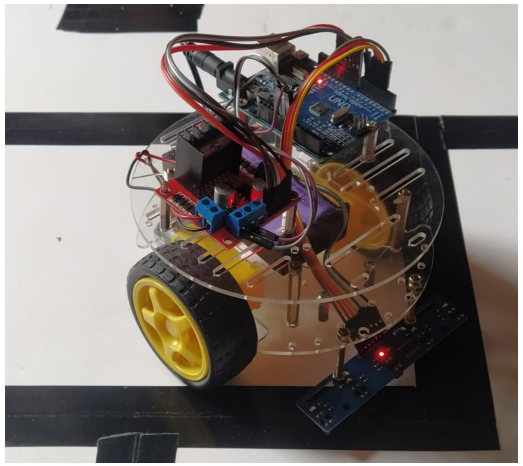
NSDC – Junior Skills Championship Mobile robotics



N·S·D·C
National
Skill Development
Corporation

HOLOWORLD

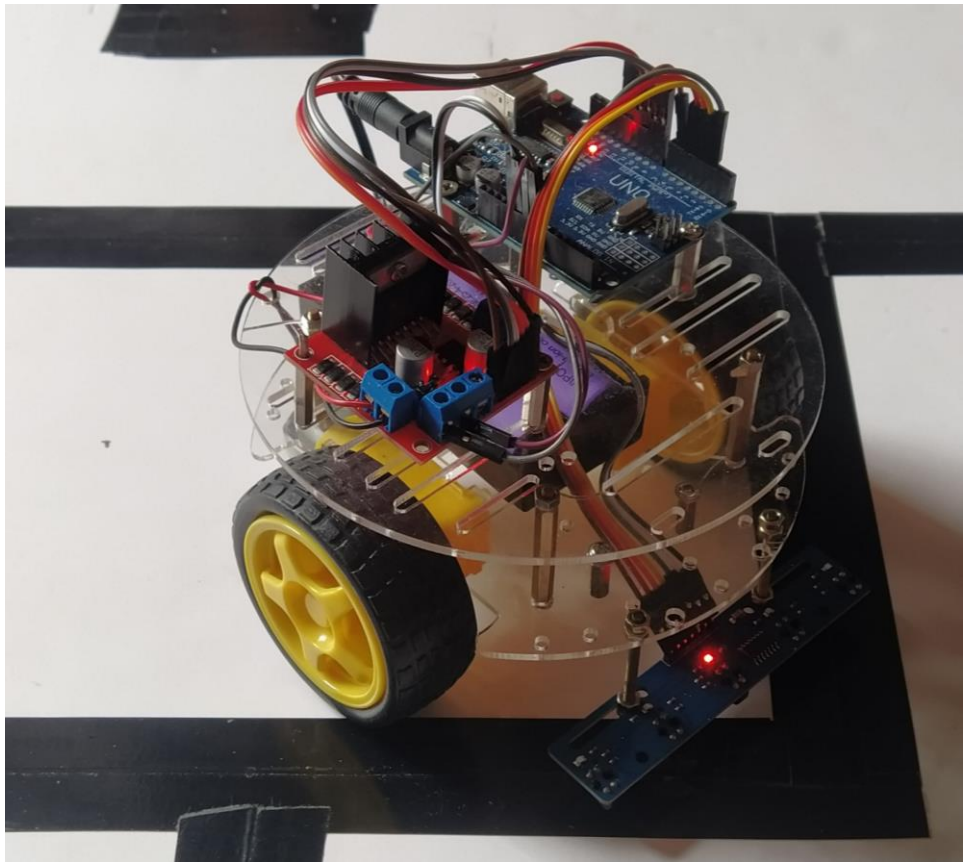


Wednesday, December 15, 2021

Agenda – Day 3

Maze solving Robot

- What's Maze solving robot
- Working principle
- Components
- Schematic and connection
- Let's code!



HOLOWORLD

Wednesday, December 15, 2021

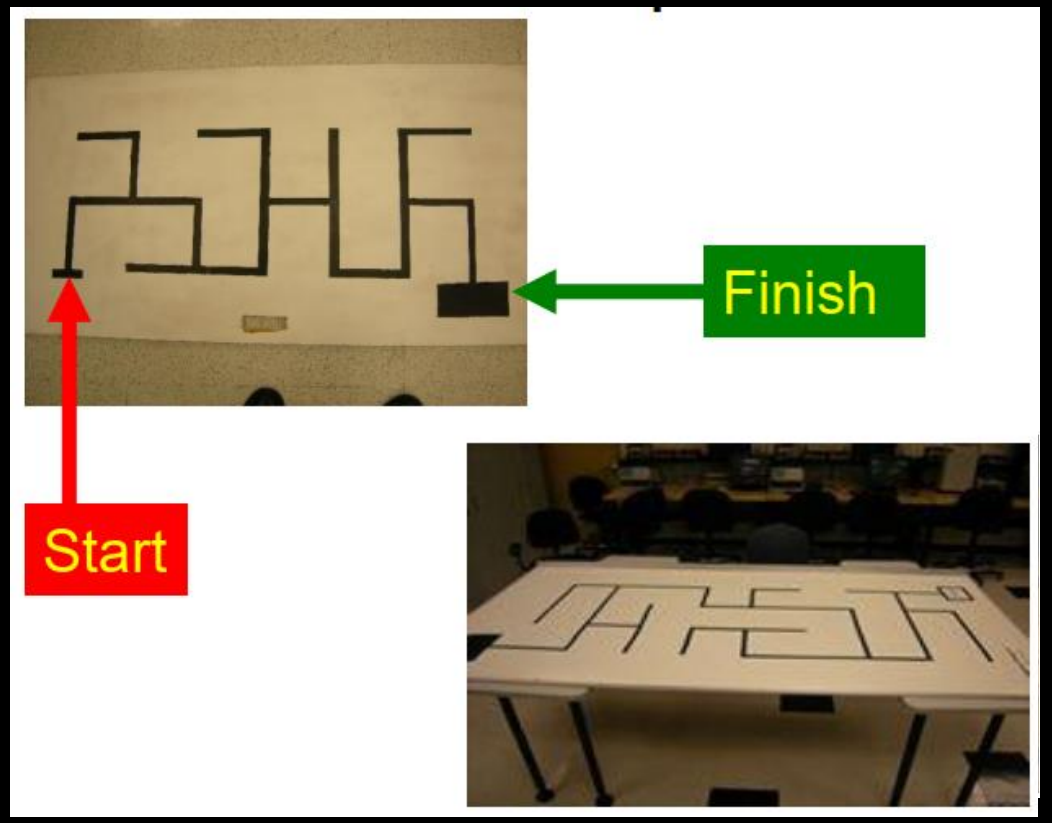
Maze solving robot

- The maze solving robot is designed to find a path without any assistance or help.
- As a type of autonomous robot, it has to decode the path on its own to solve the maze successfully.
- So, its logic is quite different from the line following robot which follows a predetermined route.
- These types of autonomous mobile robots can be used in a wide variety of applications such as material handling, warehouse management, pipe inspection, and bomb disposal

Line Maze robot

What is a line Maze?

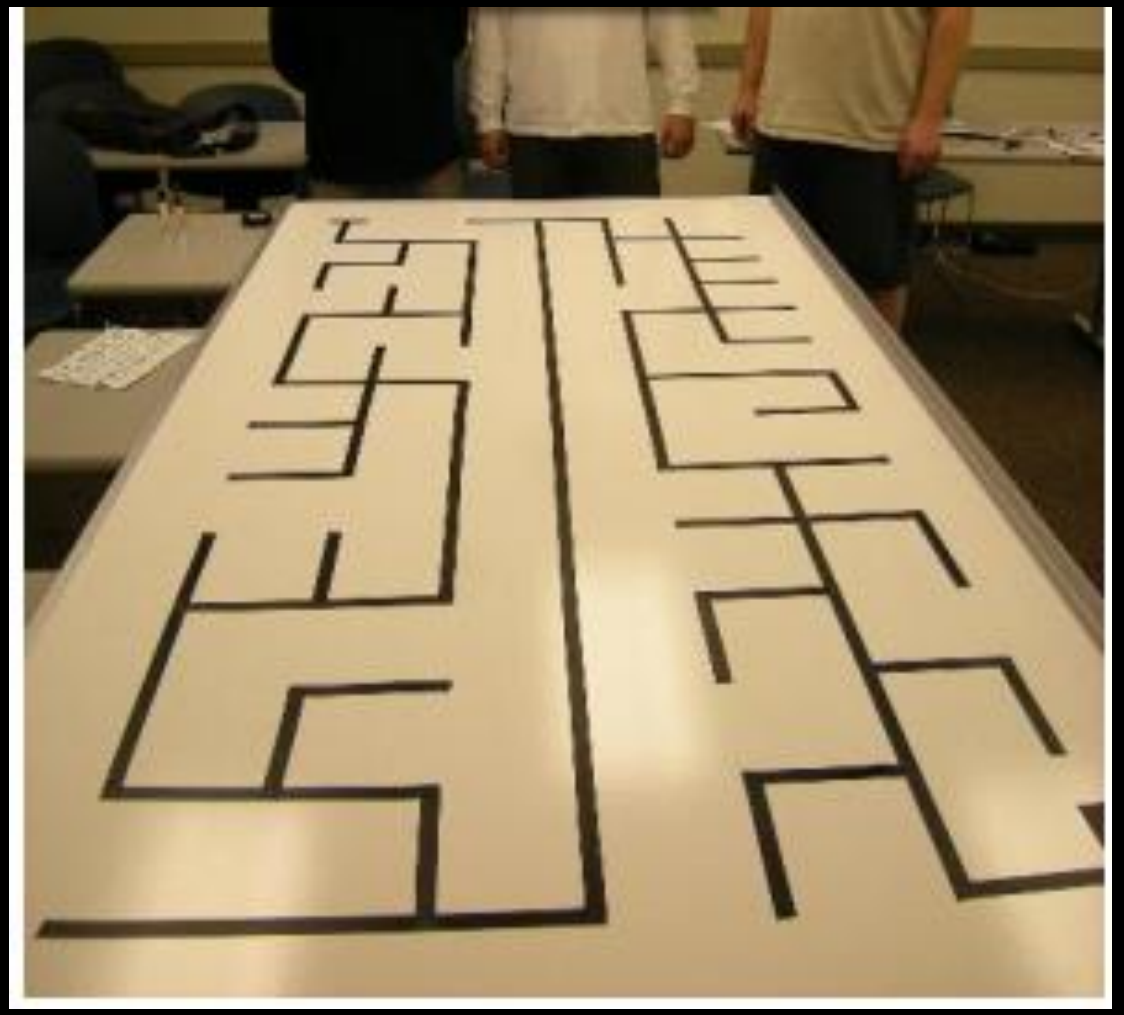
- A line maze is usually a black line on a white background. It could also be a white line on a black background, but for this concept its black lines on a white background will be used
- Each line maze has a start point and a finish point. The robot is expected to follow the lines and find it's way from start to finish in the fastest time possible



Line Maze robot

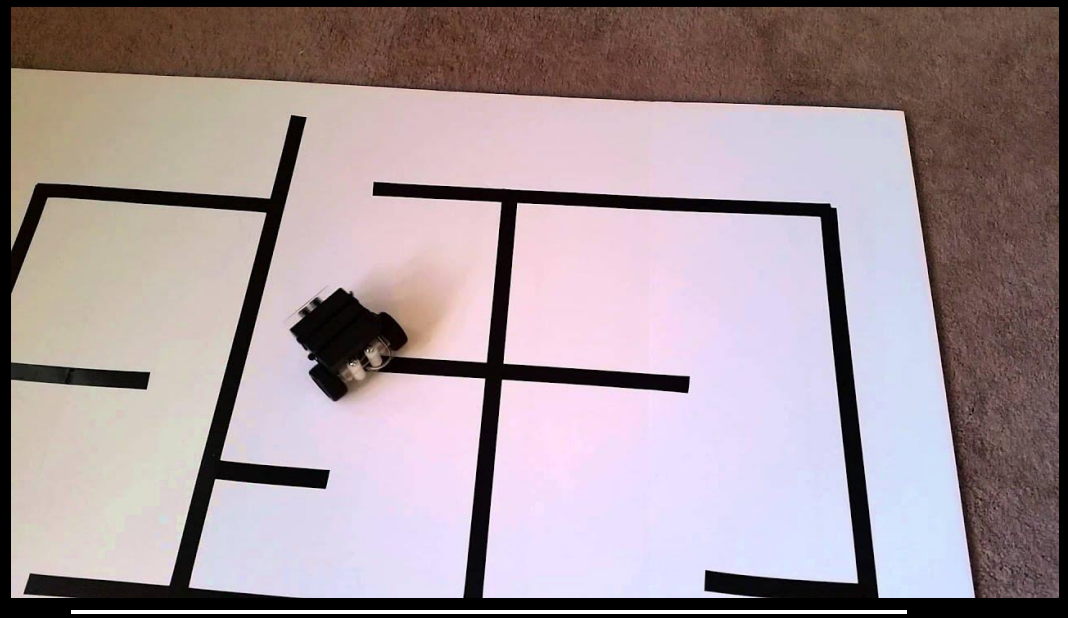
What is a line Maze?

