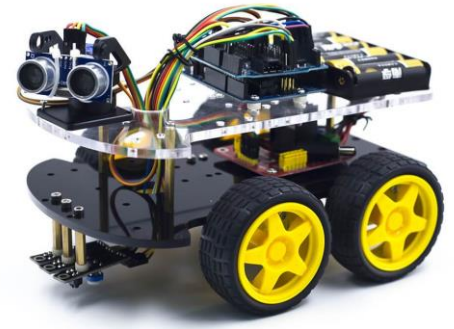
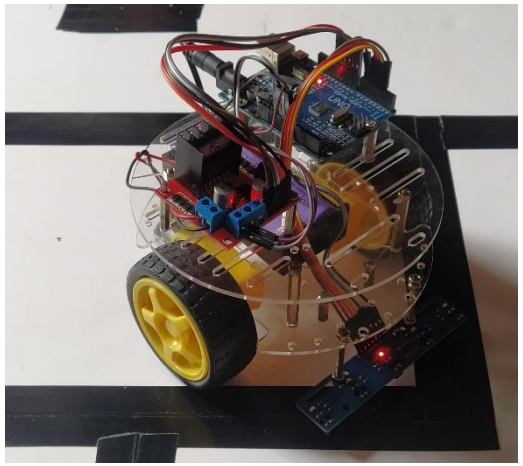
NSDC – Junior Skills Championship Mobile robotics



N·S·D·C
National
Skill Development
Corporation

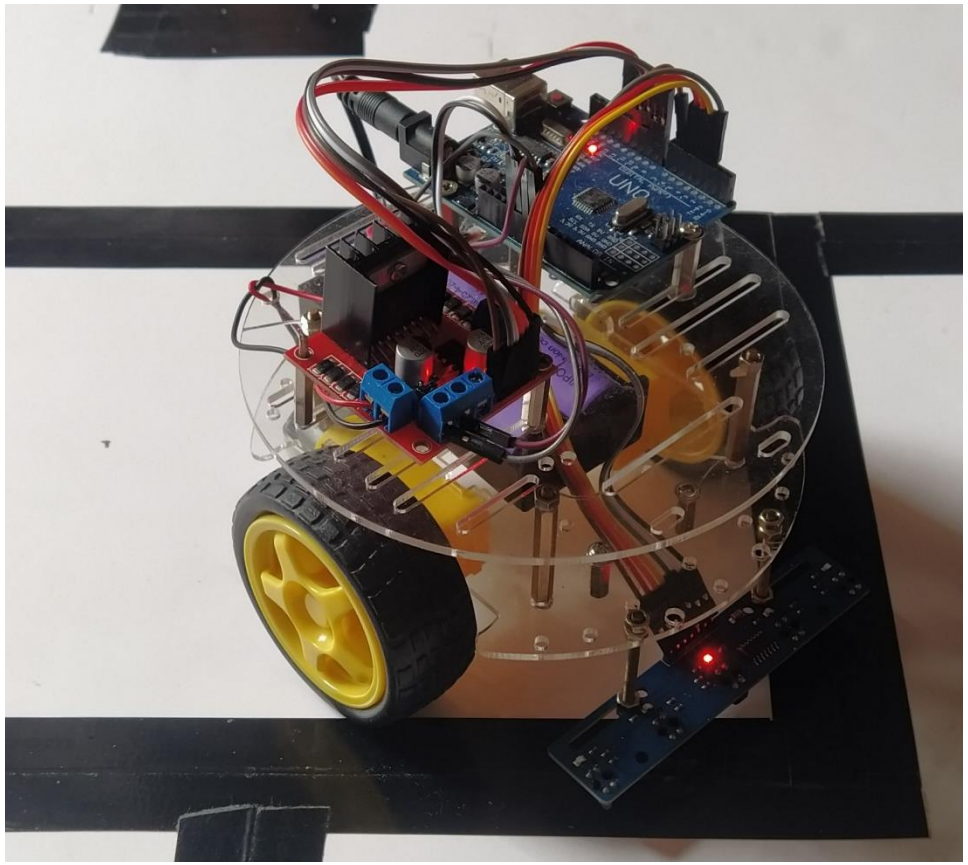
HOLOWORLD



Agenda – Day 4

Maze solving Robot

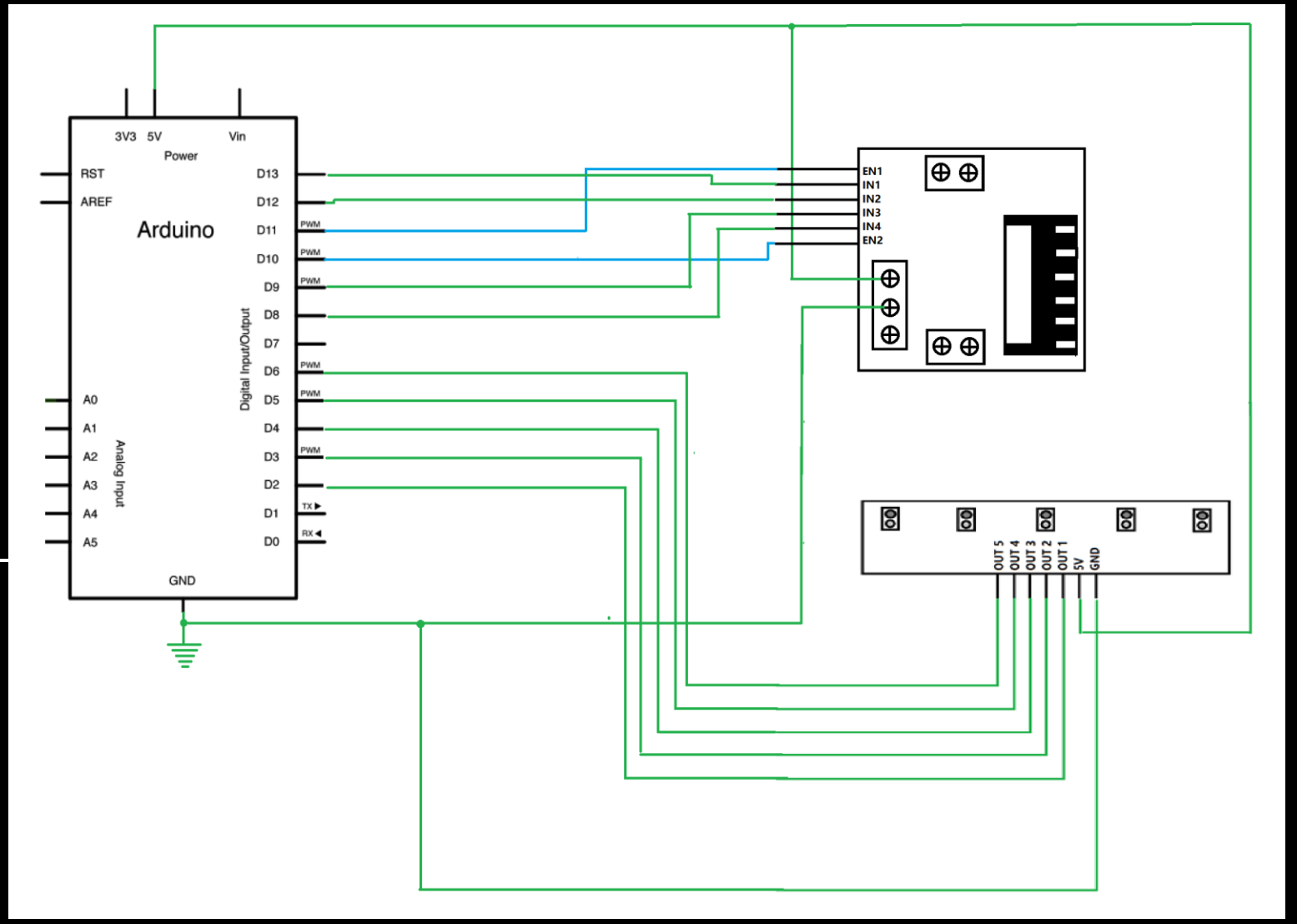
- How to code the Line maze
- Solving the maze using Left and right rules
- How to find the shortest path



