Milestone 3

Tareeq Saley

SLYTAR001

---

1. Subsystem Description

1.1 SYSTEMIC (ALGORITHMIC)

The requirements of the robot design and demo involves the designed robot having to be able to solve an unknown maze, from a random starting position. The maze-solving algorithm should not be hard-coded but should allow for the maze to be learnt by the robot in its first traversal through the maze. The robot should then use this information to calculate the shortest possible path from the starting point until the finishing point. Initially, a breadth first search algorithm was designed, and the entire maze simulated in Python in order to test the validity of the algorithm and whether it would achieve the required results. The implemented algorithm, however, used a tailored wall-follower algorithm with left priority directive. Because of time constraints and the sheer complexity of implementing a breadth first search algorithm in real life, no further algorithm was developed and implemented to find the shortest path.

2. Logical Testing of Subsystem and System Level Testing

2.1 SUBSYSTEM ALGORITHMIC TESTING PROCEDURE

To evaluate whether a breadth-first search algorithm would actually work for the given task, the entire maze was recreated and simulated through python using the algorithm. A left priority directive was used for the implementation and multiple starting points were assigned for testing purposes. After finding the maze, the path produced by the algorithm would then be checked manually to see if it was in fact shorter than any other path. This algorithm broke down the maze into blocks and used position tracking to store its relative location at any given point. At every intersection, every turn that the robot made, together with its respective location at that point would be stored in a stack and every turn that led to the robot reaching a dead-end with its respective location was popped off the stack. The robot would analyse all possible paths of every intersection (parent node) before analysing paths that branched from them (child nodes), and in that way would be guaranteed to find the shortest possible path. Storing and tracking the position of the robot's location was fundamental in ensuring that the robot does that stay in a loop at any point. The algorithm can be found in appendix 1.

2.2 SUBSYSTEM ALGORITHMIC TESTING RESULTS

The specified algorithm that was suggested to be used in Milestone 2 was used for the simulated maze and produced the shortest possible path every single time for any random starting (and even ending) point. The contents of the stack were displayed on the screen for the user to verify the

path. An analysis of the elements stored in the stack throughout the algorithm's implementation, together with the final path found can be seen in Appendix 2.

2.3 SYSTEM TESTING PROCEDURE

Since the robot that was used for implementation purposes used an Arduino microcontroller, the Arduino IDE was used to develop the algorithm used to solve the maze. Initially, the intention was to first get the robot to follow a line well enough, and to configure and calibrate the sensors well enough to make the right decisions at various intersections in the maze. For simplicity, a wall follower algorithm with left priority directive was used for the maze solving algorithm. To get the maze out of a loop, for every 3 left consecutive left that it made, it would either go straight or turn right at the next intersection. This algorithm took a while to perfect, and because of time constraints could not be optimized or replaced with a breadth first search algorithm to find the shortest possible path and was hence used for both attempts in the demo. This algorithm can be found in Appendix 3.

2.4 SYSTEM TESTING RESULTS

The robot was able to sense the black line accurately and would readjust itself to maintain its presence on the black line whenever it was leaning off track. The robot was also able to complete the maze, and find and stop on the black dot in a relatively quick time. However, during its traversal through the maze, and specifically after some turns, it veered off track to the extent where it would lose sight of the line completely and turn repetitively until it was back on track. This was because if it wasn't aligned well enough on the line before taking a turn, it would overcompensate for turns causing either the leftmost/rightmost sensor to trigger and the robot to think that it needs to turn once more. A more detailed analysis of this problem can be found in the algorithm commentary.

The same algorithm was used for the second demo attempt and hence did not produce a shorter path.

3. Commentary on the Algorithmic Design

3.1 ORIGINALLY PROPOSED ALGORITHM IMPLEMENTATION

The robot would initially follow a default-left turn pattern according to a predefined priority list: Left, Straight, Right, Back. It would traverse the maze from start to finish according to this left-priority directive. In the advent where it had made 3 consecutive left or right turns, it was designed to overlook the next left or right turn respectively and evaluate the remaining open paths from the list instead. The robot, using this algorithm, would not traverse the entire maze and find a shortest possible path, but was guaranteed to find the end of the maze.