- A Track was designed with several long straight paths to give an advantage to a robot that knows when it can increase speed
- Notice that there are a number of dead-end paths in the maze. The robot typically cannot traverse the maze without first taking several wrong turns



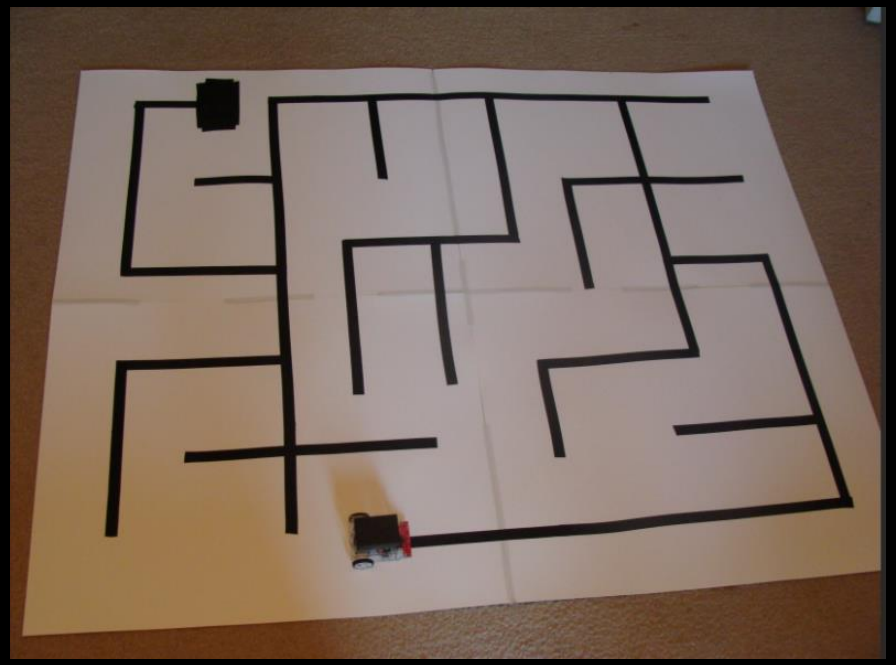
Line Maze robot

Algorithm - Maze solving robot



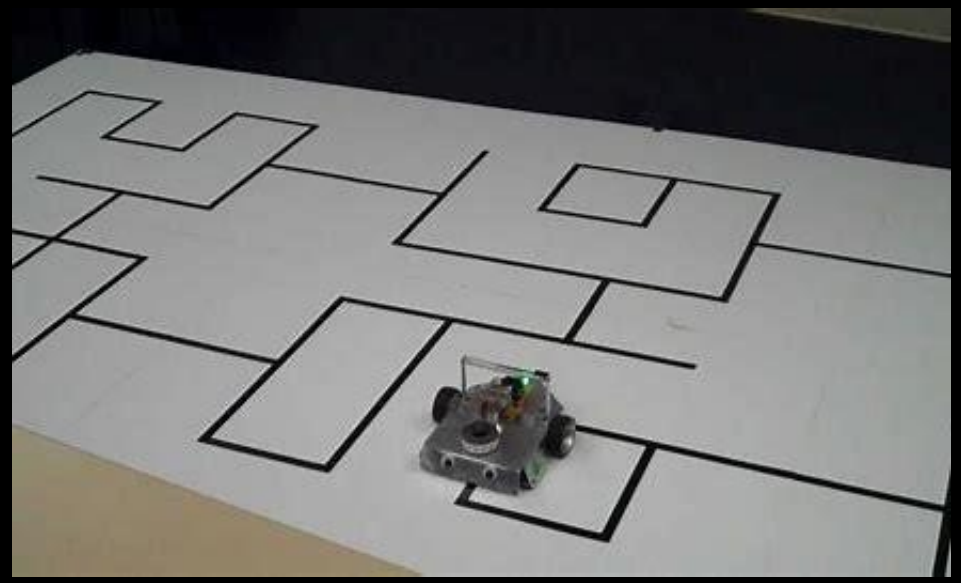
- The most efficient Maze solving algorithm is flood fill algorithm. It is derived from “Bellman Ford Algorithm
- The algorithm works by assigning value for all cells in the maze, where these values indicate the steps from any cell to the destination cell. The first array is holding the walls map values, while the other one is storing the distance values
- If we are given the goal coordinates by the user, we need to figure out how to get from S (Start) to F (Finish)

Algorithm - Maze solving robot



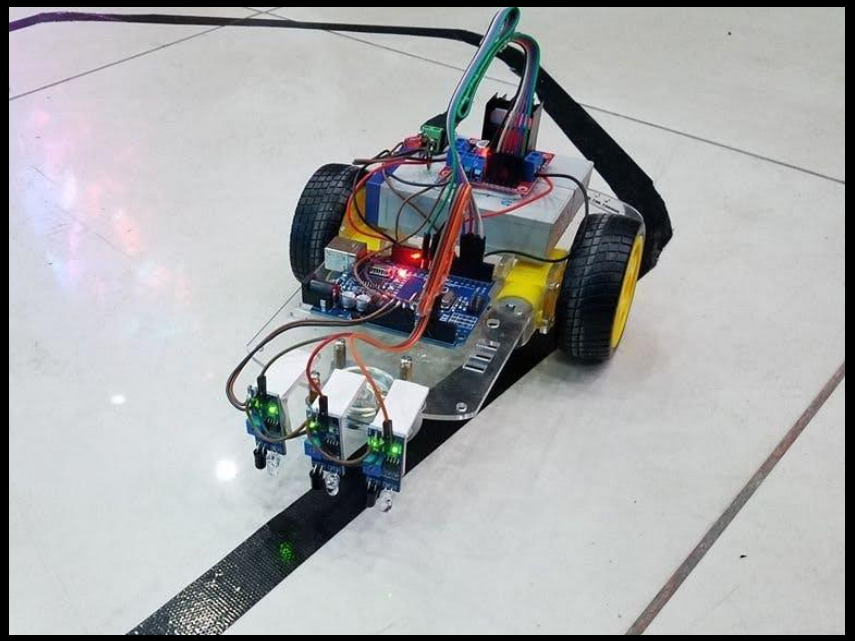
- An excellent method for this is a flood-fill algorithm for counting the distance steps of every maze square from the start
- Eventually, we reach the goal destination, which in this example is the bottom left to reach destination which is located in left top
- So, by now we know two things:
  - The goal position is reachable from the start, and
  - The shortest path from start to the goal has 40 steps
- In general, we do not know what the actual path is and the destination.

Working – Maze solving robot 3 IR Sensor



- The maze solving robot detects the track by using the IR sensor module and follow the robot in the line, until it finds a destination.
- The array of IR sensors has 3 or 5 IR sensors on the left side of the robot we have two IR sensors. Similarly on the right side we have two more and in centre we have one IR sensor.
- The robot is programmed to drive over the black lines of the maze and use optical sensors on the bottom of the robot to track the lines.
- As it travels along, the program we are using will solve the maze for the shortest path with a simple to understand method called the "Left Hand Rule" or sometimes called the "Left Hand on Wall" method.

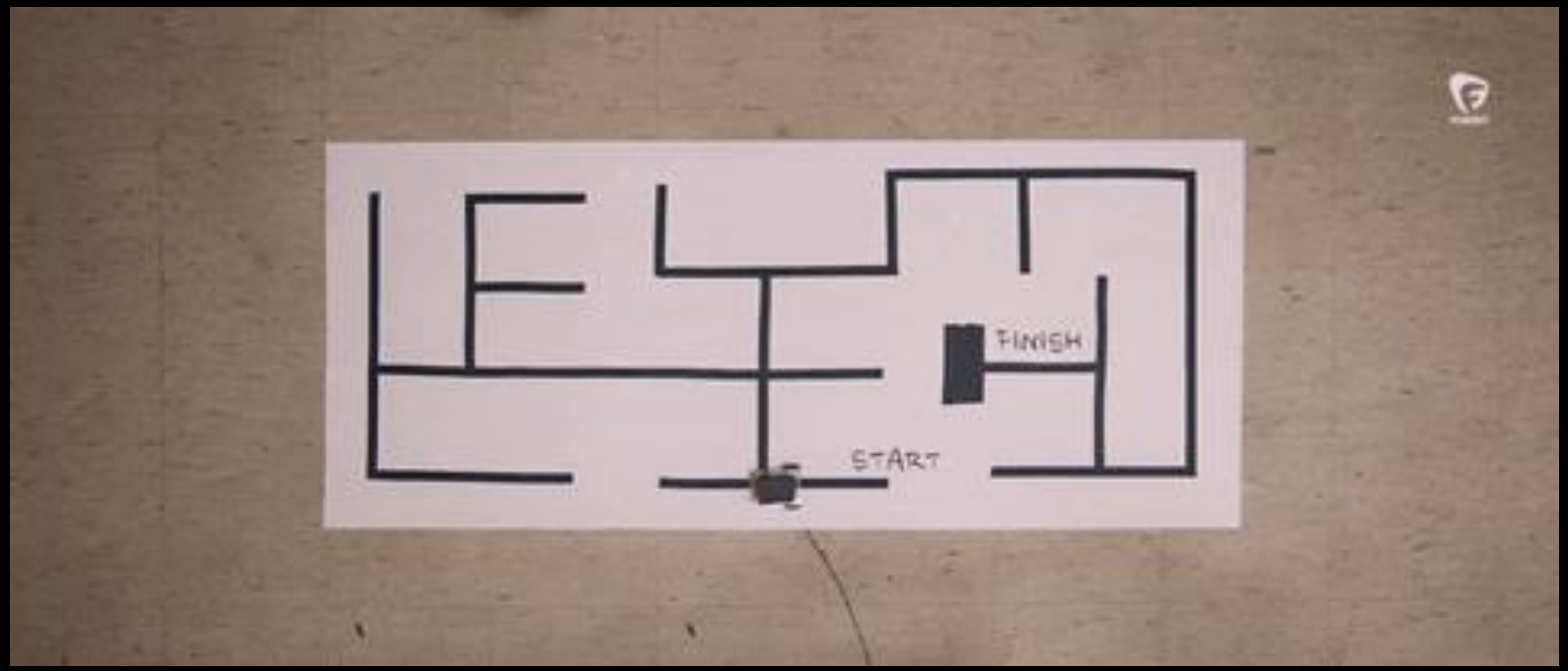
Concept of Line following robot



- The concept of the line follower robot is related to light. Here, we use the behavior of light on the black and white surface.
- The white color reflects all the light that falls on it, whereas the black color absorbs the light.
- In this line follower robot, we use IR transmitters and receivers (photodiodes).
- They are used to send and receive the lights. When IR rays fall on a white surface, it is reflected towards IR receiver, generating some voltage changes.
- Based on these inputs, an Arduino Uno provides the proper output to control the bot.

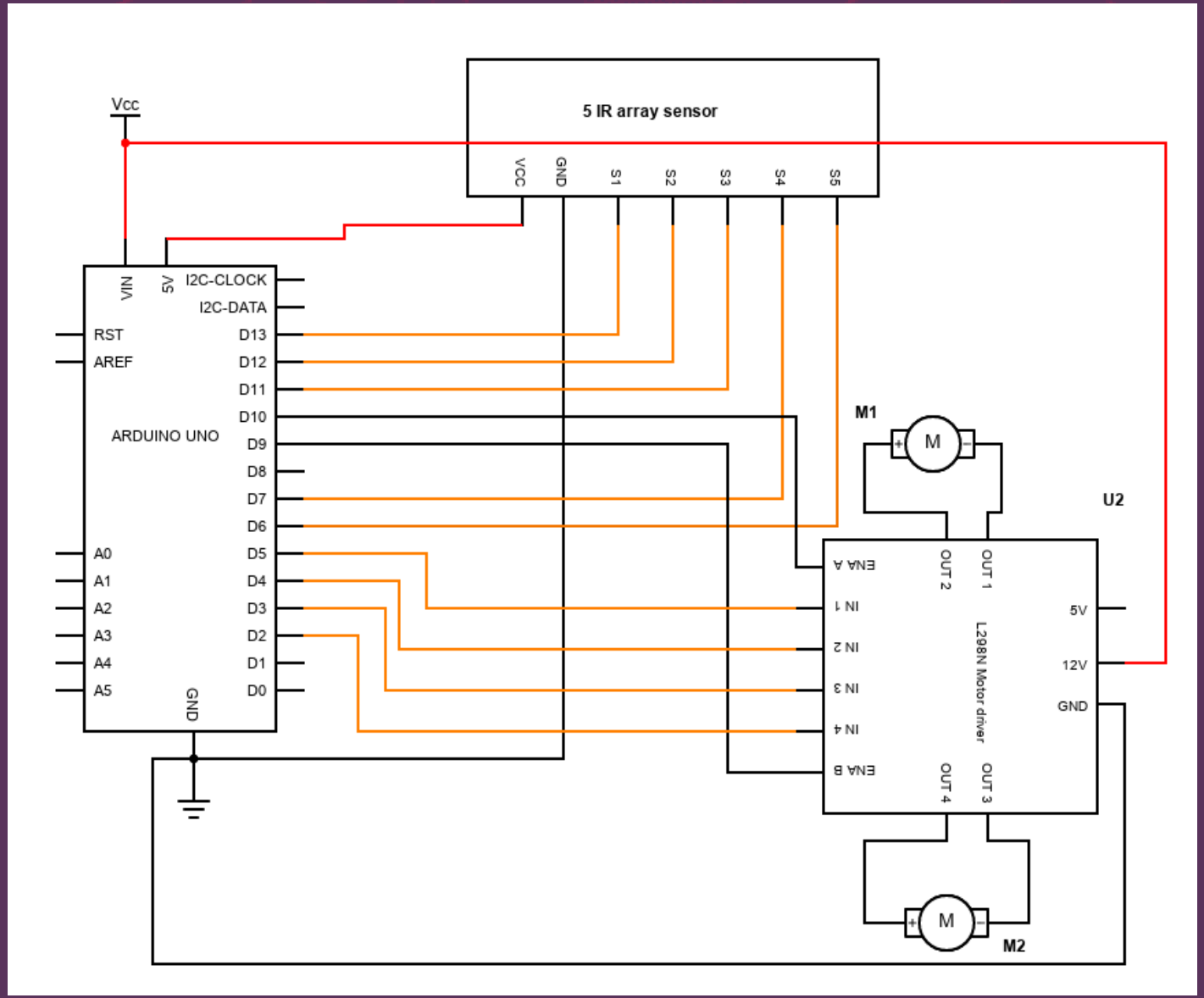
Concept of Line following robot

- When IR rays fall on a black surface, it is absorbed by the black surface, and no rays are reflected; thus, the IR receiver doesn’t receive any rays.
- When the IR sensor senses a white surface, an Arduino gets 1 (HIGH) as input, and when it senses a black line, an Arduino gets 0 (LOW) as input.