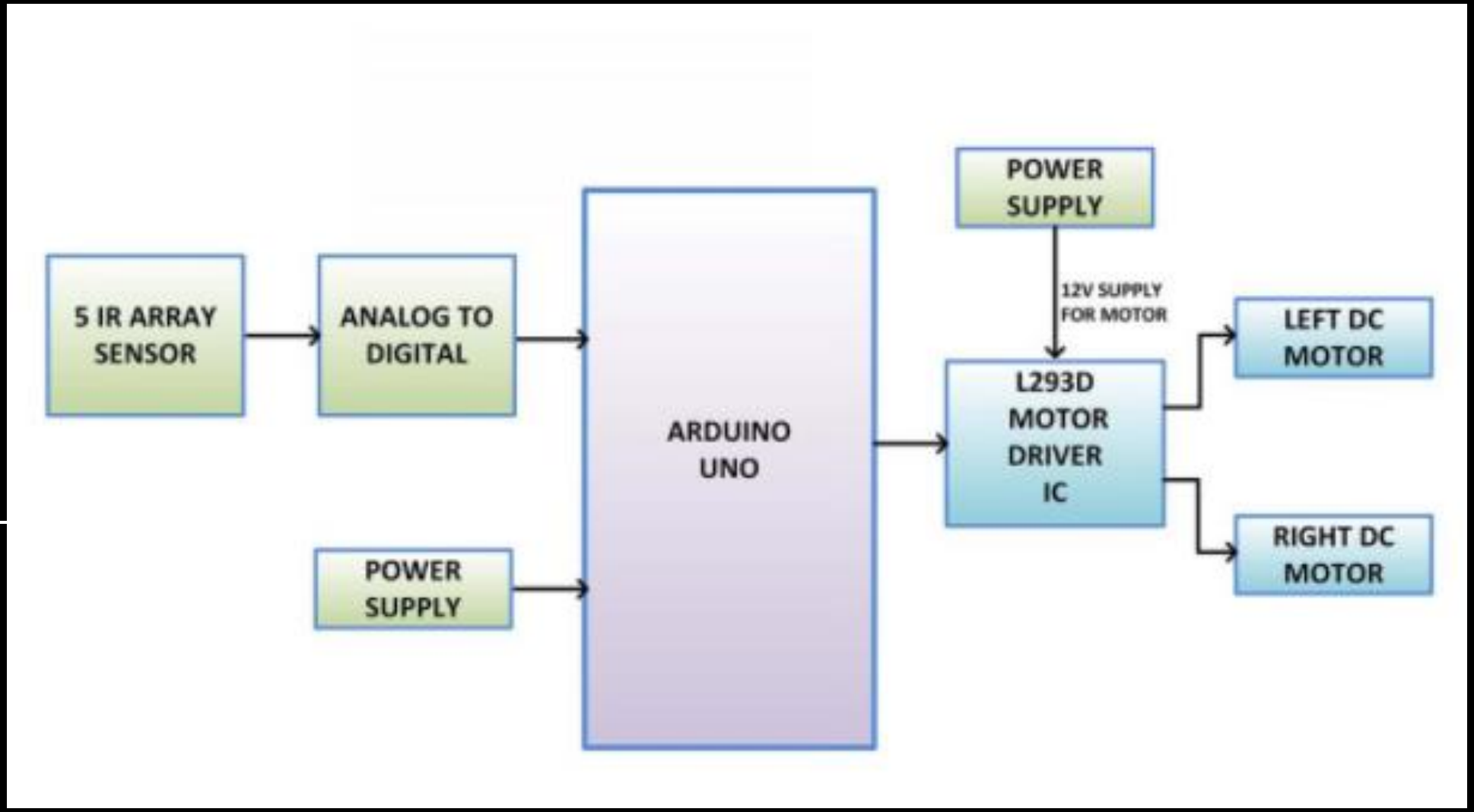
HOLOWORLD

Wednesday, December 15, 2021

Line Following Robot - Circuit Diagram



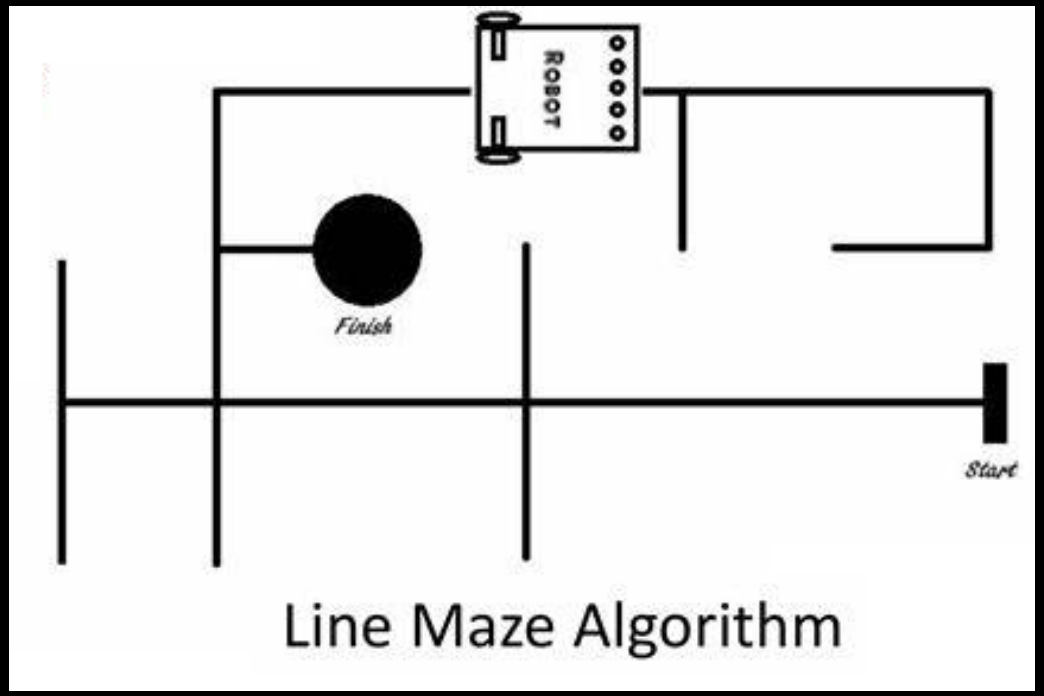
Line Following Robot - block Diagram



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

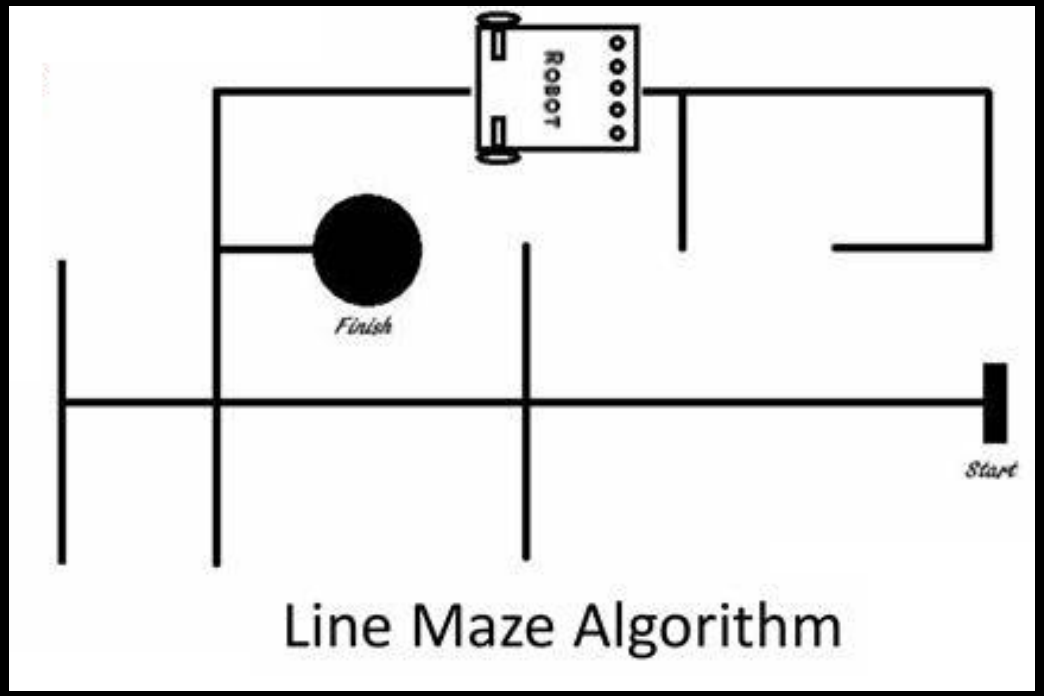
  If the maze has no loops, this will always get you to the end of the maze