The optimization algorithm that was intended to be used involved having decisions made by the robot at every intersection of the maze saved onto a stack and having those decisions that led to a dead-end popped off the stack through its traversal. In that way this would allow for the robot to find a shorter (and, possibly the shortest) path to the end of the maze. This could not be completed in time and was not implemented.

3.2 DESIGN MODIFICATIONS

The main issue that made coding a breadth first search algorithm from the off terribly complex, was the fact that the robot had all 6 of its sensors lined up in a straight line. This sensor configuration

would work wall for a line following but becomes a hassle to deal with when implementing an algorithm to solve a maze. This is because it requires that the robot reverses for a bit for every turn that it has to make, this meant that the development of an optimized breadth first search algorithm became nearly impossible. This was because a lot of unnecessary reverses were made for each turn that could not be compensated/accounted for in the stack well enough, and led to the stack maintaining an inaccurate set of decisions to find the shortest path. A more simplified algorithm was then developed and used in implementation to cater for the time constraints and to allow the focus to be shifted on getting the robot to find the end of the maze first and foremost. Time was also dedicated into refining and calibrating sensors and turning speeds/angles, as well as line following accuracy, to ensure that the robot does not lose sight of the line and end up not completing the maze at all.

3.3 LESSONS LEARNED

The main lesson learned during the demo was that not enough attention was given to turn refinement. This was crucial in keeping the robot on the line throughout its traversal through the maze. First and foremost, it was essential to keep the robot aligned perpendicularly to the line, and had it being able to maintain that perpendicular alignment well enough, and have its alignment readjusted quickly enough once it was veering to the left or right, the chances of a turn going wrong would be minimized. Also, had the sensors been repositioned, it would allow for left and right turns to be made without the robot first having to reverse, this would increase stability and again lower the risk of the robot turning 90 degrees when it's not aligned perpendicularly to the line. Lastly, and probably most importantly, by decreasing the positional shift of the robot when it moves forward, and forcing it to take smaller steps forward, it would be able to detect turns with more consistency (i.e, detecting that a turn needs to be made at the start of an intersection and not when its already halfway through it). This would ensure that the robot always has the line right across its centre and would minimize the need for it readjust itself back onto the line, hence decreasing the risk of it going off track or turning too much or too little.

3.4 THINGS DONE RIGHT

The decision to compromise on the more complex algorithm and rather focus on producing a working robot that solves a maze, regardless of the path it takes, proved to be a good one. This ensured that the task was able to be completed on time.

The robot did fantastically well to readjust itself back onto the line whenever it was veering off track, be it to the left or the right both while it was moving forward or in reverse. The robot also did really well to get back on track and complete the maze even if it happened to lose sight of the black line for a while.

Good coding practice was also used by ensuring that the code was tested and debugged incrementally, and that it was tested after every single change and not after multiple changes so as to prevent having to not know what to debug if issues came about because of those changes.

The value used to differentiate the black line from its white surroundings worked remarkably well, and because of that, there was no issues with the sensors detecting black lines and the times at which they did so.