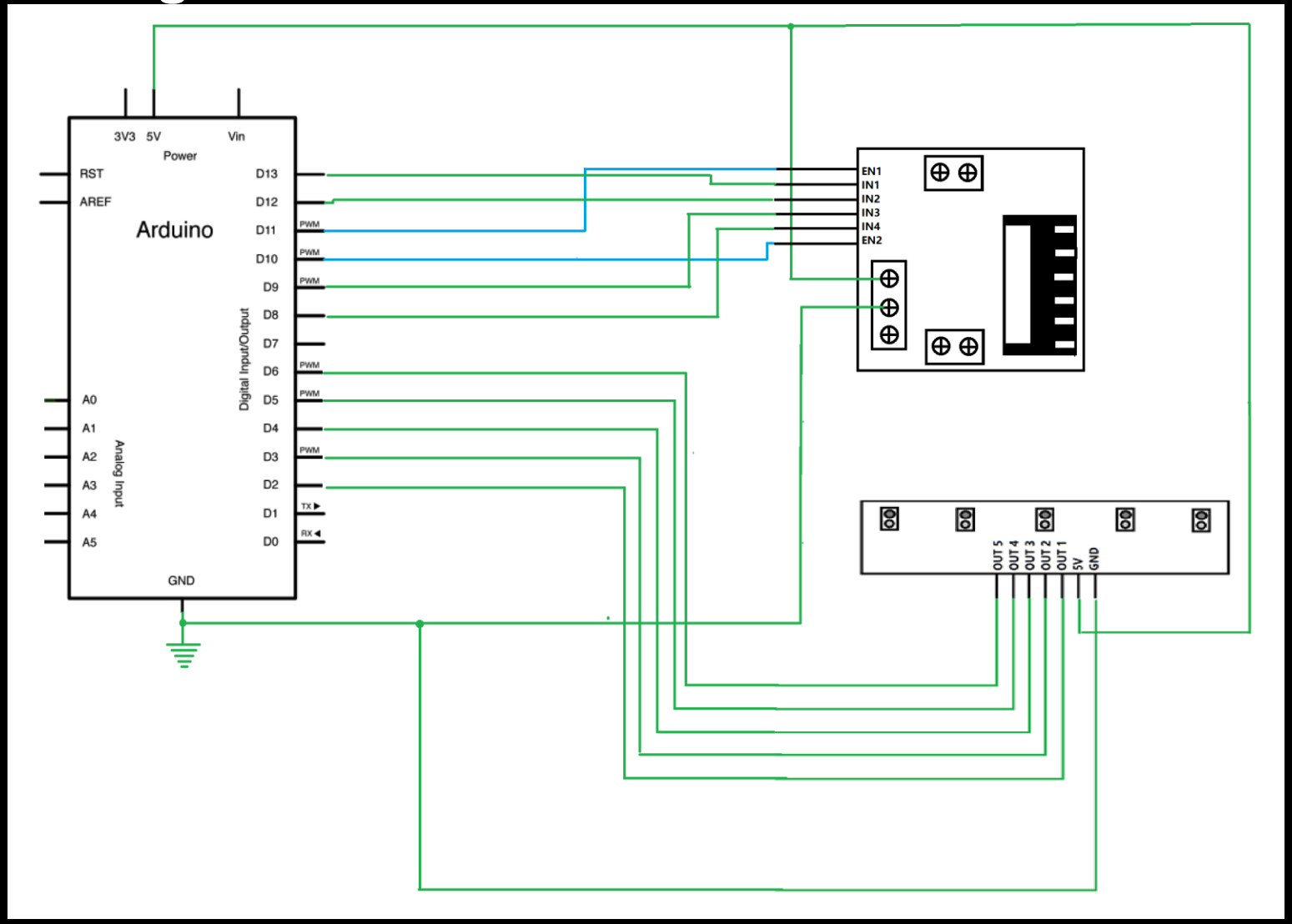


Working – IR sensor Array

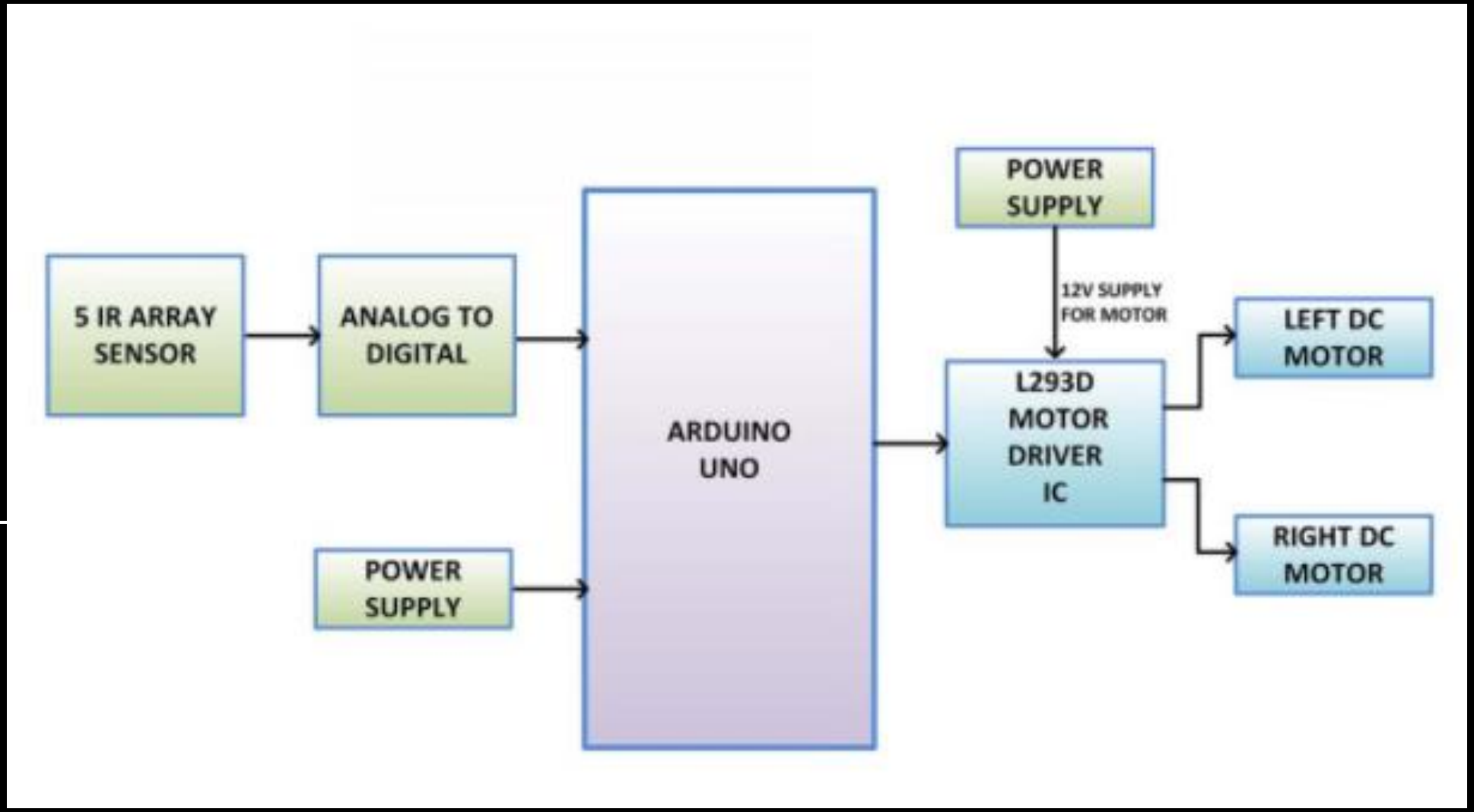
Battery positive	Arduino Vin
Battery negative	Arduino GND
Motor driver 12v pin	VCC or battery positive
Motor driver GND	Arduino GND
Motor Driver ENA A	D9
Motor Driver ENA B	D11
Motor 1 A and B	Motor driver A1 B1
Motor 2 A and B	Motor driver A2 B2
Arduino – D5 D4 D3 D2	Motor driver IN1,IN2,IN3,IN4
IR sensor array VCC	Arduino 5v pin
IR sensor array GND	Arduino GND
IR sensor array Output	D13, D12, D11, D7, D6



Line Following Robot - Circuit Diagram

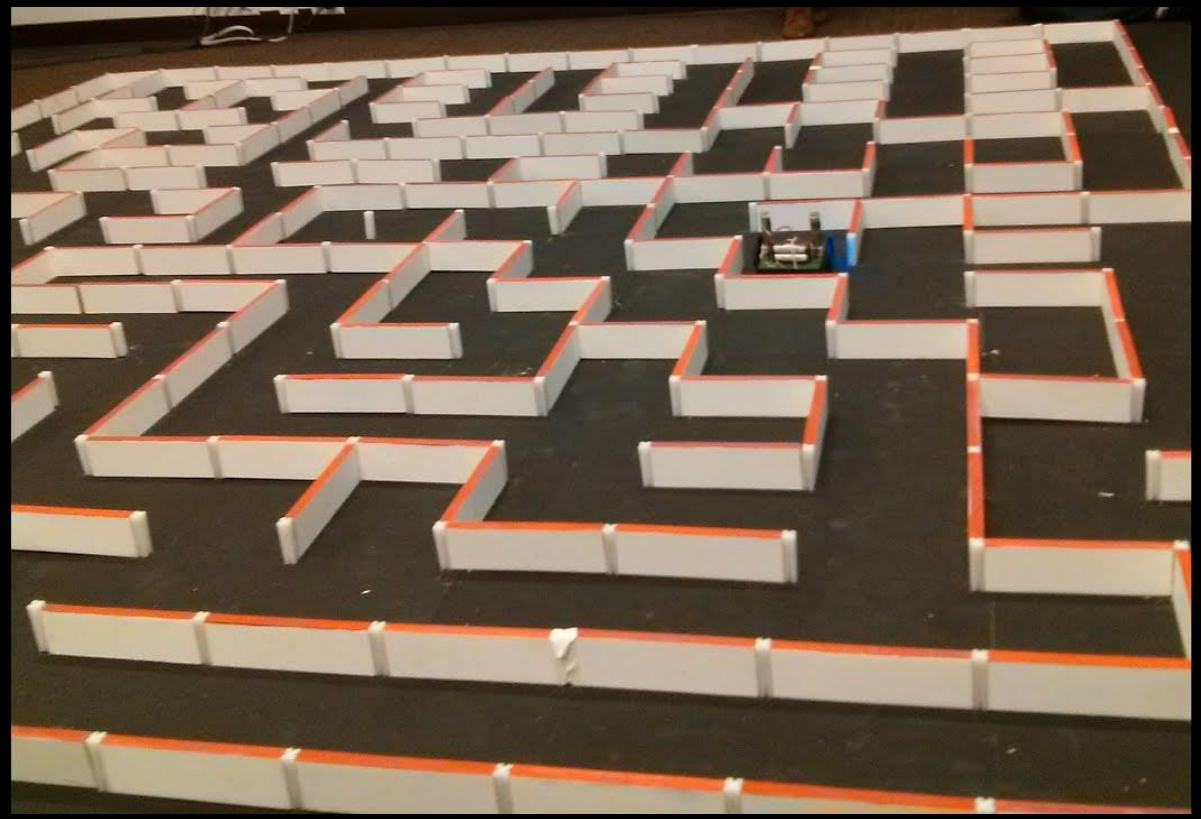


Line Following Robot - block Diagram



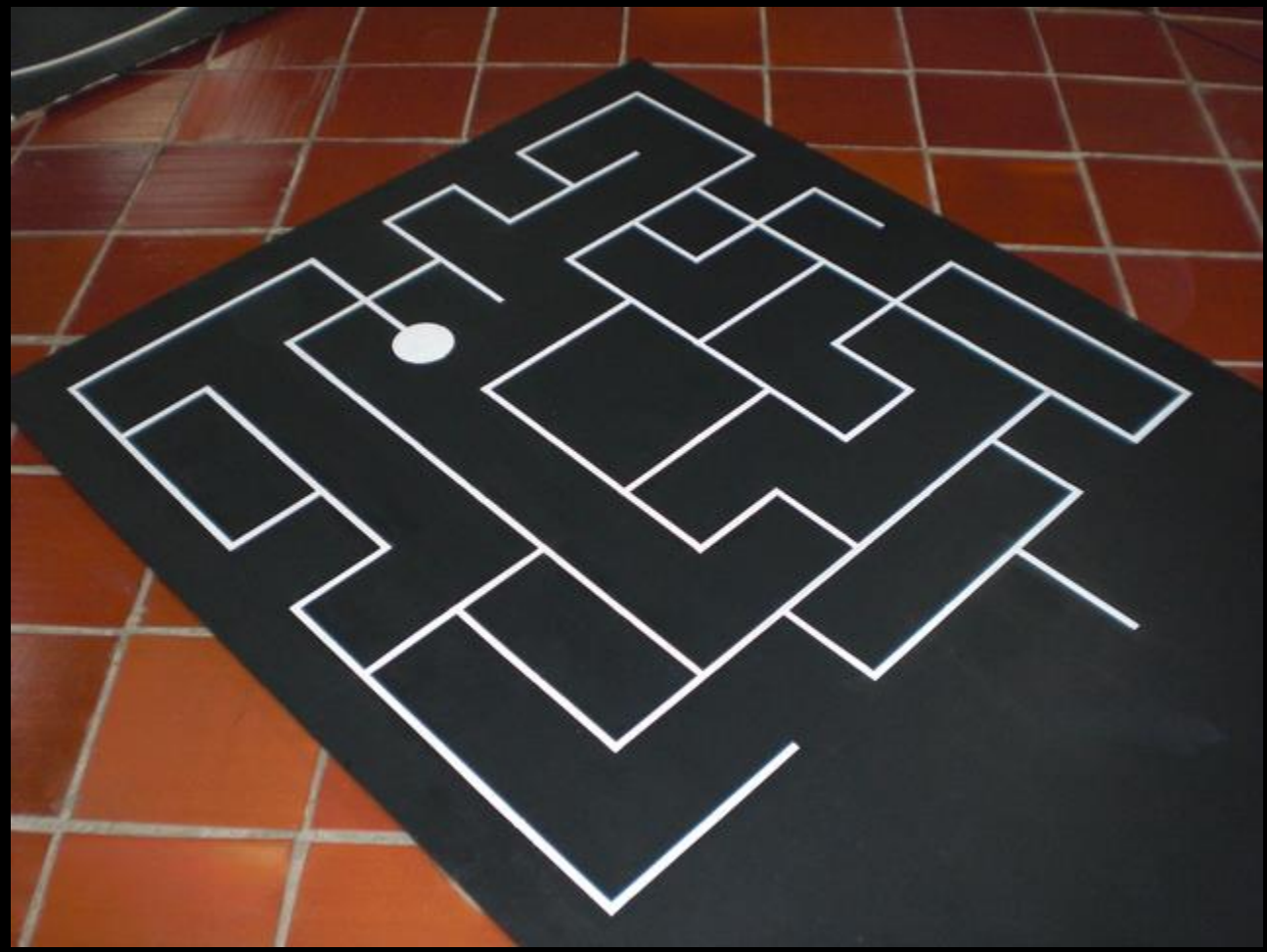
Line Following Robot - Circuit Diagram

- You can implement your maze solving robot with just one or two ultrasonic sensors also but having three sensors ensures more precise movements.
- Ultrasonic sensors measure the distance to the obstacle for the robot to be able to navigate autonomously.