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

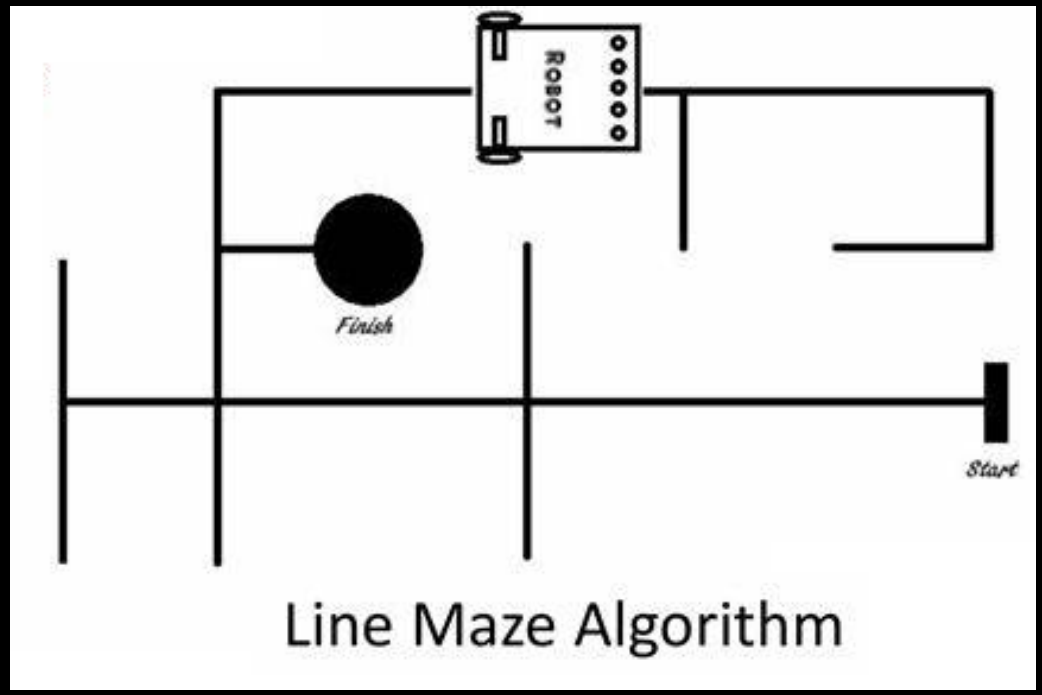
  If the maze has no loops, this will always get you to the end of the maze



Which rule do we use???

It really doesn't matter

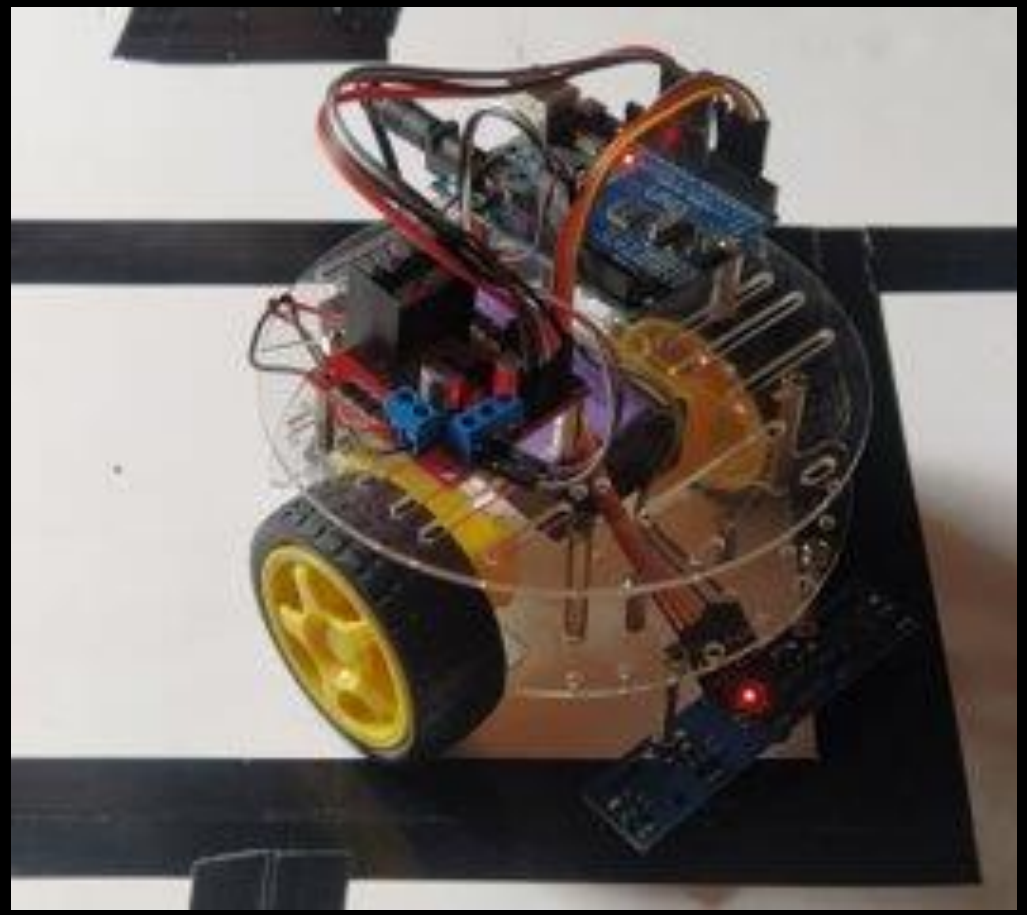
- Both the left-hand and the right-hand rules will get you to the end of the simple maze
- Which you select is purely a matter of personal preference
- Just pick one and be consistent

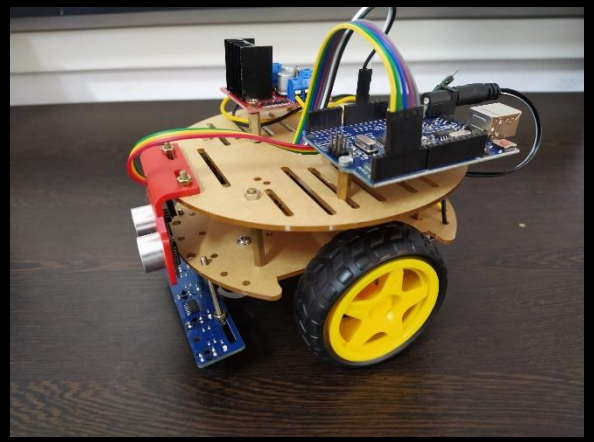


Line Maze Algorithm

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.