Appendix 1

```
import queue

orientation = 0;

def createMaze():
    maze = []
    maze.append(["#", "#", "#", "#", "#", "O", "#"])
    maze.append(["#", " ", " ", " ", "#", " ", "#"])
    maze.append(["#", " ", "#", " ", "#", " ", "#"])
    maze.append(["#", " ", "#", " ", " ", " ", "#"])
    maze.append(["#", " ", "#", "#", "#", " ", "#"])
    maze.append(["#", " ", " ", " ", "#", " ", "#"])
    maze.append(["#", "#", "#", "#", "#", "X", "#"])

    return maze


def createMaze2():
    maze = []
    maze.append(["#", "#","#", "#", "#", "#", "#", "#", "#", "#", "#"])
    maze.append(["#", " ","#", " ", "#", "#", "#", "O", "#", " ", "#"])
    maze.append(["#", "#","#", " ", "#", " ", " ", " ", " ", " ", "#"])
    maze.append(["#", "#","#", " ", "#", " ", "#", "#", " ", "#", "#"])
    maze.append(["#", "#","#", " ", " ", " ", "#", " ", "X", " ", "#"])
    maze.append(["#", "#","#", " ", "#", "#", "#", " ", "#", " ", "#"])
    maze.append(["#", " ","#", " ", " ", " ", " ", " ", " ", " ", "#"])
    maze.append(["#", "#","#", " ", "#", " ", "#", " ", "#", " ", "#"])
    maze.append(["#", " ","#", " ", "#", " ", "#", "#", "#", " ", "#"])
    maze.append(["#", "#","#", " ", "#", " ", " ", " ", " ", "#", "#"])
    maze.append(["#", "#","#", "#", "#", "#", "#", "#", "#", "#", "#"])

    return maze


def printMaze(maze, path=""):
    for a, row in enumerate(maze):
        for b, col in enumerate(row):
            if col == "X":
                i = b
                j = a


    pos = set()
    for move in path:
        if ( move == "L"):
            i -= 1


        elif ( move == "S"):
            j -= 1


        elif (move == "R"):
            i += 1

        pos.add((j, i))

    for j, row in enumerate(maze):
        for i, col in enumerate(row):
 #          if (j, i) in pos:
 #              print("+ ", end="")
  #         else:
               print(col + " ", end="")
        print()


def valid(maze, moves):
    for a, row in enumerate(maze):
        for b, col in enumerate(row):
            if col == "X":
                i = b
                j = a

    orientation = 0
    for move in moves:
        if (orientation == 0 ):
            if ( move == "L" ):
                i -= 1
                orientation = 90

            elif ( move == "S"):
                j -= 1
                orientation = 0

            elif (move == "R"):
                i += 1
                orientation = 270


        elif (orientation == 90):
```

```
            if (move == "L"):
                j += 1
                orientation = 180

            elif (move == "S"):
                i -= 1
                orientation = 90

            elif (move == "R"):
                j -= 1
                orientation = 0


        elif (orientation == 180):
            if ( move == "L" ):
                i += 1
                orientation = 270

            elif ( move == "S"):
                j += 1
                orientation = 180

            elif (move == "R"):
                i -= 1
                orientation = 90


        elif (orientation == 270):
            if (move == "L"):
                j -= 1
                orientation = 0

            elif (move == "S"):
                i += 1
                orientation = 270

            elif (move == "R"):
                j += 1
                orientation = 180



        if (maze[j][i] == "#"):
            return False


    return True


def findEnd(maze, moves):
    for a, row in enumerate(maze):
        for b, col in enumerate(row):
            if col == "X":
                i = b
                j = a
    #print(moves)

    orientation = 0
    for move in moves:
        if (orientation == 0):
            if ( move == "L" ):
                i -= 1
                orientation = 90

            elif ( move == "S"):
                j -= 1
                orientation = 0

            elif (move == "R"):
                i += 1
                orientation = 270
            #print (j,i)

        elif (orientation == 90):
            if (move == "L"):
                j += 1
                orientation = 180

            elif (move == "S"):
                i -= 1
                orientation = 90

            elif (move == "R"):
                j -= 1
                orientation = 0
            #print(j, i)

        elif (orientation == 180):
            if ( move == "L" ):
                i += 1
                orientation = 270

            elif ( move == "S"):
```

```
            j += 1
            orientation = 180

        elif (move == "R"):
            i -= 1
            orientation = 90
        #print (j,i)

    elif (orientation == 270):
        if (move == "L"):
            j -= 1
            orientation = 0

        elif (move == "S"):
            i += 1
            orientation = 270

        elif (move == "R"):
            j += 1
            orientation = 180
        #print(j, i)
    #print(orientation)

  if maze[j][i] == "O":
    print("Found: " + moves)
    printMaze(maze, moves)
    return True

  return False


# MAIN ALGORITHM

nums = queue.Queue()
nums.put("")
add = ""
maze = createMaze2()

while not findEnd(maze, add):
  add = nums.get()
  print(add)
  for j in ["L", "S", "R"]:
    put = add + j
    if valid(maze, put):
      nums.put(put)
      #print(put)
```