Solving a line maze

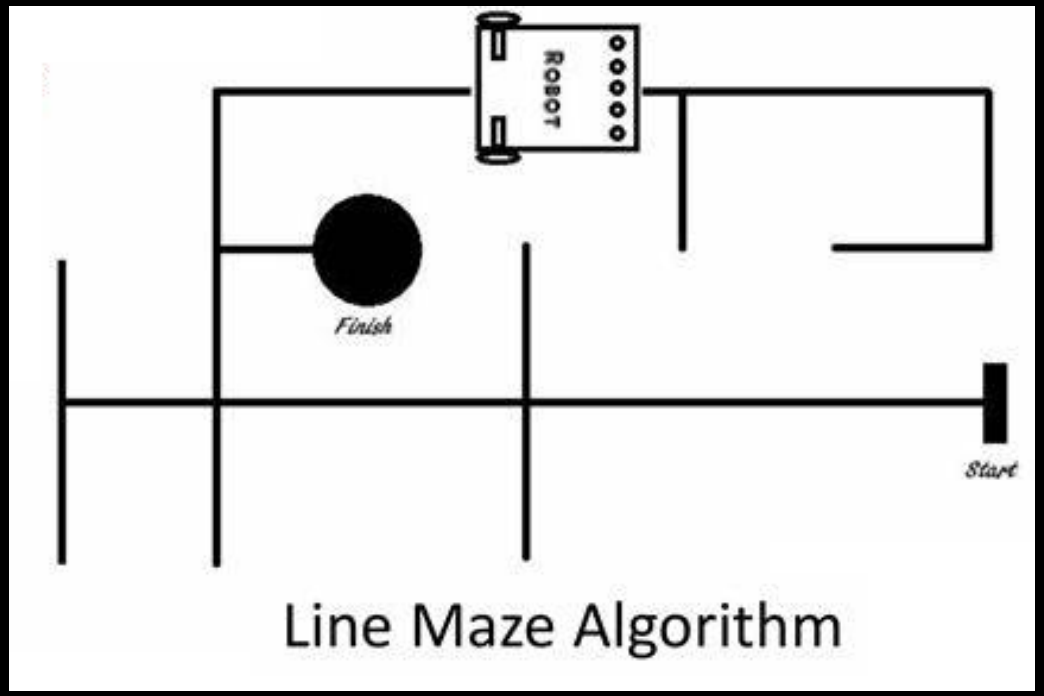
- From this slide will walk a robot hobbyist through the logic and procedure a robot needs to solve a line maze such as the one shown here
- Commonly there are two rules
  - Left hand rule
  - Right hand rule



Left hand rule

- The robot will always use the left-hand rule which means:
  - Always prefer a left turn over going straight ahead or taking a right turn
  - Always prefer going straight over going right

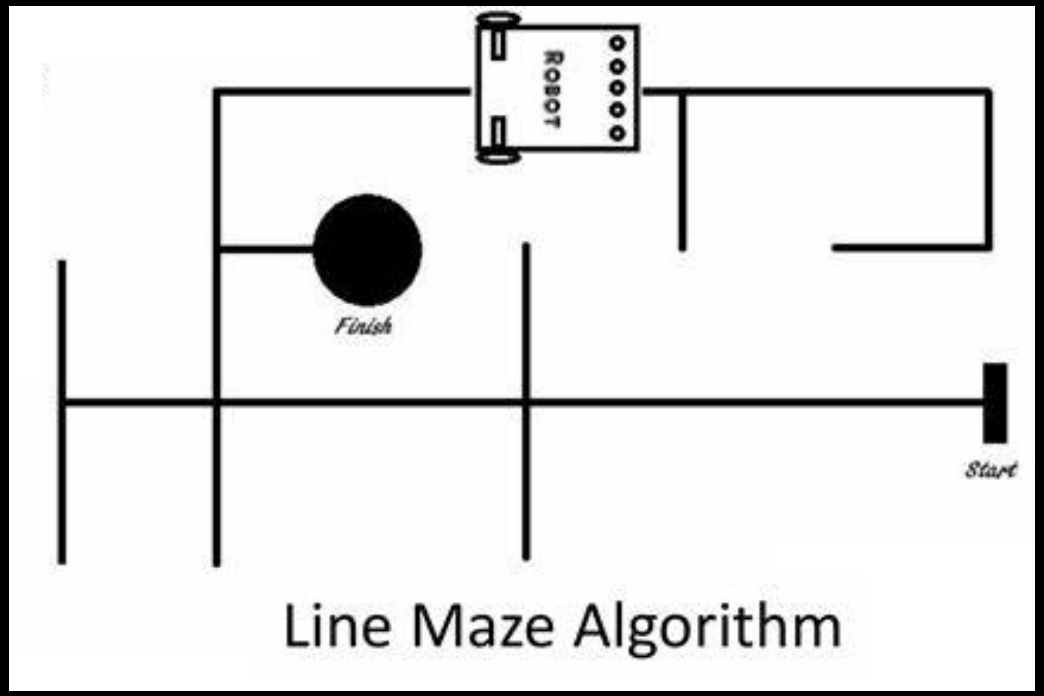
  If the maze has no loops, this will always get you to the end of the maze



Right hand rule

- The right-hand rule is just the opposite:
  - Always prefer a right turn over going straight ahead or taking a left turn
  - Always prefer going straight over going left

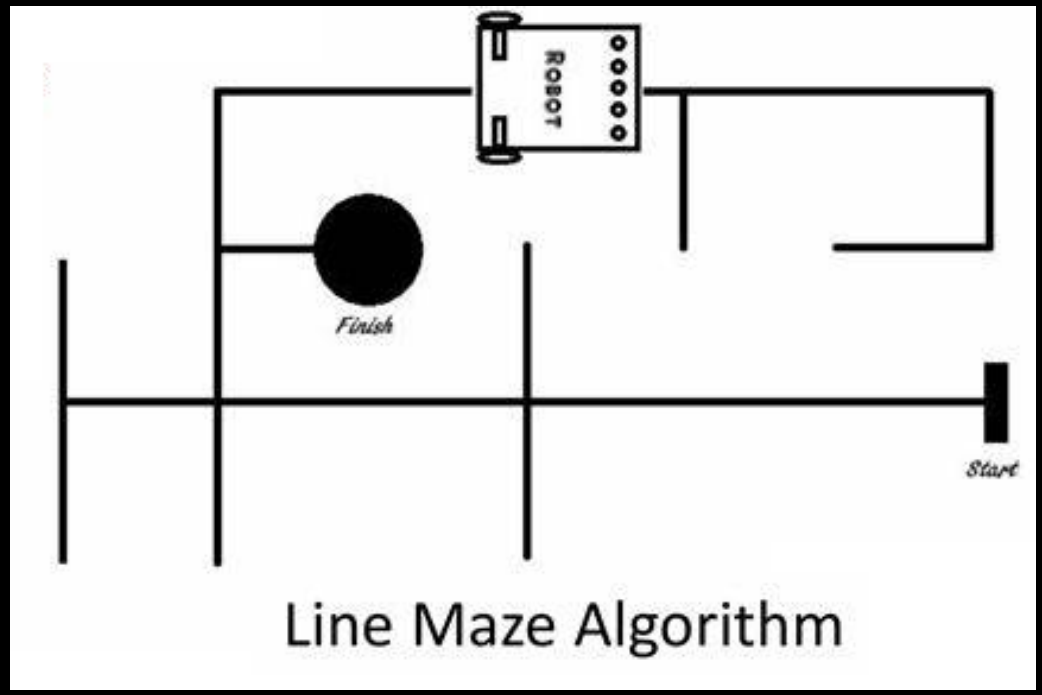
  If the maze has no loops, this will always get you to the end of the maze



Which rule do we use???

It really doesn't matter

- Both the left-hand and the right-hand rules will get you to the end of the simple maze
- Which you select is purely a matter of personal preference
- Just pick one and be consistent

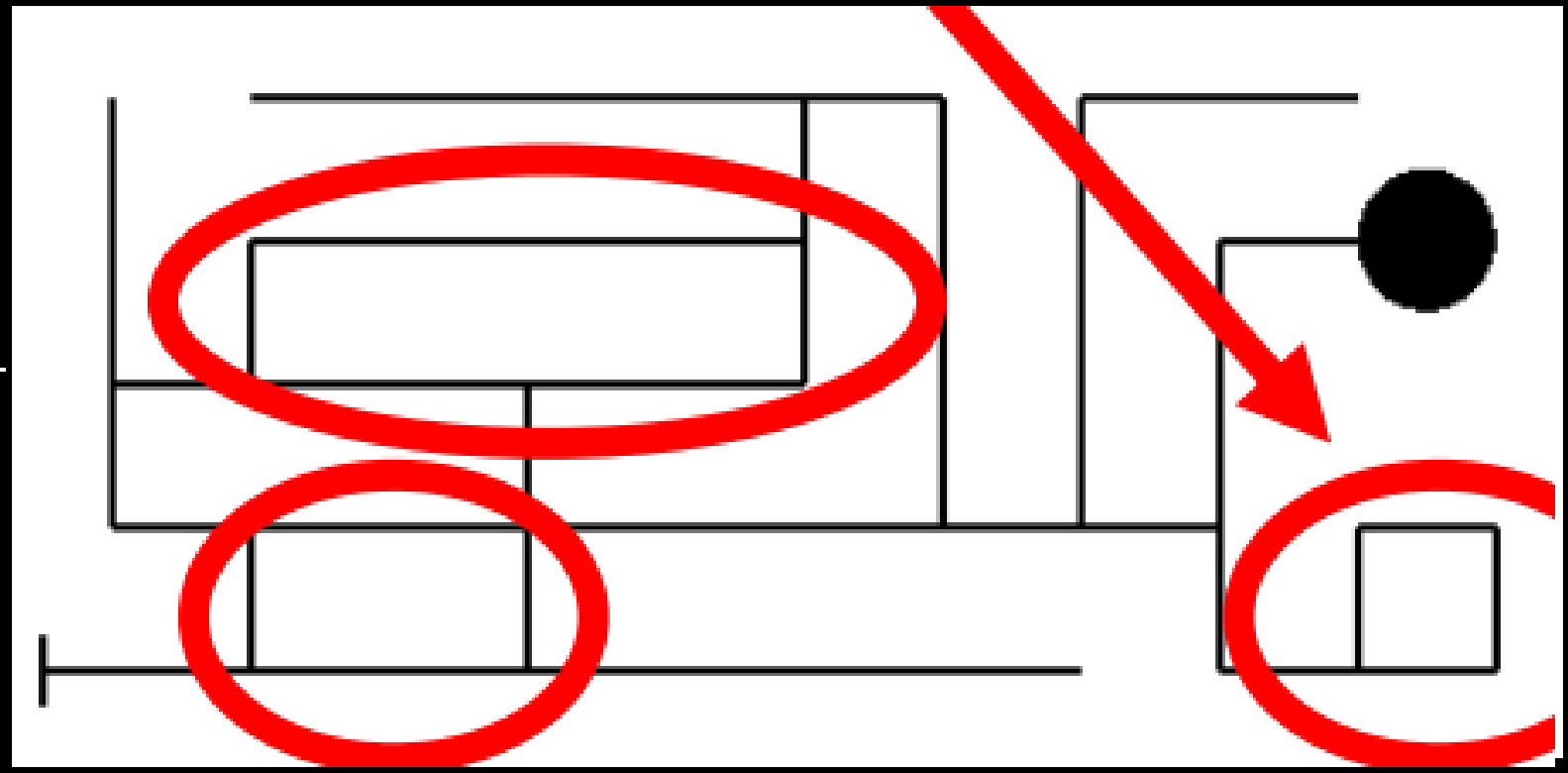


Line Maze Algorithm

Simple Maze

In the below maze, notice that

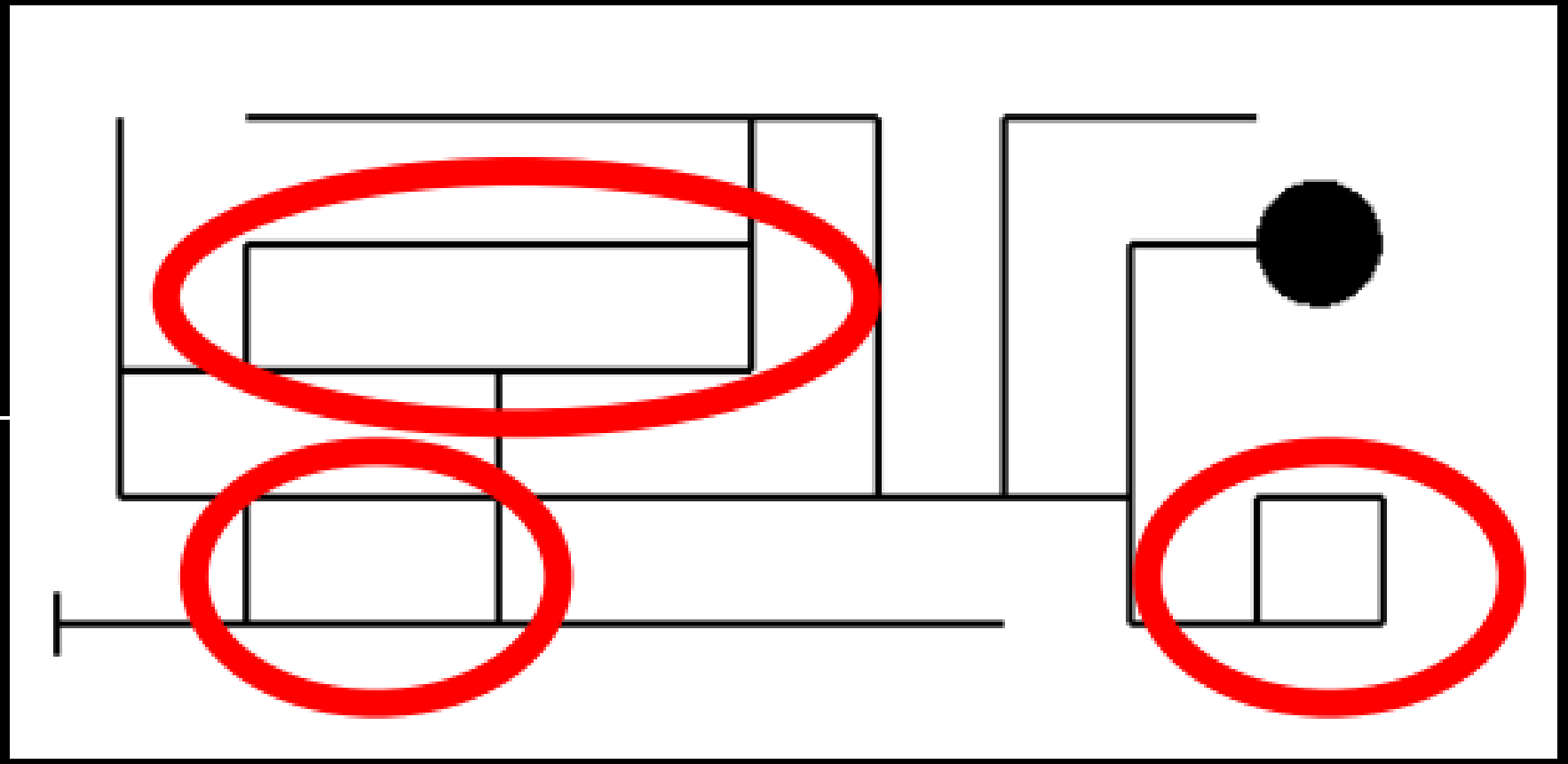
- The left-hand maze has no loops. Using the left hand (or right hand) rule will always get you to the end of the maze
- The right-hand maze has several loops
- Notice the loops in the right-hand maze