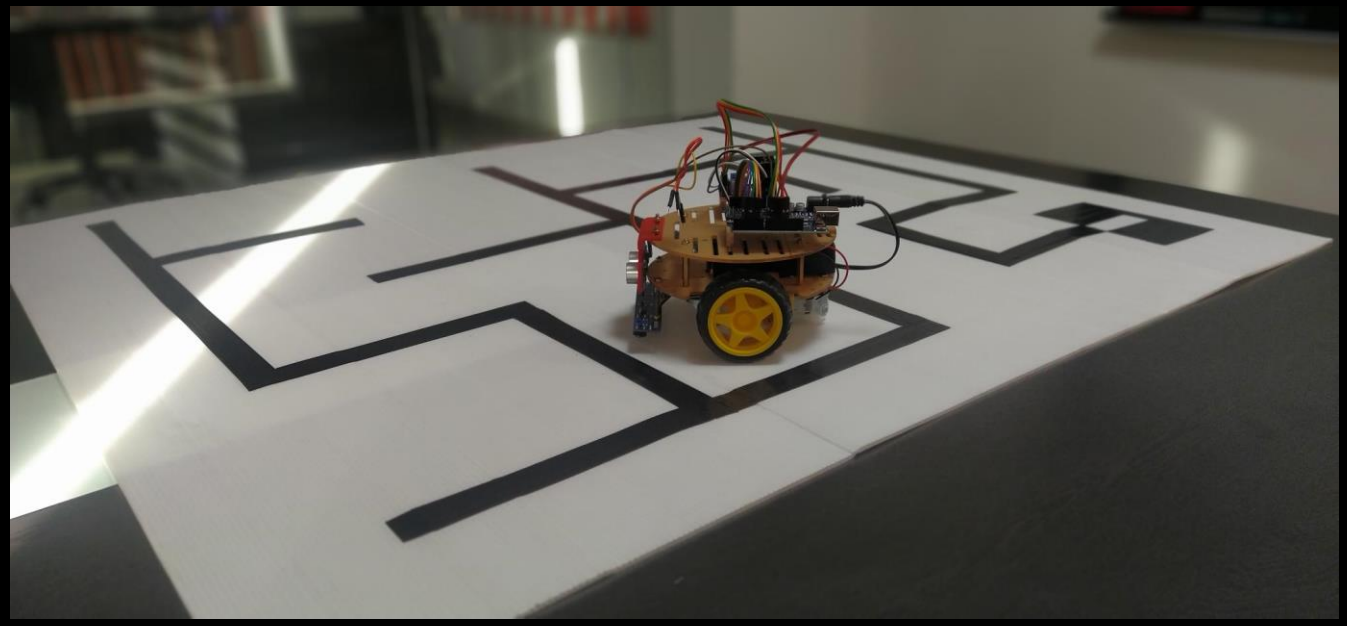




How robot behaviours?

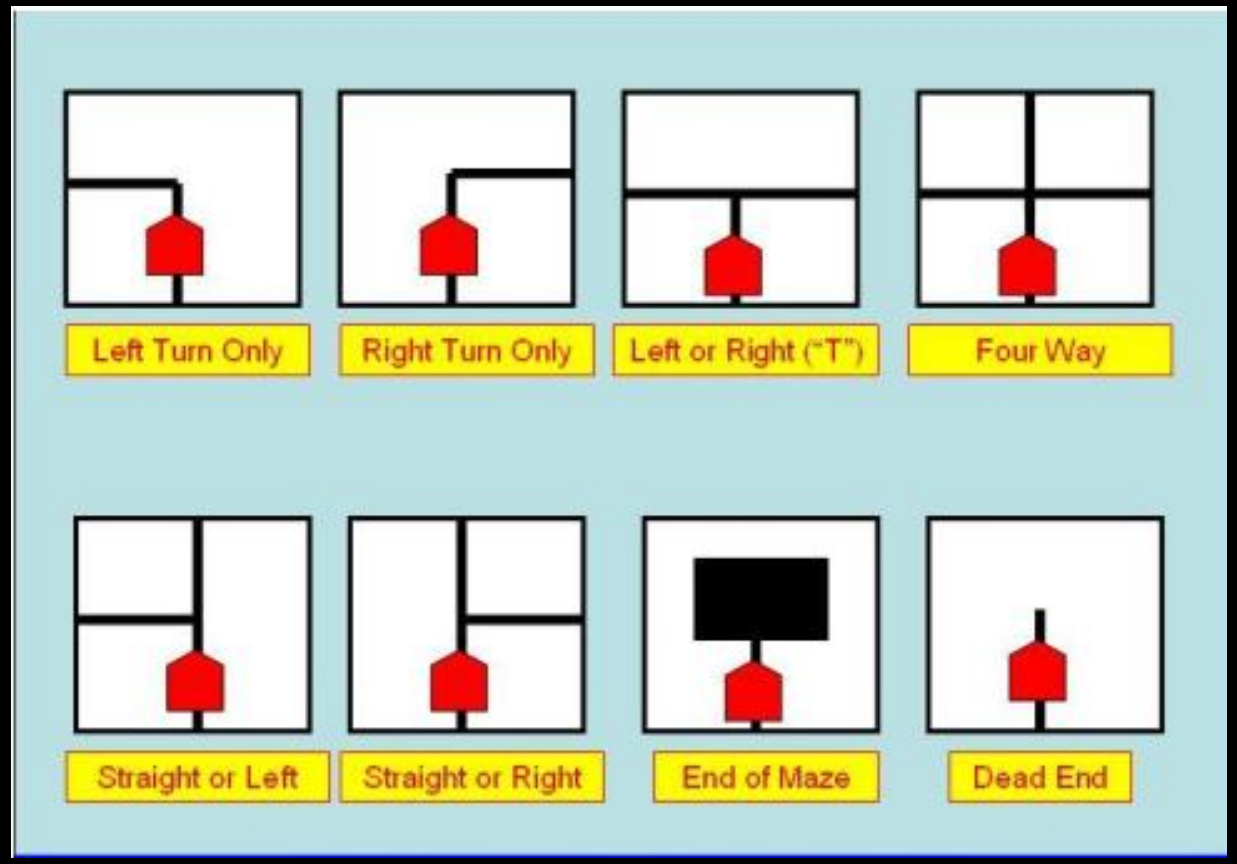
- The robot most of the time, will be involved in one of the following behaviors
  - Following the line. Looking for the next intersection
  - At an intersection, deciding what types of intersection it is
  - At an intersection, making a turn

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



Intersection and turn handling

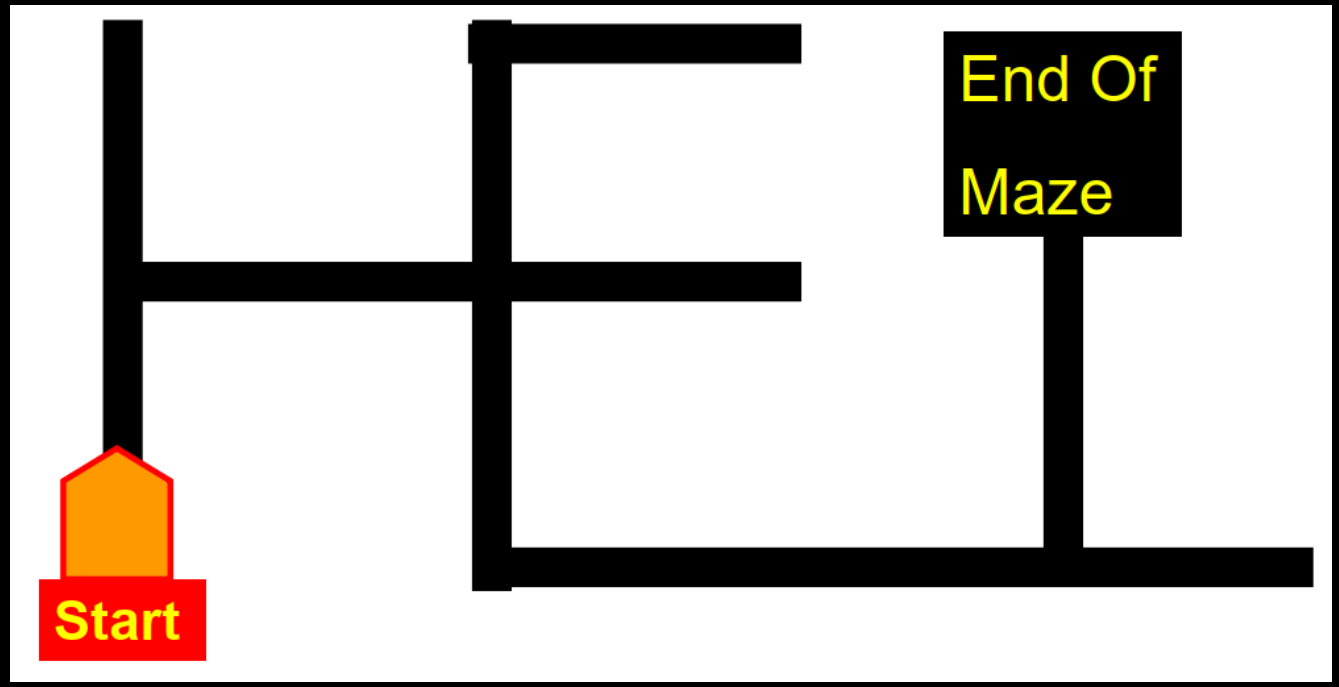
- The robot needs to be taught the correct behavior depending on the type of turn or intersection it encounters
- First let us give a more rigid definition of an intersection. I will define an intersection as " a place where the robot has more then one choice of direction"