Simple Maze

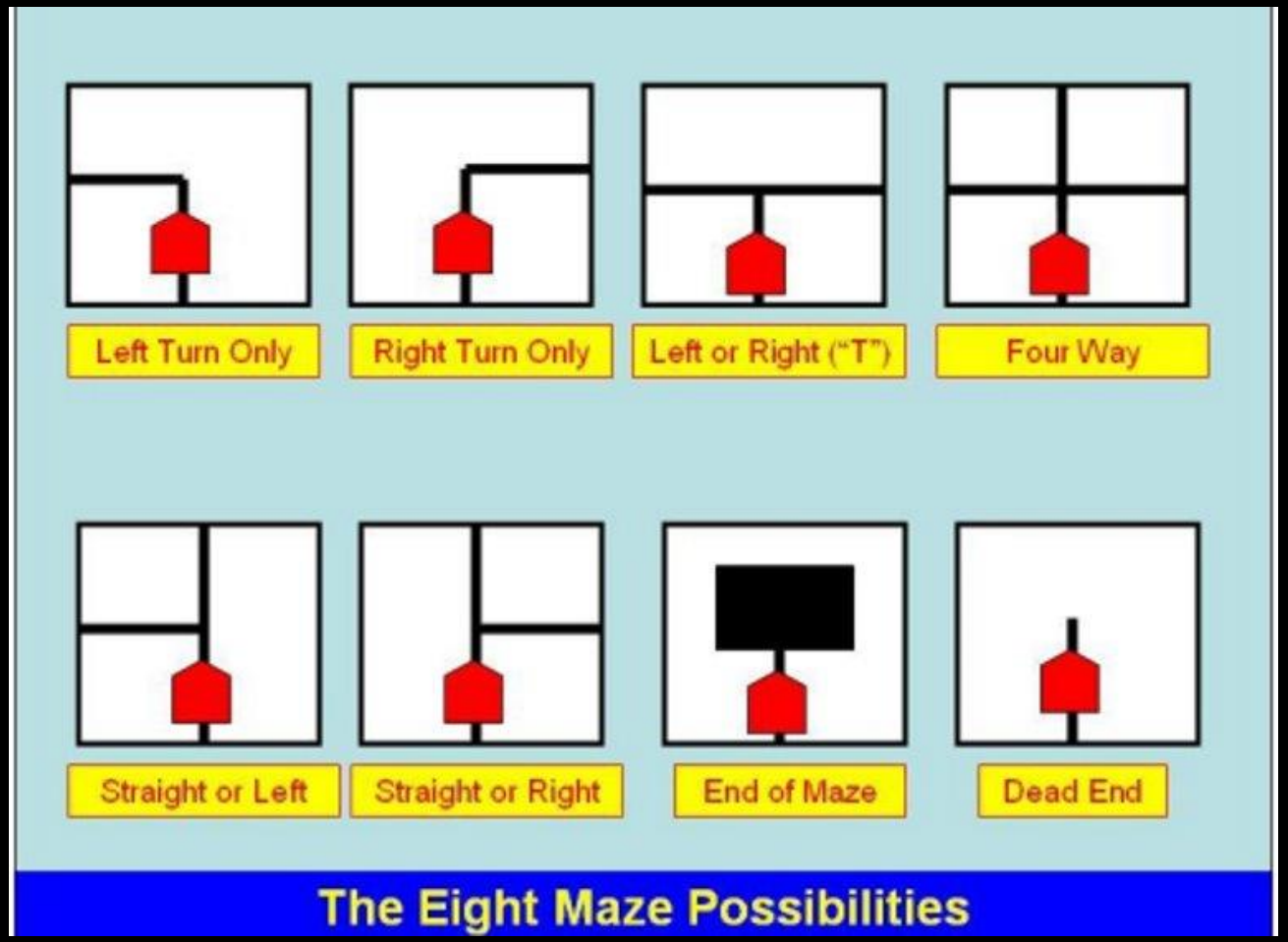
In the below maze, notice that

- At the present time, this presentation does not address how to solve the maze below. This algorithm may be added at a future date



Algorithm - Maze solving robot

Given a maze with no loops, there are only 8 possible situations that the robot can encounter

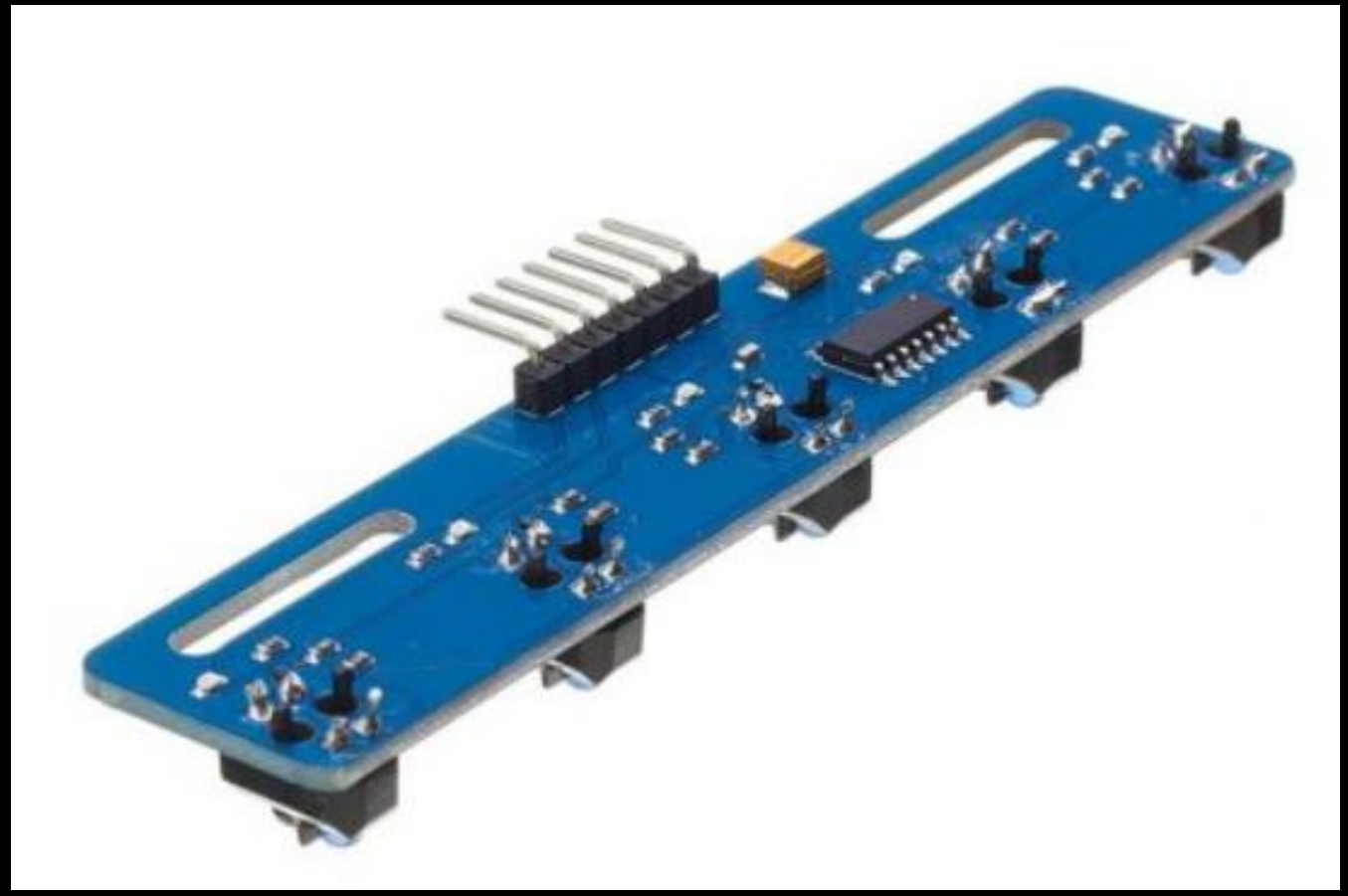


Line Sensor Array

- 5 Channel Line Tracking Sensor Module is a sensor board designed for use with line follower robots. This module has been sufficient to meet the day-to-day task of tracking, but also with the infrared distance sensor and touch detection sensor, the board makes your robot design able to adapt to the situation easily

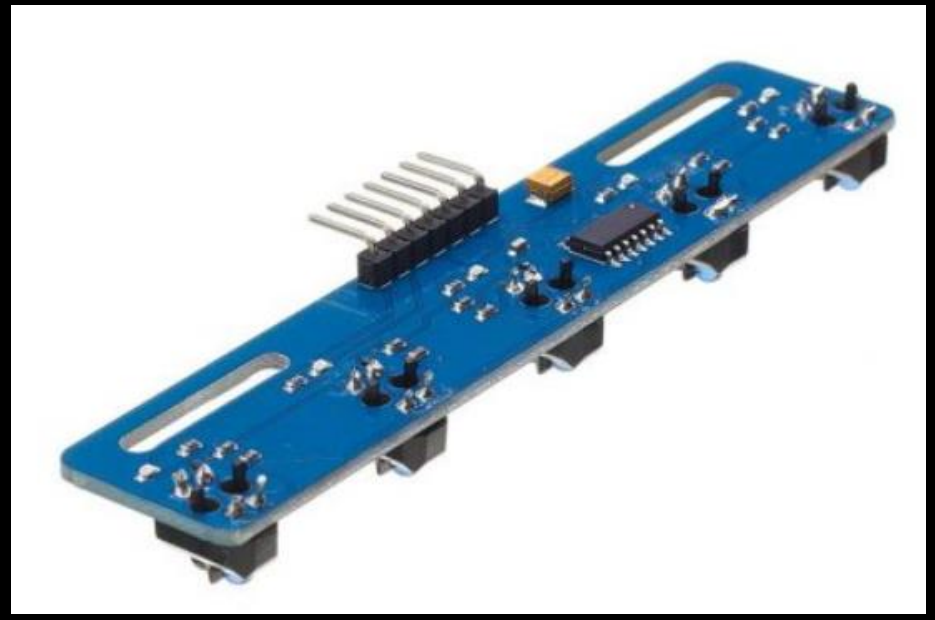
Features:

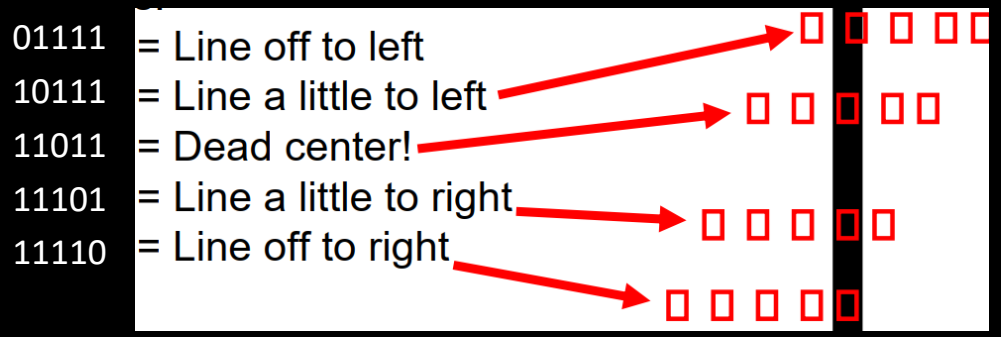
- Input voltage: 5V DC VCC, GND Pins.
- Output: 5 from is S1, S2, S3, S4, S5 digital.
- Output: 1 from Bump switch is CLP digital.
- Output: 1 from IR Obstacle sensor Near digital.



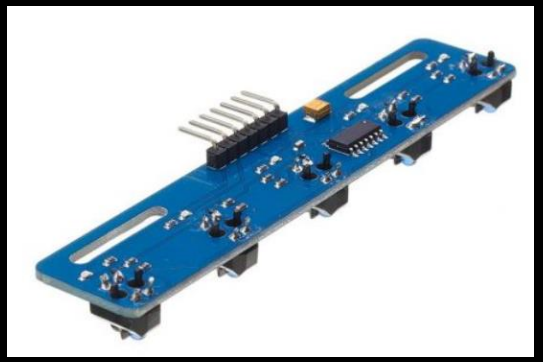
Line Sensor Array

- Line sensors can have several configuration. This sensor comes with five infrared sensors like those shown here
- Line sensors shine visible or infrared light down at the floor and the measure the reflection
- Using a 0 signal to mean " sensor sees black" and 1 to mean "sensor sees white".
- A robot travelling along the black line to the right might produce several patterns

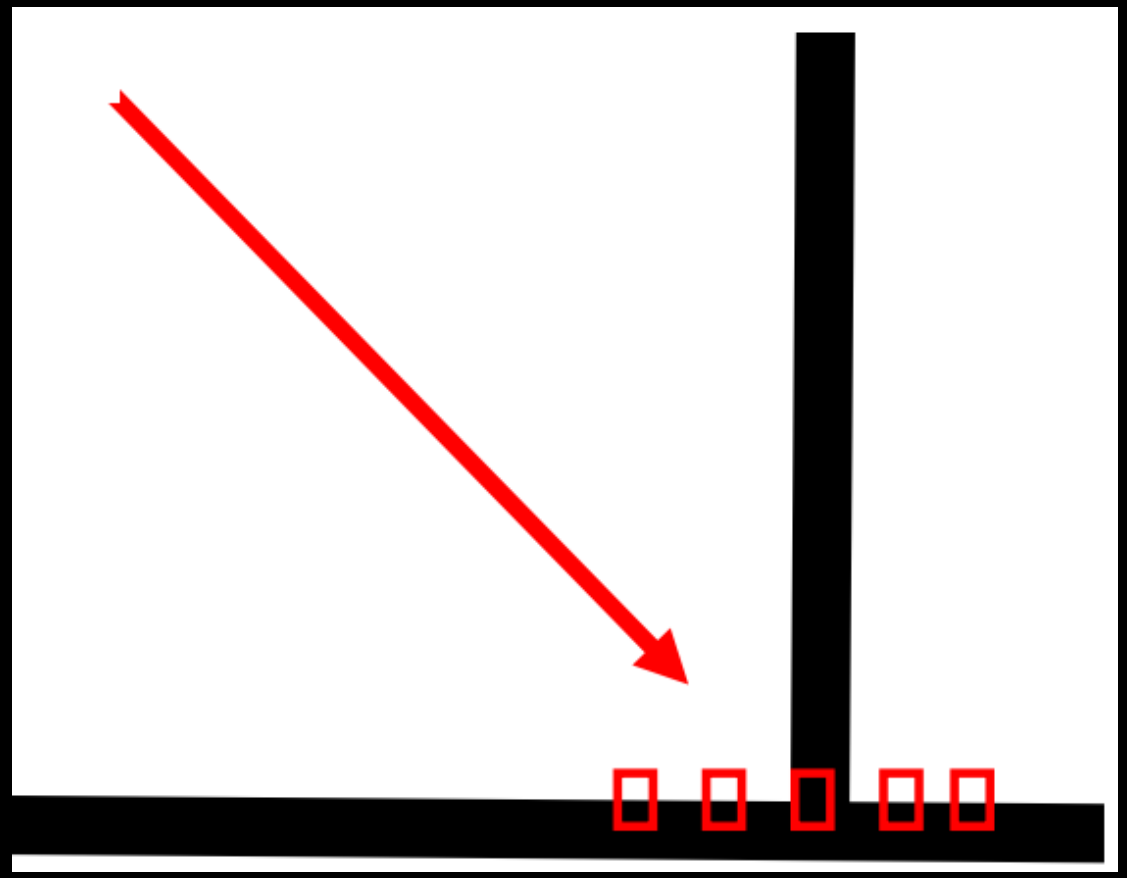




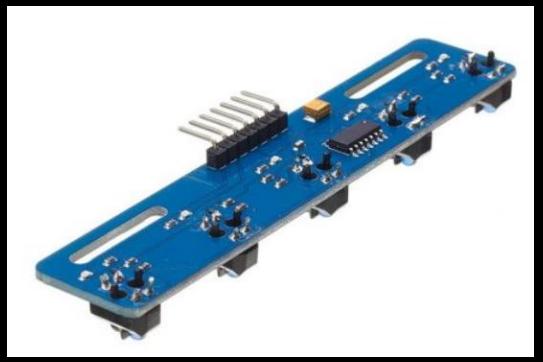
Line Sensor Array



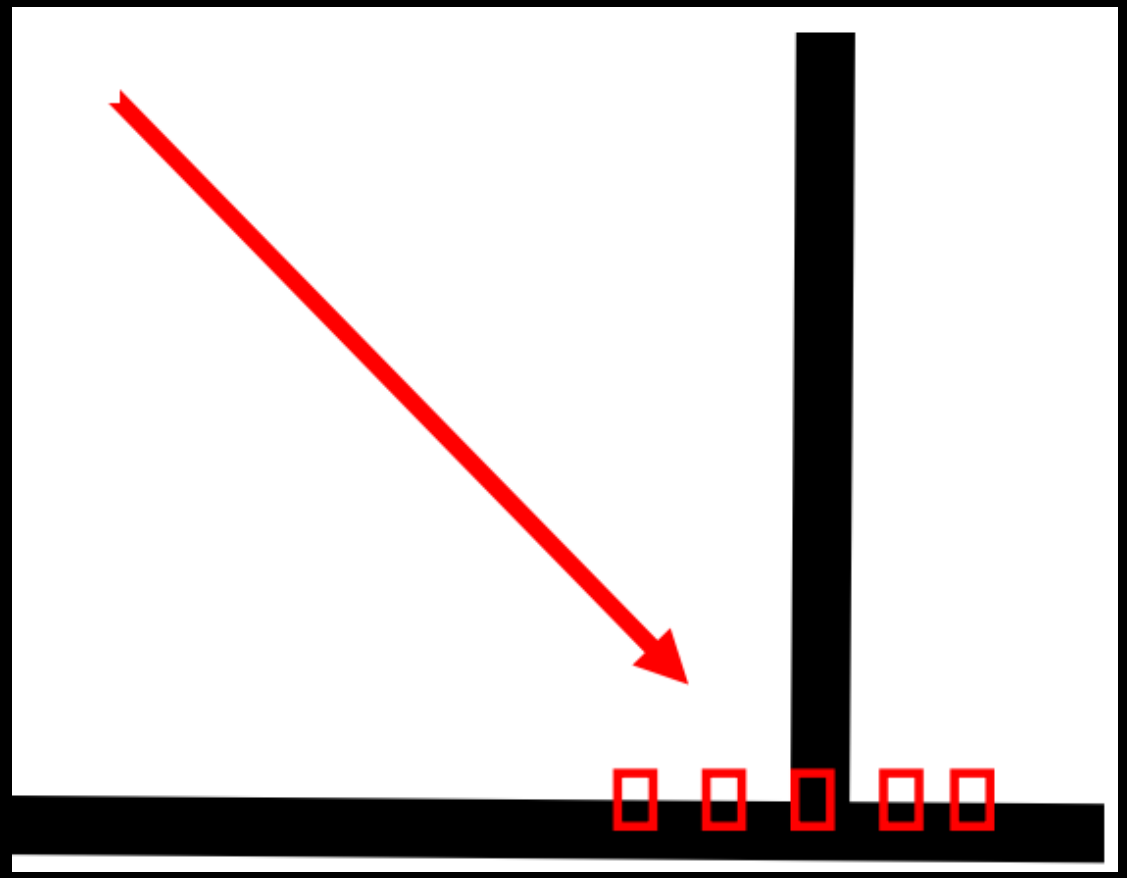
- Notice that if the robot is travelling downward on the black line to the right, when it reaches the “T” intersection at the bottom, the pattern will change from 11011 to 11111
- The program reads these patterns to determine where it is in the maze



Line Sensor Array

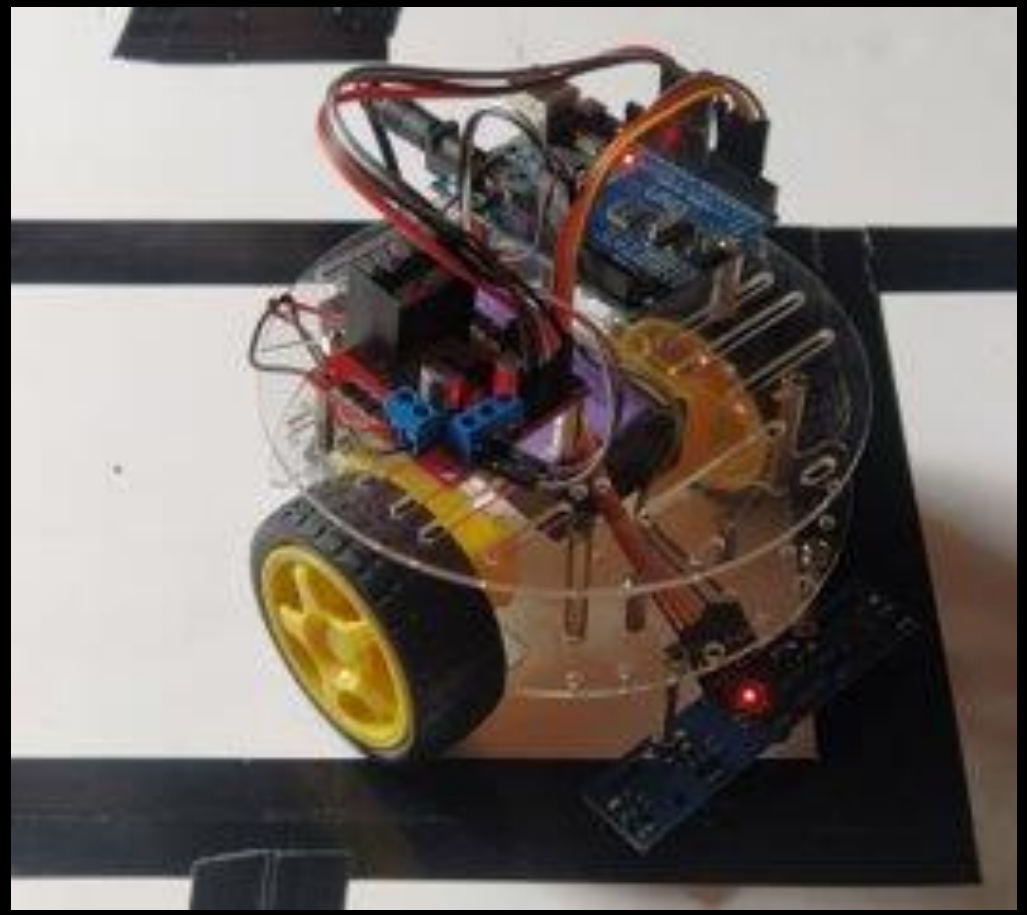


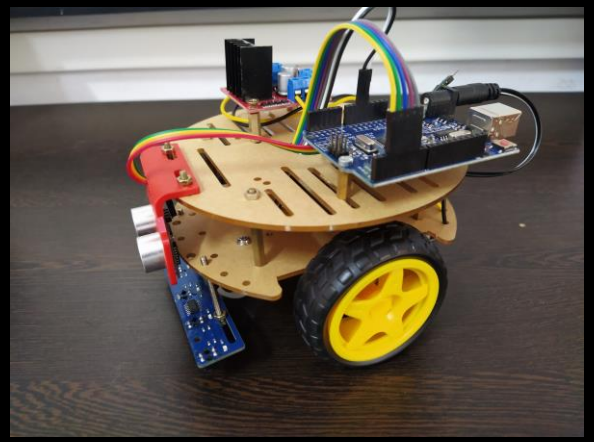
- Notice that if the robot is travelling downward on the black line to the right, when it reaches the “T” intersection at the bottom, the pattern will change from 11011 to 11111
- The program reads these patterns to determine where it is in the maze



How many combinations? 2^5 or 32

- With five sensors that can each be a one or a zero, there are 2^5 or 32 possible combinations.
- We will be walking through many of these, but also be aware some of these are impossible or highly unlikely in the mazes
- Examples 01010, 10101,11100, 00111 etc.

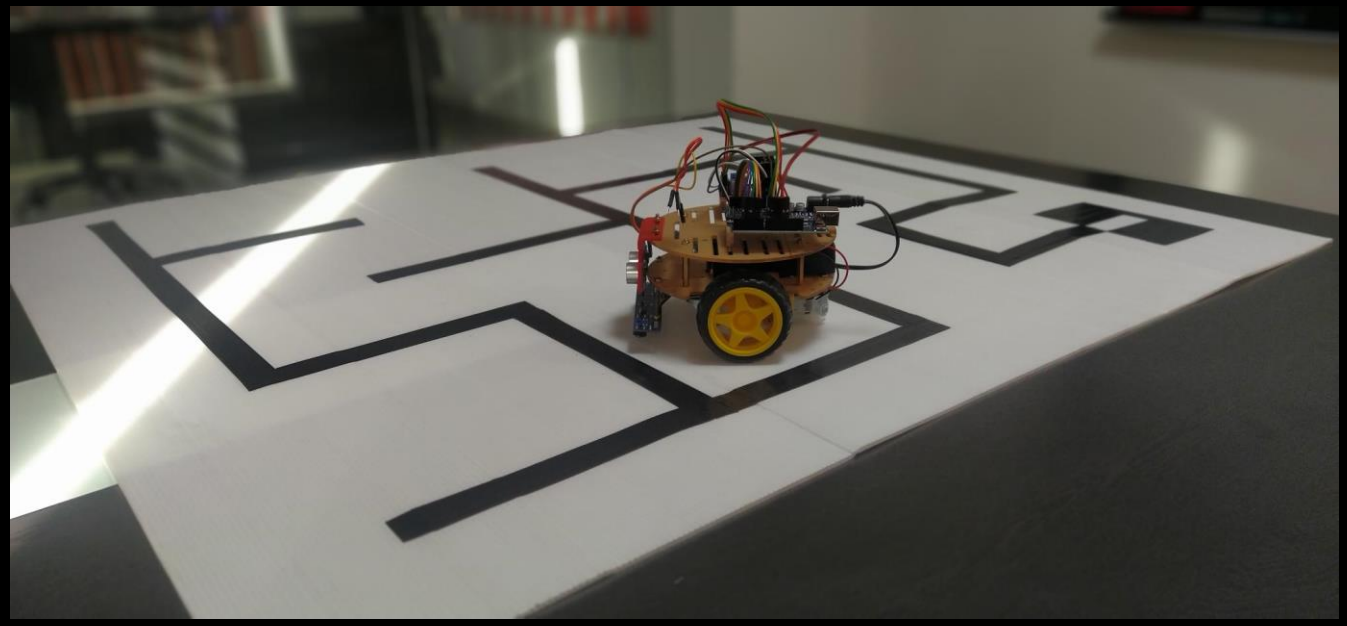




How robot behaviors?