The main Algorithm

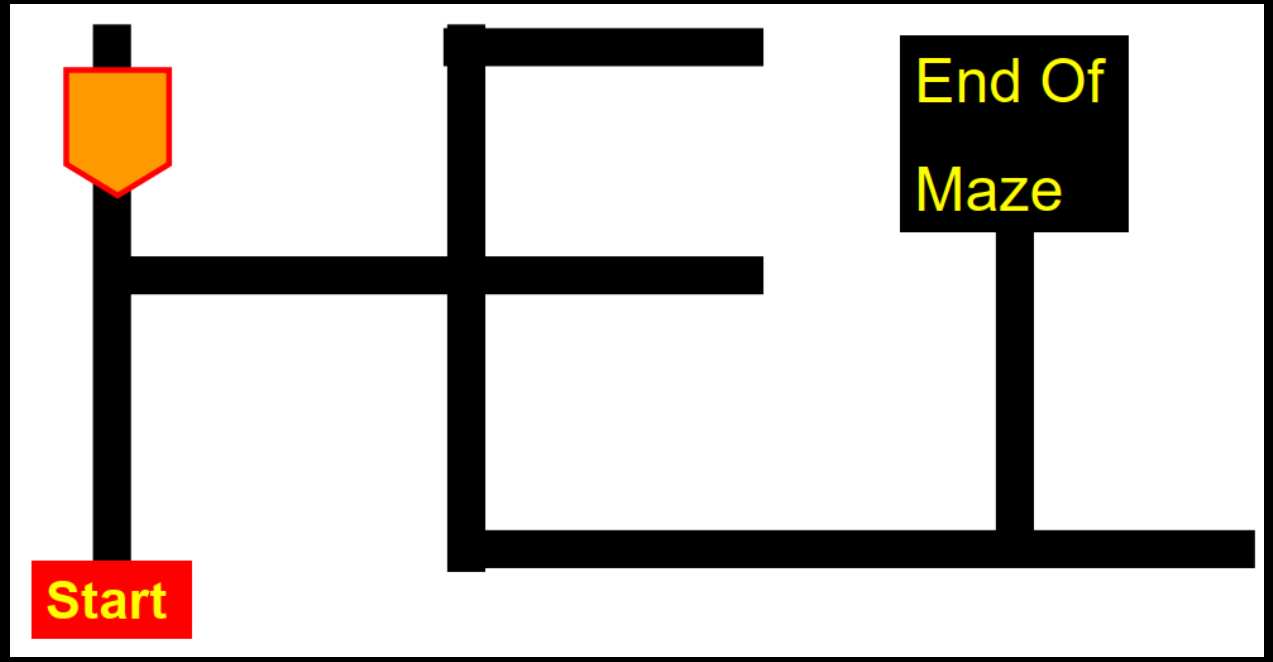
- First, we start with a very simple maze
- We will walk through this maze step-by-step to explain the maze solving algorithm
- So, we will use the left-hand rule for the first pass through the maze.
- There will be several dead-ends encountered and we need to store what we did so the correct path can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