- The robot most of the time, will be involved in one of the following behaviors
  - Following the line. Looking for the next intersection
  - At an intersection, deciding what types of intersection it is
  - At an intersection, making a turn

These three steps continue looping over and over until the robot senses the end of the maze

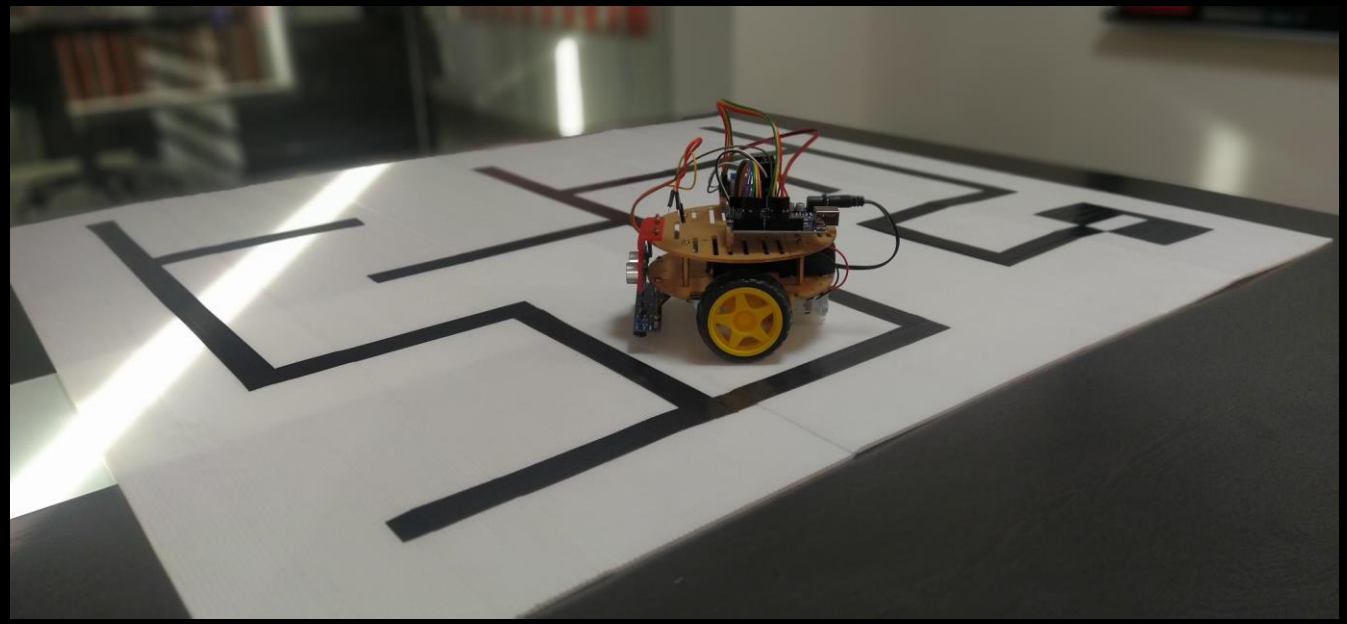


Follow the line

Following the line is relatively easy. Here is some pseudocode:

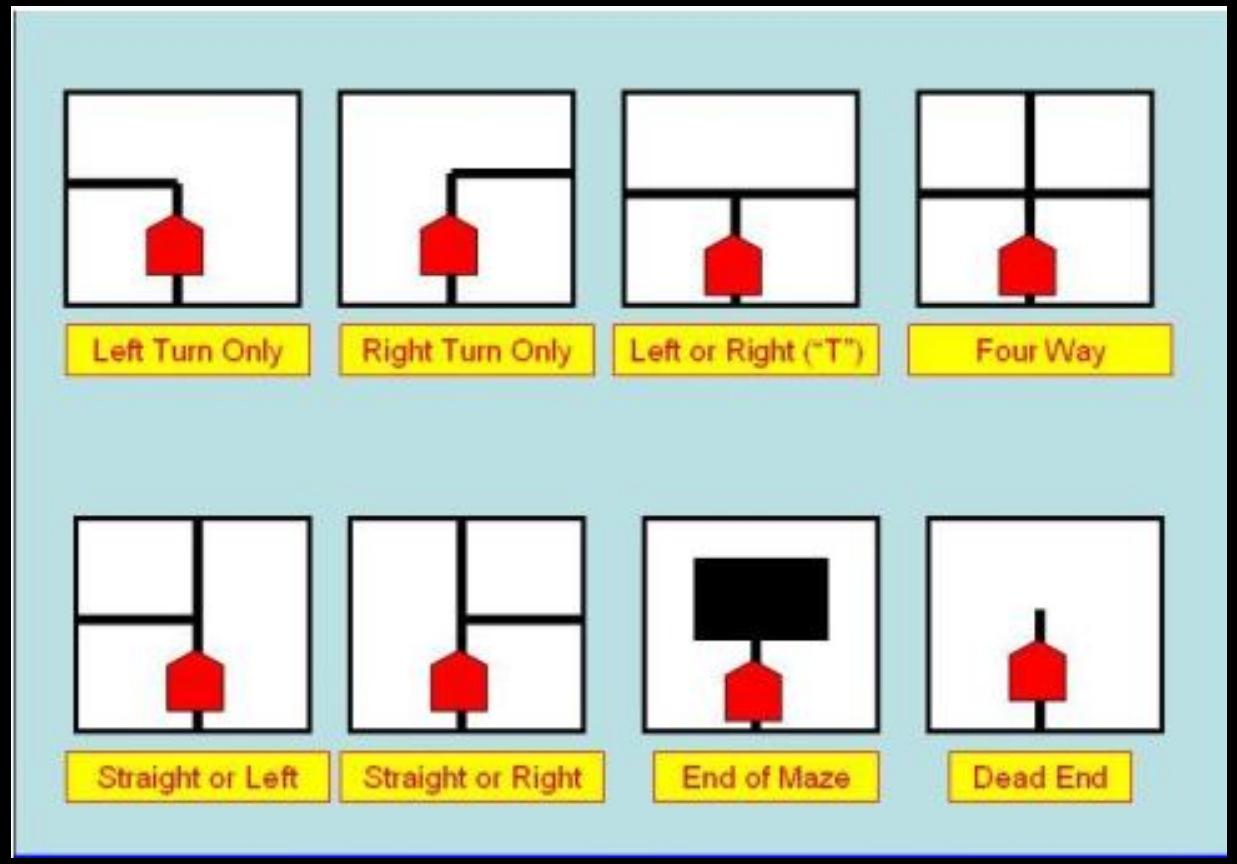
- Select case pattern
    - Case pattern = 11011 “full speed ahead”
    - Case pattern = 10011 “Go left a little”
    - Case pattern = 11001 “Go right a little”

    Slow, medium and fast are arbitrary speeds that should get the robot turning or moving in the correct direction when straying from the line



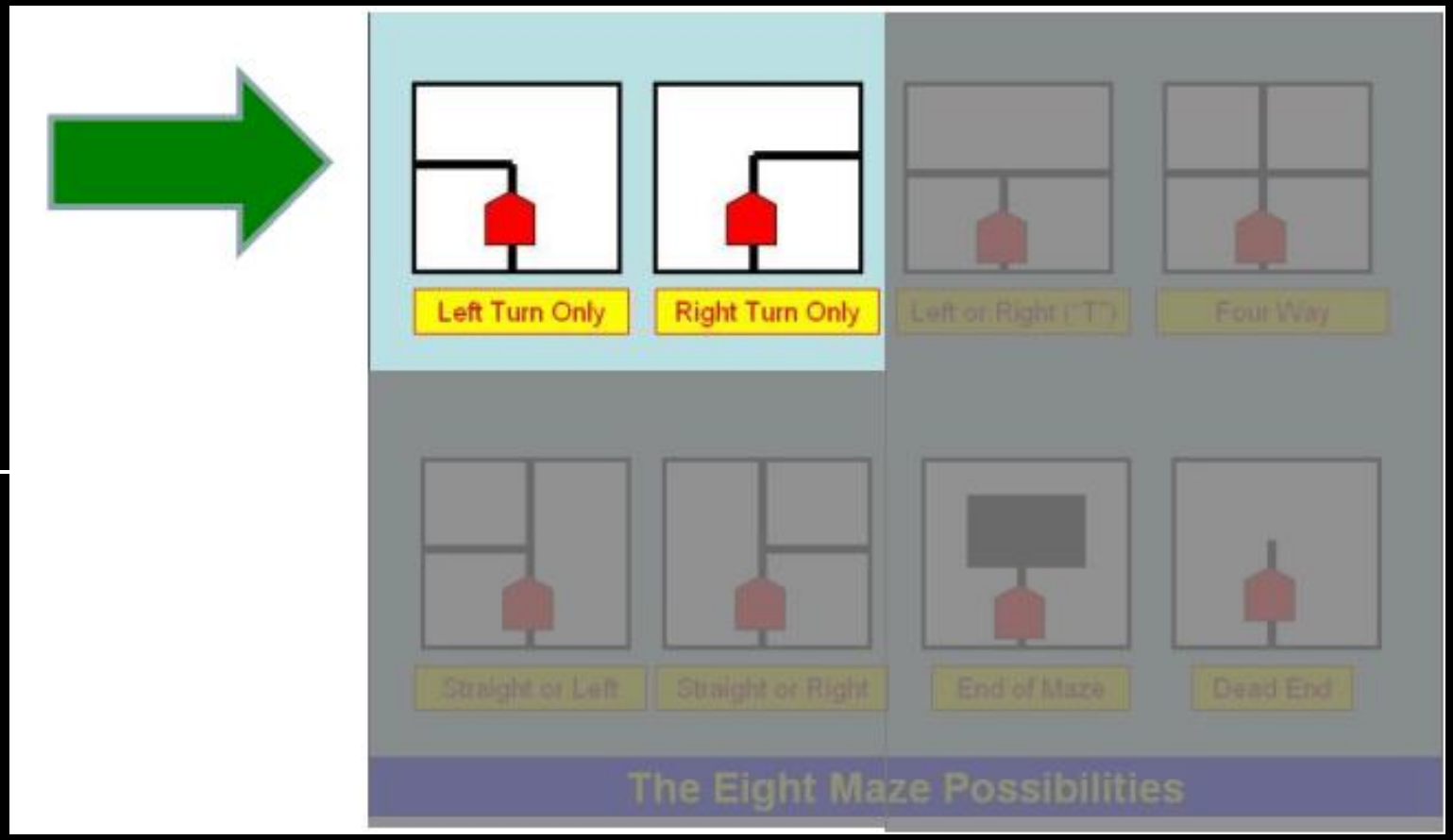
Intersection and turn handling

- The robot needs to be taught the correct behavior depending on the type of turn or intersection it encounters
- First let us give a more rigid definition of an intersection. I will define an intersection as " a place where the robot has more then one choice of direction"



Not intersections

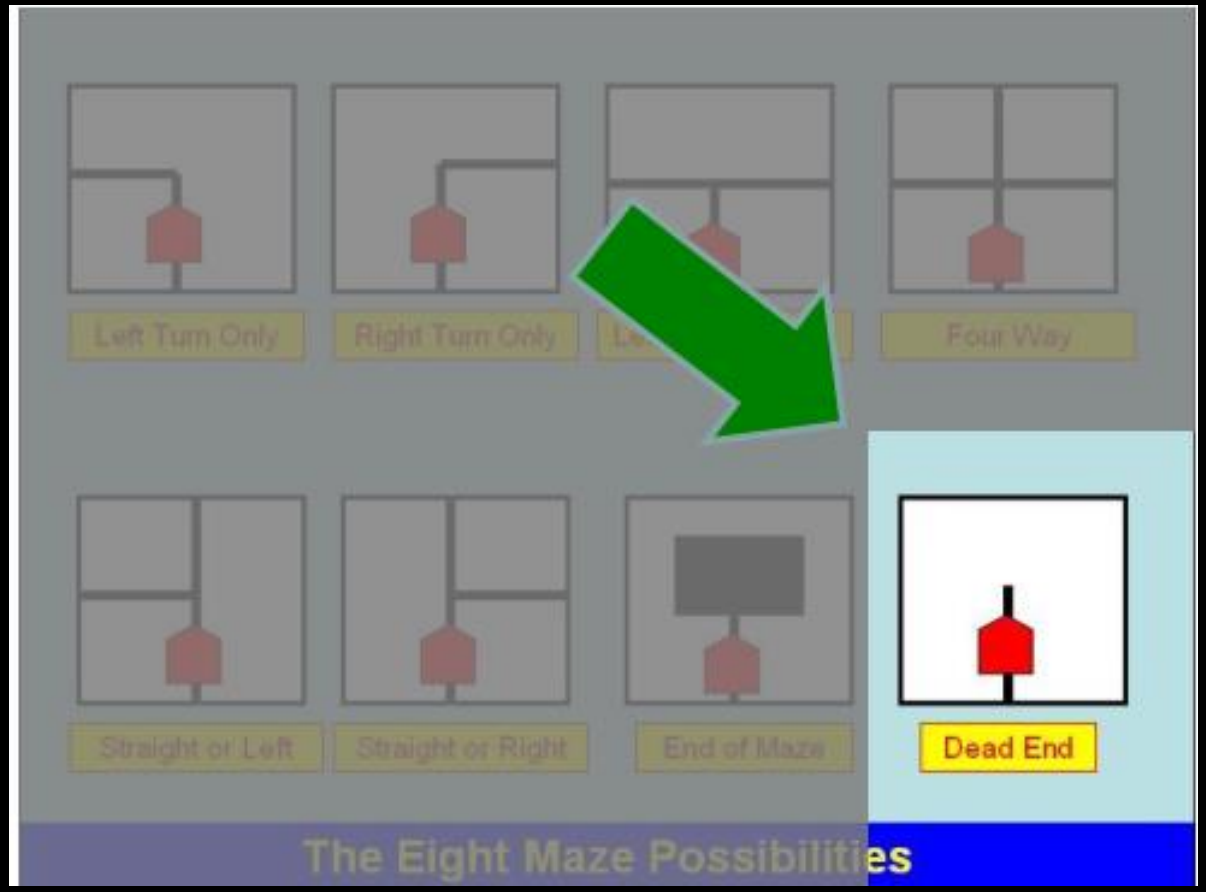
In these two cases, the robot has no choice but to make a 90-degree turn. Since the robot will always make the same 90 degree turn and it has no other option, this turn need not be stored when solving the maze.



Not intersections

In these cases, the robot has no choice but to make a 180-degree turn to exit the dead end. Since reaching a dead-end means that the robot has recently made a bad turn, we need to store this fact so that a previous turn can be corrected. The robot should not visit this dead-end on the next run

- The dead-end is the easiest intersection
- Sensors will go from 11011 to 11111
- This is the only normal situation where the robot should see all ones
- The correct behavior at a dead-end is to make a U-Turn.



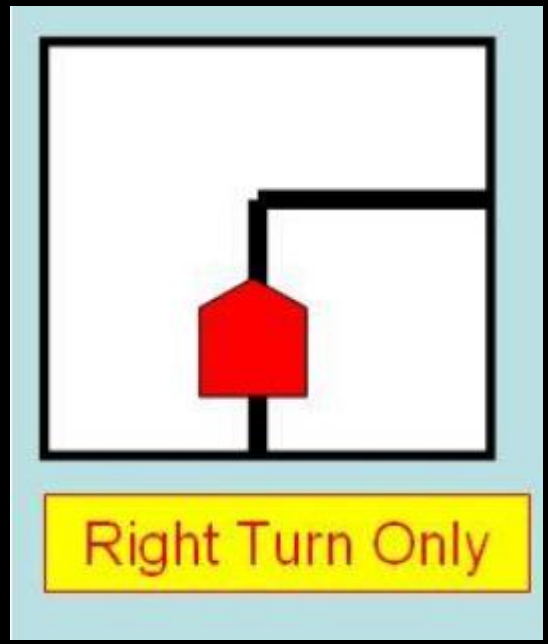
Right only or Straight / right

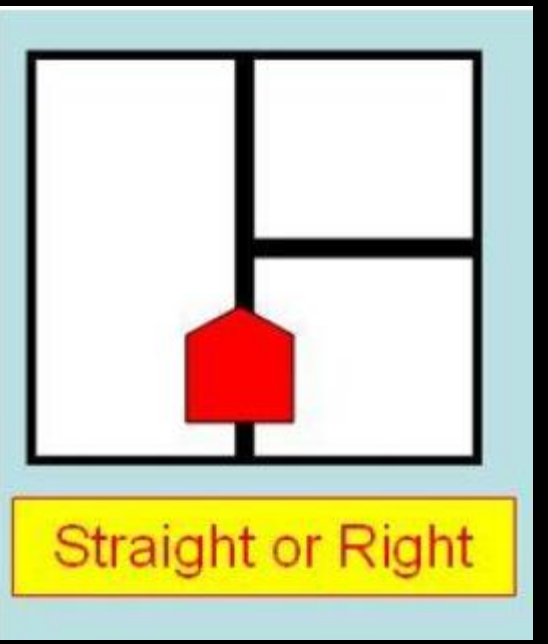
- Here is a situation where the same pattern will initially appear for two different situation
- Both shows the pattern 11000 when initially detected

How do we know which is which?