Assume that we are using left-hand rule



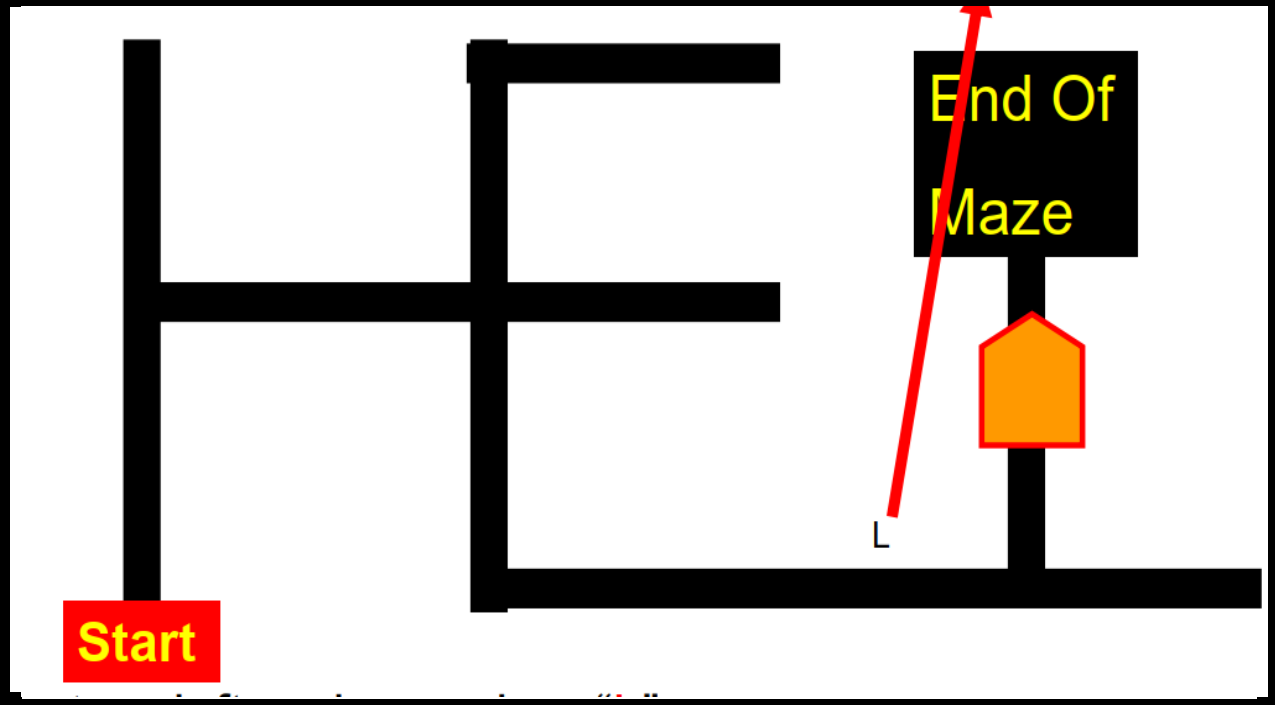
The main Algorithm

- 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

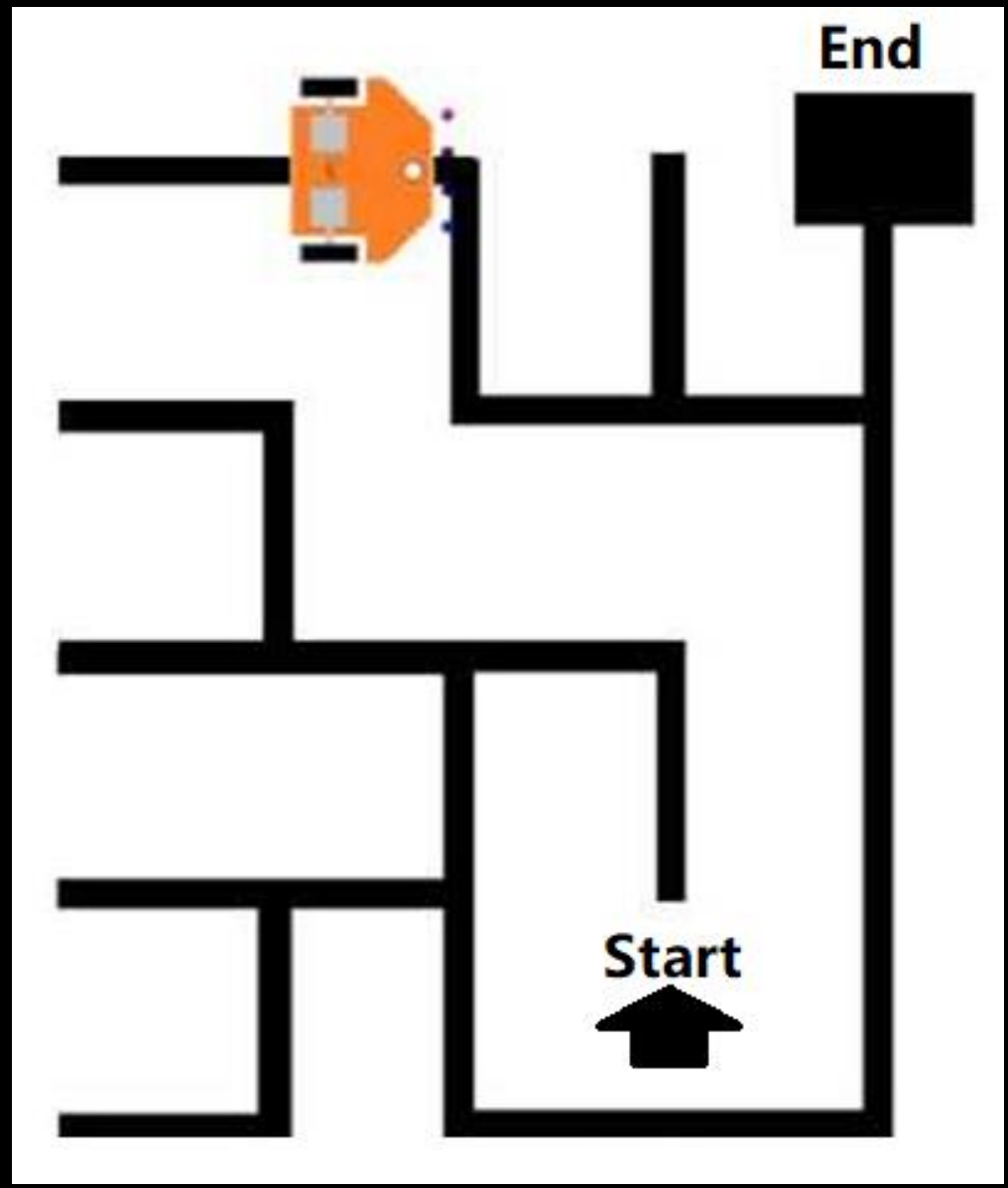
- The left hand-rule calls for a left turn, so take a left and record the turn FUL
- Then go forward and reaches the cross junction and take the left and record the turn FULL
- Since it is free right, it will not store the value and it takes U turn FULLU
- Since there is a dead end, it come back to same junction and stores FULLUL
- Robot reaches the dead-end and take a U-turn and stores FULLULU
- Again, come back to same junction and takes left turn and stores FULLULUL
- At last, it takes the Left turn and reaches the destination and stores FULLULULL

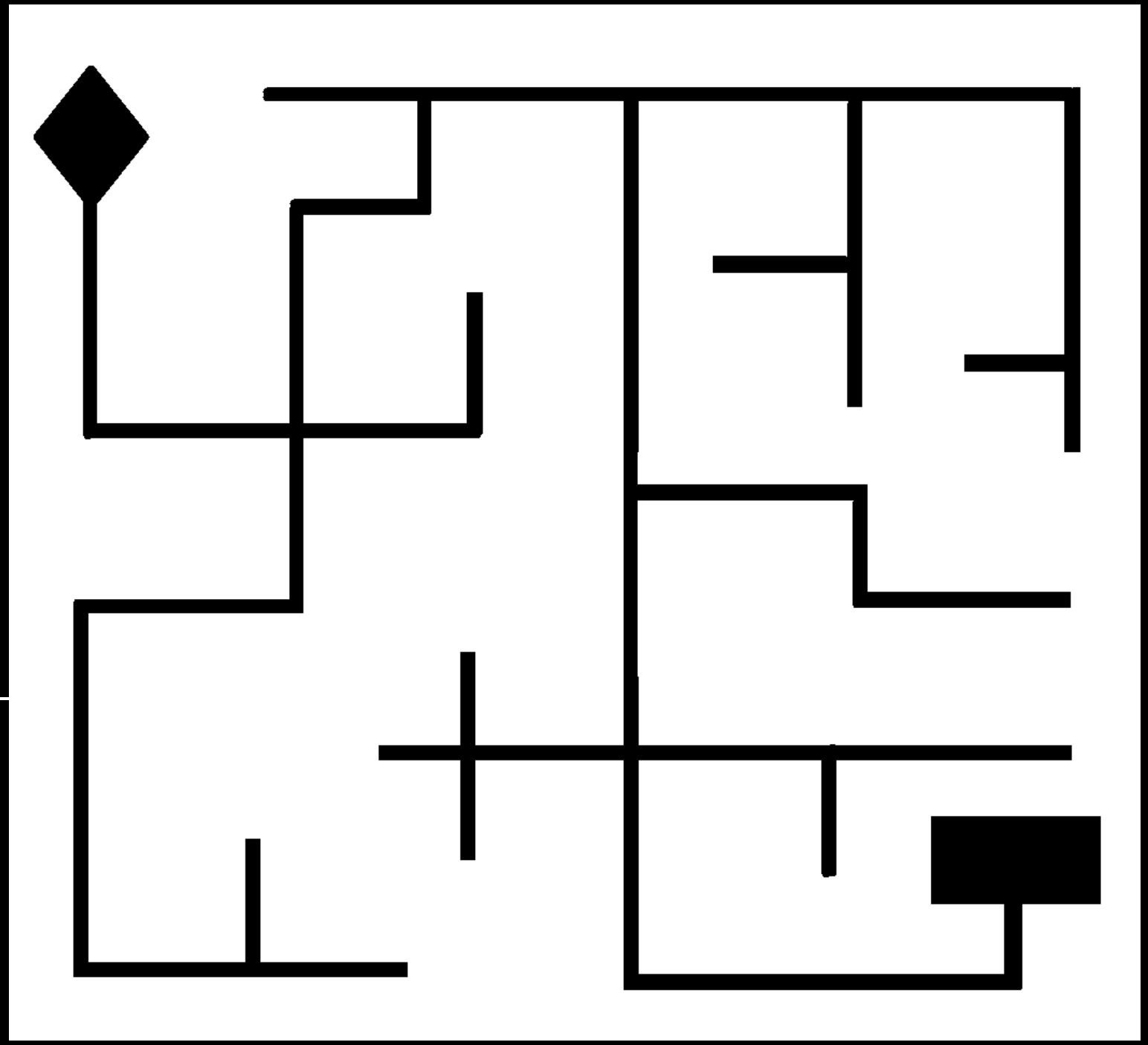


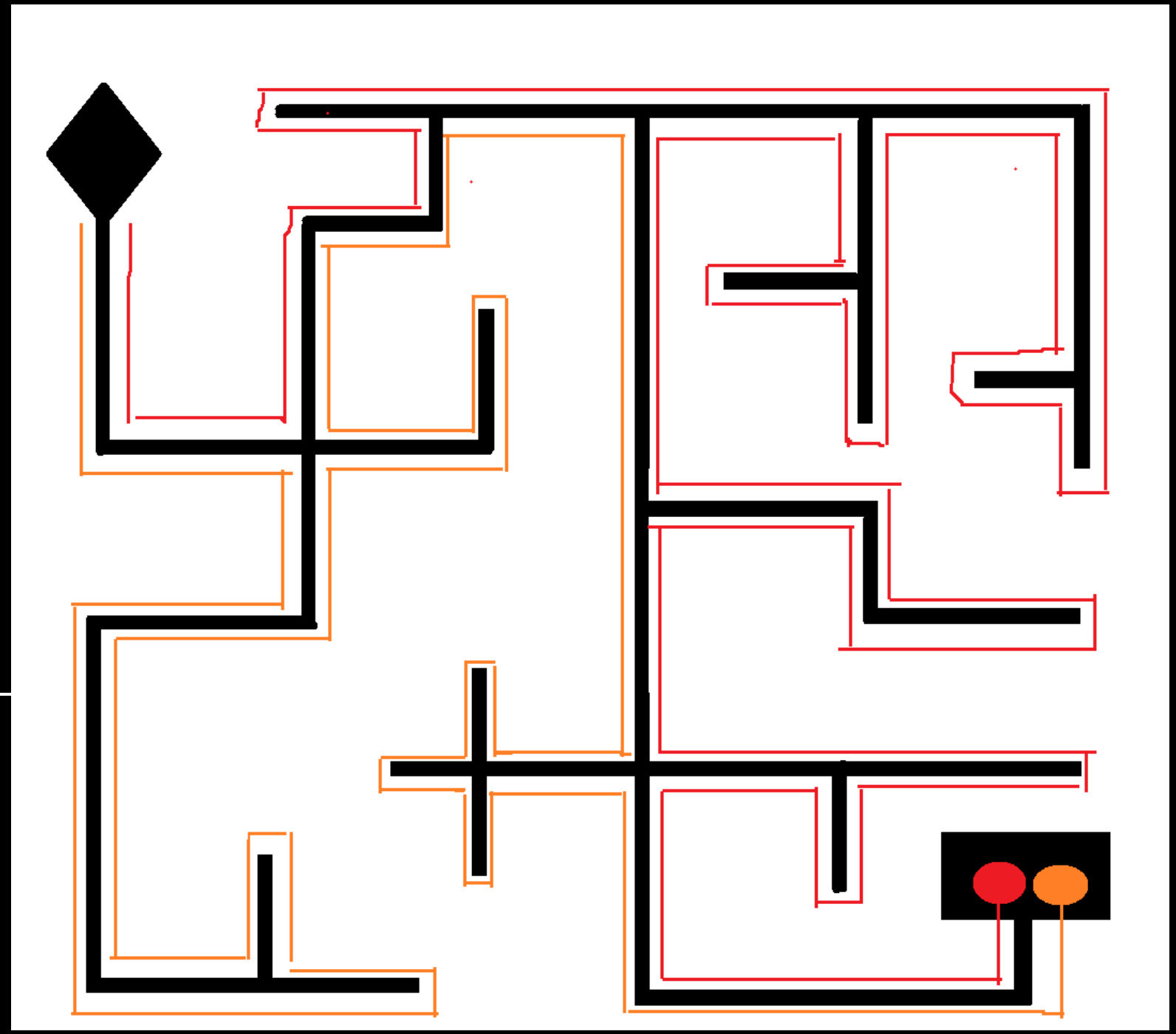
Simplifying the stored value

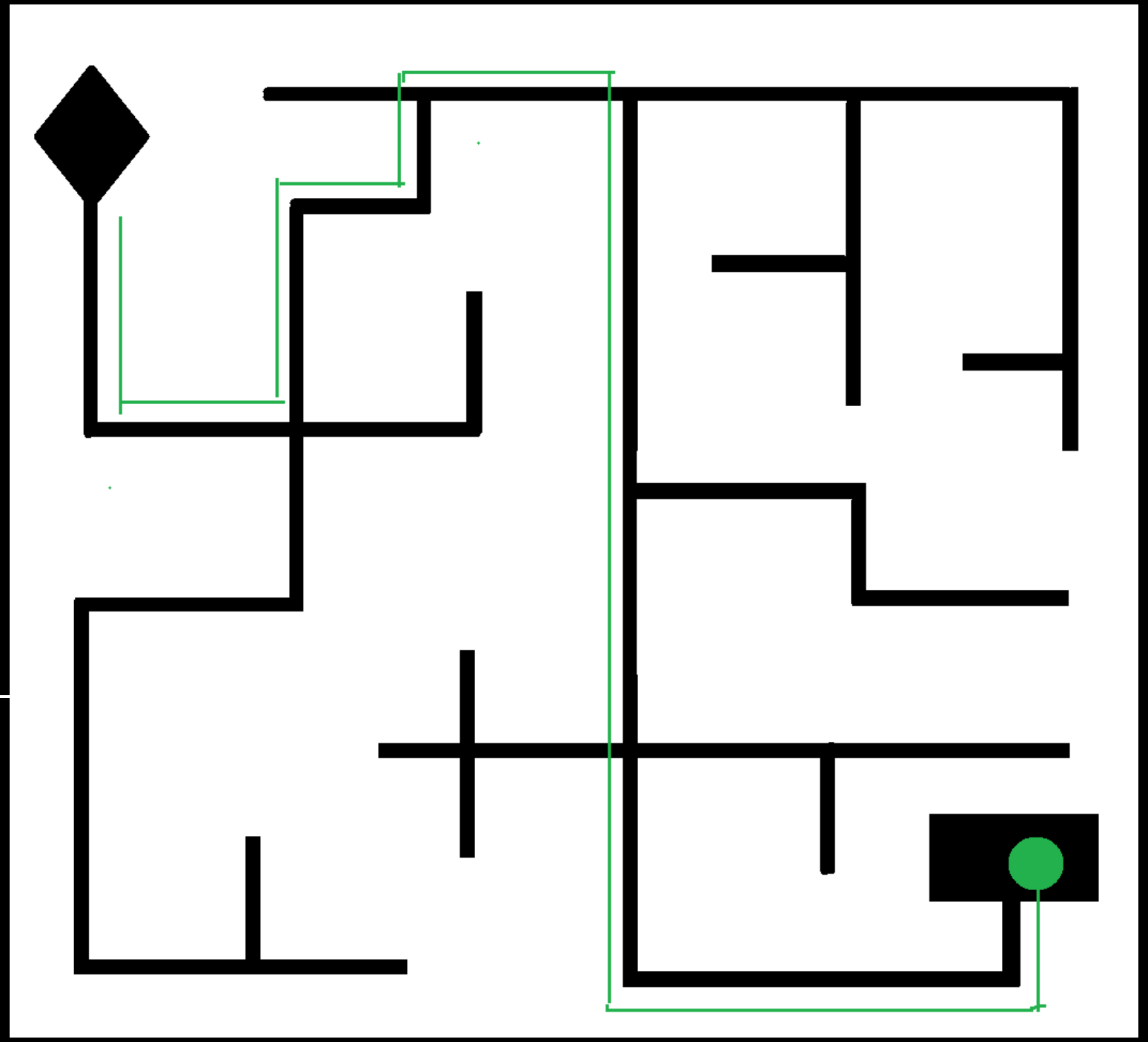
- Left – U turn – Right (LUR) = U (U turn)
- Left – U turn – Front (LUF) = R (Right)
- Right – U turn – Left (RUL) = U (U turn)
- Front – U turn – Left (FUL) = R (Right)
- Front – U turn – Front (FUF) = U (U turn)
- Left – U turn – Left (LUL) = F (Front)

FULLULULL = RRL









```
char path[] = "FFULULLLULULL";
int pathLength;

void setup()
{
    Serial.begin(9600);
    pathLength = strlen(path);
    while (CheckForUPath())
    {
        simplifyPath();
        for (int i = 0; i < pathLength; i++)
        {
            Serial.println(path[i]);
        }
        CheckForUPath();
    }
}



void ShiftPath(int i)
{
    for (int j = i; j < pathLength-2; j++)
    {
        path[j] = path[j + 2];
    }
    pathLength -= 2;
}
```

```
bool CheckForUPath()
{
    bool IsUPresent = false;
    for (int j = 0; j < pathLength; j++)
    {
        if (path[j] == 'U')
        {
            IsUPresent =true;
            break;
        }
        else
            continue;
    }
    return IsUPresent;
}




void simplifyPath()
{
    if (pathLength < 3 )
    {
        return;
    }

    int i;
    for ( i = 0; i < pathLength; i++)
    {
        if (path[i] == 'U')
        {
            if (path[i - 1] == 'F' && path[i + 1] == 'L')
            {
                path[i - 1] = 'R';
                ShiftPath(i);
            }
            else if (path[i - 1] == 'F' && path[i + 1] == 'F')
            {
                path[i - 1] = 'U';
                ShiftPath(i);
            }
            else if (path[i - 1] == 'L' && path[i + 1] == 'L')
            {
                path[i - 1] = 'F';
                ShiftPath(i);
            }
            else if (path[i - 1] == 'R' && path[i + 1] == 'L')
            {
                path[i - 1] = 'U';
                ShiftPath(i);
            }
            else if (path[i - 1] == 'L' && path[i + 1] == 'F')
            {
                path[i - 1] = 'R';
                ShiftPath(i);
            }

            else if (path[i - 1] == 'L' && path[i + 1] == 'R')
            {
                path[i - 1] = 'U';
                ShiftPath(i);
            }
            break;
        }
        else
        {
            continue;
        }

    }

}
```