- Move forward one inch
  - Create a subroutine called forward inch() that moves the robot forward one inch
  - Now read the sensor again
  - If the sensor pattern is 11111 then the robot is at a ‘right turn only’

    with any other reading the robot is at the straight or right intersection





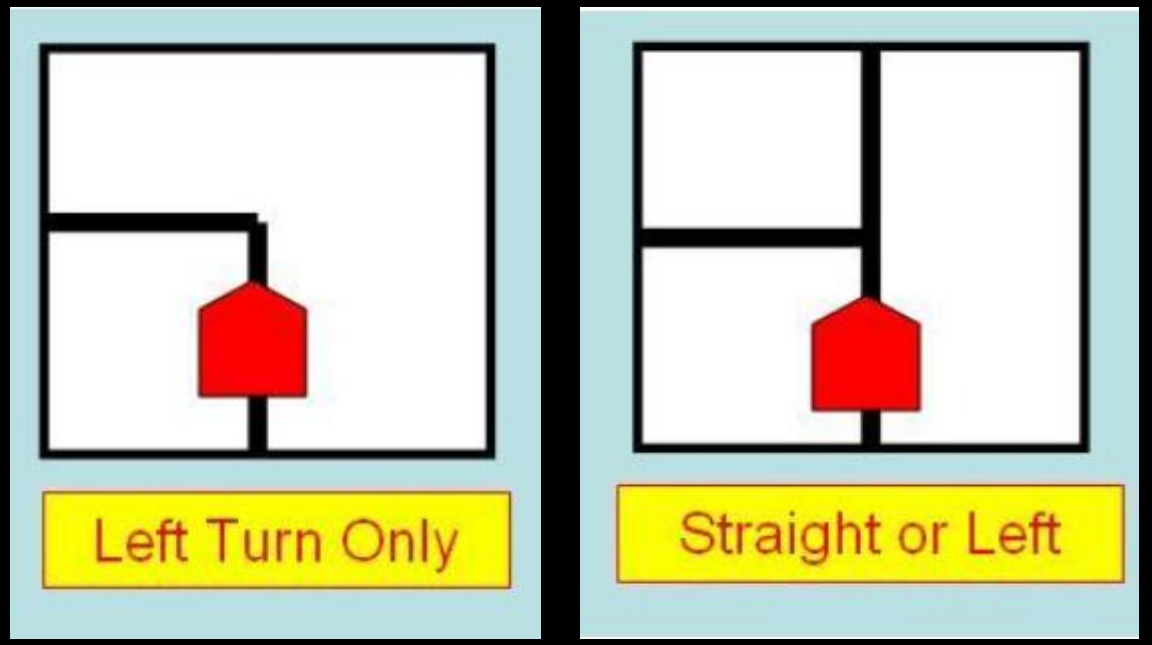
Left only or Straight or left

- Here is a situation where the same pattern will initially appear for two different situation
- Both shows the pattern 00011 when initially detected

How do we know which is which?

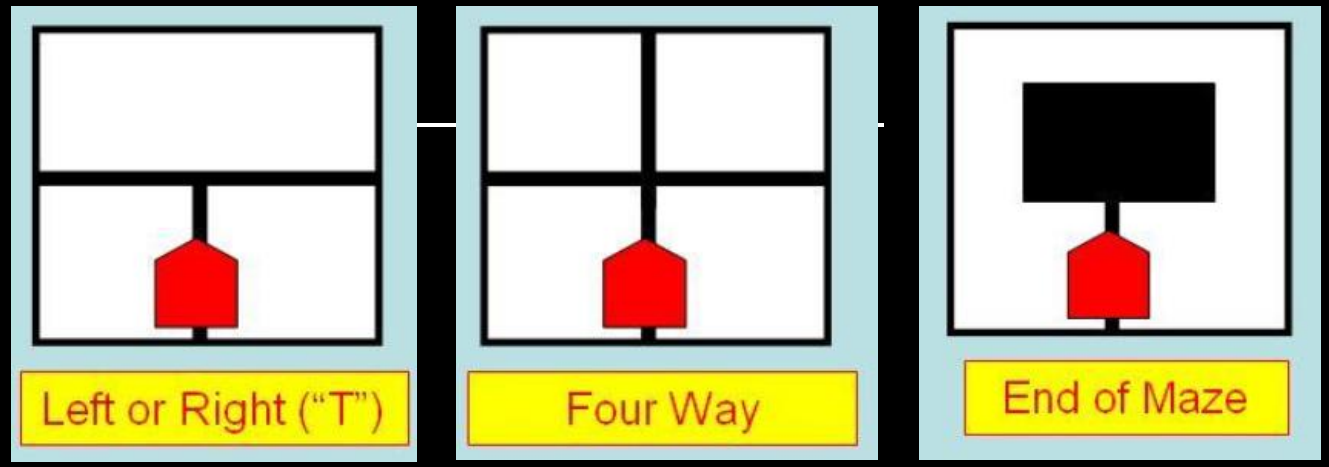
- Move forward one inch
  - Create a subroutine called forward inch() that moves the robot forward one inch
  - Now read the sensor again
  - If the sensor pattern is 11111 then the robot is at a ‘left turn only’

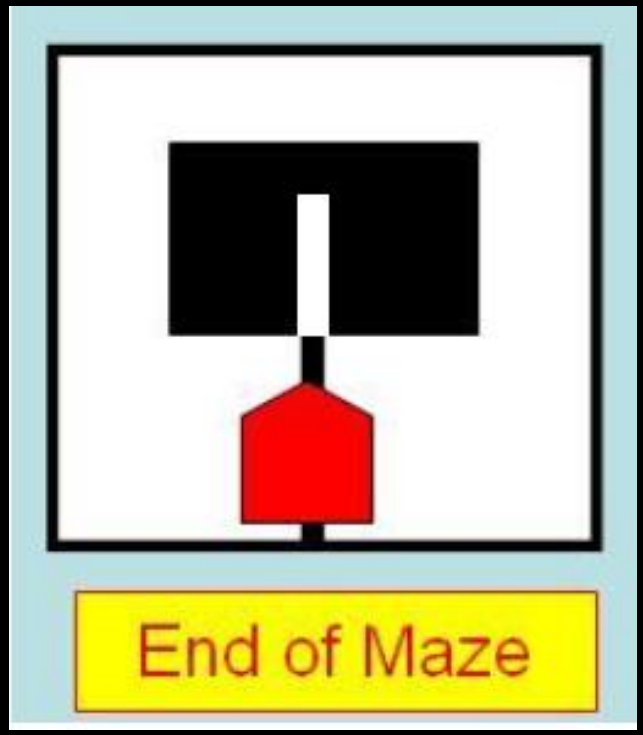
    with any other reading the robot is at the straight or right intersection



T junction, cross junction or Destination

- All these three intersection will present an initial pattern of 00000 to the robot
- Once again forward inch() will help us to determine the correct intersection
- In order to solve the maze, the robot needs to traverse the maze twice
- In the first turn, it goes down some number of dead-ends, but records these as bad path so that they can be avoided on the second run

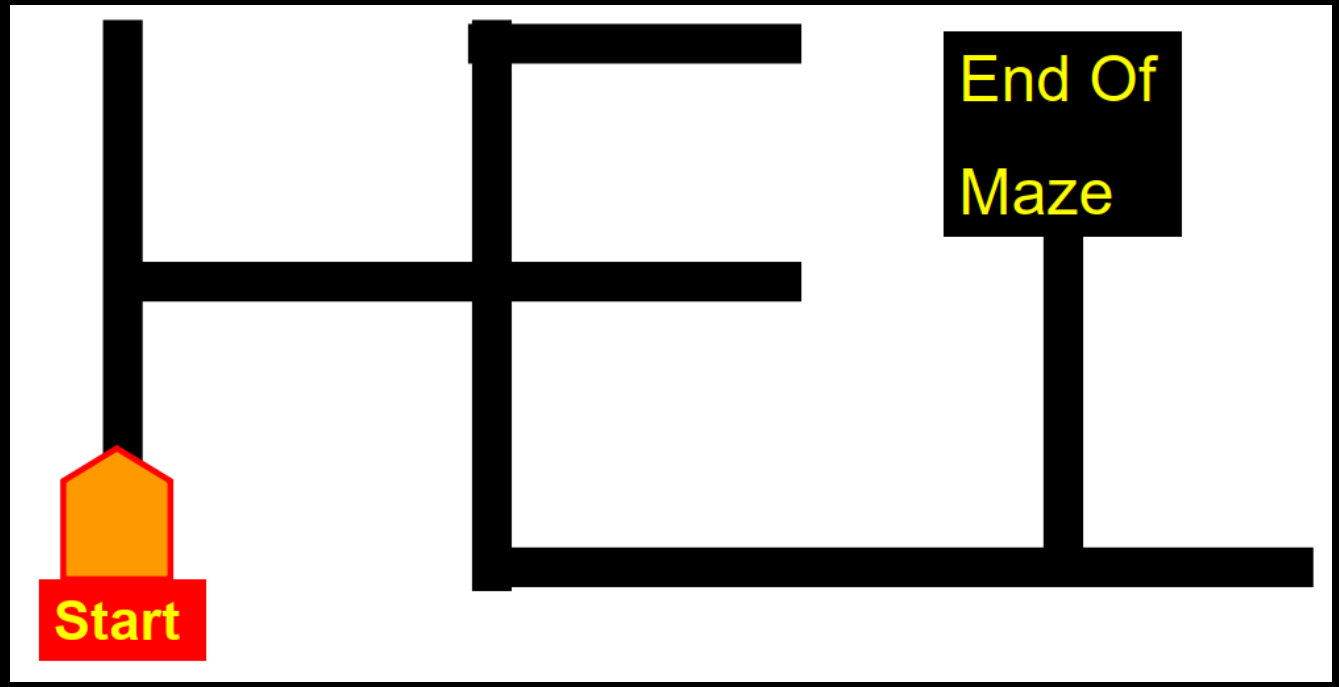




The main Algorithm

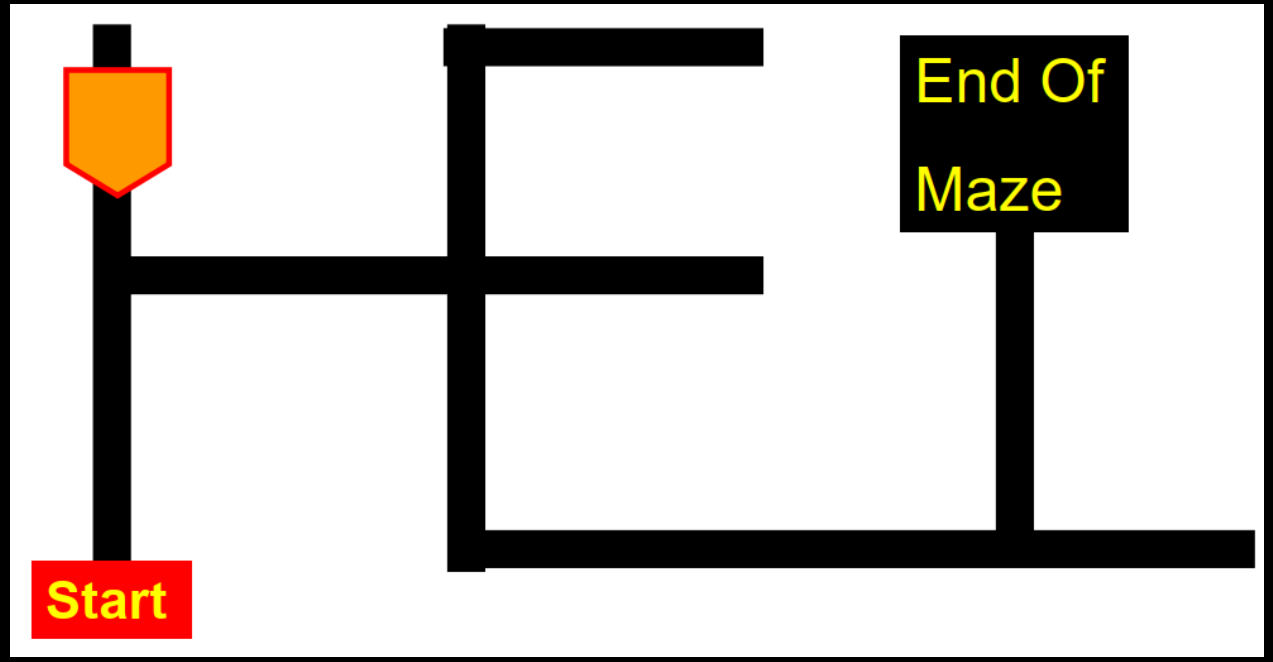
- First, we start with a very simple maze
- We will walk through this maze step-by-step to explain the maze solving algorithm
- So, we will use the left-hand rule for the first pass through the maze.
- There will be several dead-ends encountered and we need to store what we did so the correct path can be computed
- The robot takes off and starts to solve the maze

Assume that we are using left-hand rule



The main Algorithm

- First, the robot senses a straight or right intersection. The left-hand rules requires that the robot go straight F
- The robot continue straight stores the turn taken “F” in memory
- The robot runs off the path (pattern = 11111) and the correct behavior is to take a U turn
- After the U turn, record the turn taken U in memory FU



The main Algorithm

- In this maze, every dead-end encountered means that the robot has taken a wrong turn
- In any maze, the best / shortest path through it never includes going down a dead-end path
- So, a dead-end can always be said to tell us: the previous action / turn was
- We can't be sure how we got here or the correct way to modify the previous turn until we turn to the previous intersection
- The left hand-rule calls for a left turn, so take a left and record the turn FUL