The main Code-1

```
//**********Motor driver pins**********//
const int Motor_A1 7  //Right Motor MA1
const int Motor_A2 8  //Right Motor MA2
const int Motor_B1 12  //Left Motor MB1
const int Motor_B2 13  //Left Motor MB2
const int ENA 10 //Lef uut Motor Enable Pin EB
const int ENB 11 //Left Motor Enable Pin EB
//**********5 Channel IR Sensor Connection**********//
const int IR_Sensor1 = 6
const int IR_Sensor2 = 5
const int IR_Sensor3 = 4
const int IR_Sensor4 = 3
const int IR_Sensor5 = 2
//************************************************//
```

```
void setup()
{
  Serial.begin(9600);
  pinMode(Motor_A1, OUTPUT);
  pinMode(Motor_A2, OUTPUT);
  pinMode(Motor_B1, OUTPUT);
  pinMode(Motor_B2, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(IR_Sensor1, INPUT);
  pinMode(IR_Sensor2, INPUT);
  pinMode(IR_Sensor3, INPUT);
  pinMode(IR_Sensor4, INPUT);
  pinMode(IR_Sensor5, INPUT);
  delay(1000);
}
```

```
void sensor_check()
{
    Status_Sensor1 = digitalRead(IR_Sensor1);  //Left Most Sensor
    Status_Sensor2 = digitalRead(IR_Sensor2);  //Left Sensor
    Status_Sensor3 = digitalRead(IR_Sensor3);  //Middle Sensor
    Status_Sensor4 = digitalRead(IR_Sensor4);  //Right Sensor
    Status_Sensor5 = digitalRead(IR_Sensor5);  //Right Most Sensor
}
```

The main Code-1

```
void forward()
{
    digitalWrite(Motor_A1, HIGH);
    digitalWrite(Motor_A2, LOW);
    digitalWrite(Motor_B1, HIGH);
    digitalWrite(Motor_B2, LOW);
}
void backward()
{
    digitalWrite(Motor_A1, LOW);
    digitalWrite(Motor_A2, HIGH);
    digitalWrite(Motor_B1, LOW);
    digitalWrite(Motor_B2, HIGH);
}
```

```
void right_90()
{
    digitalWrite(Motor_A1, LOW);
    digitalWrite(Motor_A2, LOW);
    digitalWrite(Motor_B1, HIGH);
    digitalWrite(Motor_B2, LOW);
}
void left_90()
{
    digitalWrite(Motor_A1, HIGH);
    digitalWrite(Motor_A2, LOW);
    digitalWrite(Motor_B1, LOW);
    digitalWrite(Motor_B2, LOW);
}
```

```
void right()
{
    digitalWrite(Motor_A1, LOW);
    digitalWrite(Motor_A2, HIGH);
    digitalWrite(Motor_B1, HIGH);
    digitalWrite(Motor_B2, LOW);
}
void left()
{
    digitalWrite(Motor_A1, HIGH);
    digitalWrite(Motor_A2, LOW);
    digitalWrite(Motor_B1, LOW);
    digitalWrite(Motor_B2, HIGH);
}
```

```
void U_turn()
{
    digitalWrite(Motor_A1, LOW);
    digitalWrite(Motor_A2, HIGH);
    digitalWrite(Motor_B1, HIGH);
    digitalWrite(Motor_B2, LOW);
}
void simply_stop()
{
    digitalWrite(Motor_A1, LOW);
    digitalWrite(Motor_A2, LOW);
    digitalWrite(Motor_B1, LOW);
    digitalWrite(Motor_B2, LOW);
}
```

The main Code-1

```
sensor_check();
if((Status_Sensor1 == 1) && (Status_Sensor2 == 1) && (Status_Sensor3 == 0) && (Status_Sensor4 == 1) && (Status_Sensor5 == 1))
{
  forward();
}
if((Status_Sensor1 == 1) && (Status_Sensor2 == 0) && (Status_Sensor3 == 0) && (Status_Sensor4 == 1) && (Status_Sensor5 == 1))
{
  left();
}
if((Status_Sensor1 == 1) && (Status_Sensor2 == 1) && (Status_Sensor3 == 0) && (Status_Sensor4 == 0) && (Status_Sensor5 == 1))
{
  right();
}
```

The main Code-2

```
void setup()
{
  Serial.begin(9600);
  pinMode(Motor_A1, OUTPUT);
  pinMode(Motor_A2, OUTPUT);
  pinMode(Motor_B1, OUTPUT);
  pinMode(Motor_B2, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(IR_Sensor1, INPUT);
  pinMode(IR_Sensor2, INPUT);
  pinMode(IR_Sensor3, INPUT);
  pinMode(IR_Sensor4, INPUT);
  pinMode(IR_Sensor5, INPUT);
  delay(2000);
}
  int sensor[5] = {0, 0, 0, 0, 0};
  int error;
```

```
void error_check()
{
    if ((sensor[0] == 1) && (sensor[1] == 0) && (sensor[2] == 1) && (sensor[3] == 1) && (sensor[4] == 1))
    error = -2;
    else if ((sensor[0] == 1) && (sensor[1] == 0) && (sensor[2] == 0) && (sensor[3] == 1) && (sensor[4] == 1))
    error = -1;
    else if ((sensor[0] == 1) && (sensor[1] == 1) && (sensor[2] == 0) && (sensor[3] == 1) && (sensor[4] == 1))
    error = 0;
    else if ((sensor[0] == 1) && (sensor[1] == 1) && (sensor[2] == 0) && (sensor[3] == 0) && (sensor[4] == 1))
    error = 1;
    else if ((sensor[0] == 1) && (sensor[1] == 1) && (sensor[2] == 1) && (sensor[3] == 0) && (sensor[4] == 1))
    error = 2;
    else if ((sensor[0] == 0) && (sensor[1] == 0) && (sensor[2] == 1) && (sensor[3] == 0) && (sensor[4] == 0))
    error = 100;
    else if ((sensor[0] == 0) && (sensor[1] == 0) && (sensor[2] == 0) && (sensor[4] == 1)) // (sensor[3] == 1 / 0)
    error = 101;
    else if ((sensor[0] == 1) && (sensor[2] == 0) && (sensor[3] == 0) && (sensor[4] == 0)) // (sensor[1] == 1 / 0)
    error = 102;
    else if ((sensor[0] == 0) && (sensor[1] == 0) && (sensor[2] == 0) && (sensor[3] == 0) && (sensor[4] == 0))
    error = 103;
    else if ((sensor[0] == 1) && (sensor[1] == 1) && (sensor[2] == 1) && (sensor[3] == 1) && (sensor[4] == 1))
    error = 104;
    else if ((sensor[0] == 1) && (sensor[2] == 0) && (sensor[4] == 1))
    error = 105;
}
```

The main Code-2

```
if (error == 0)
{
  forward();
}
```

```
else if (error == -2 || error == -1) // 11001 || 11101 - free left
{
  do
  {
    sensor_check();
    error_check();
    left_90();
  } while (error != 0);
}
```

```
else if(error == 2 || error == 1) // 10011 || 10011 - free right
{
  do
  {
    sensor_check();
    error_check();
    right_90();
  } while (error != 0);
}
```

The main Code-2

```
if (error == 0)
{
  forward();
}
```

```
else if (error == -2 || error == -1) // 11001 || 11101 - free left
{
  do
  {
    sensor_check();
    error_check();
    left_90();
  } while (error != 0);
}
```

```
else if(error == 2 || error == 1) // 10011 || 10011 - free right
{
  do
  {
    sensor_check();
    error_check();
    right_90();
  } while (error != 0);
}
```

```
else if(error == 104) // 11111 - U turn
 {
   simply_stop();
   delay(1000);
   do
   {
       sensor_check();
       error_check();
       left();
   } while (error != 0);
   delay(1000);
 }
```

ALL THE BEST