Appendix 2

```
L
S
R
LL
SS
RR
LLS
SSL
SSR
RRS
LLSL
LLSS
LLSR
SSLS
SSLR
Found: SSLR
# # # # # # # # # #
#       # # # O #   #
# #     #           #
# #     #   # #   # #
# #         #   X   #
# #     # # #   #   #
#                   #
# #     #   #   #   #
#       #   # # #   #
# #     #       #   #
# # # # # # # # # # #
```

Appendix 3

```
#include <Servo.h>
#include<StackArray.h>
boolean l, r, s,rev;
Servo myservo1;
Servo myservo2;// create servo object to control a servo
// twelve servo objects can be created on most boards
int sensor;
int pos = 0;    // variable to store the servo position
int p = 2;
int pp = 2;
int ppp = 2;

int LeftSensor;
float LeftVoltage;
int MidSensor1;
float MidVoltage1;
int MidSensor2;
float MidVoltage2;
int RightSensor;
float RightVoltage;
int LF1;
float LF1V;
int LF2;
float LF2V;
StackArray <int> st;
StackArray <int> moves;
void setup() {
  Serial.begin(9600);
  s = true;
  l=false;
  rev = false;

  //myservo1.write(0);
  //myservo2.write(0);
}

void updateSensors()
{
  LeftSensor = analogRead(A5);
  LeftVoltage = LeftSensor * (5.0 / 1023.0);
  MidSensor1 = analogRead(A2);
  MidVoltage1 = MidSensor1* (5.0/1023.0);
  MidSensor2 = analogRead(A3);
  MidVoltage2 = MidSensor2* (5.0/1023.0);
```

```
 RightSensor = analogRead(A0);
 RightVoltage = RightSensor * (5.0 / 1023.0);
 LF1 = analogRead(A1);
 LF1V = LF1 * (5.0 / 1023.0);
 LF2 = analogRead(A4);
 LF2V = LF2 * (5.0 / 1023.0);
}

void loop()
{

bloop:

updateSensors();
//  if ((LeftVoltage < 0.5)&&(RightVoltage<0.5))
//  {
//  myservo1.attach(11);
//  myservo2.attach(12);
//  sensor =99;
//  straight();
//  delay(400);
//  LeftSensor = analogRead(A5);
//  LeftVoltage = LeftSensor * (5.0 / 1023.0);
//  RightSensor = analogRead(A0);
//  RightVoltage = RightSensor * (5.0 / 1023.0);
// // Serial.println(sensor);
//    if ((LeftVoltage < 0.5)&&(RightVoltage<0.5))
//    {
//      myservo1.attach(11);
//      myservo2.attach(12);
//      dot();
//
//    }
//    else
//    {
//    myservo1.attach(11);
//    myservo2.attach(12);
//    //sensor =99;
//    reverse();
//    delay(400);
//    }
//  }

  if (LeftVoltage < 1)
  {
```

```
l  = true;
sensor = 1;


}
else if ((RightVoltage <1)&&(s==false))
{
  sensor = 3;
  s == true;
}
else if (((MidVoltage1 < 1)||(MidVoltage2<1))||(LF1V<1)||(LF2V<1))
{
  if (l == false)
  {
    s = true;
    sensor = 2;
  }
}
else if (RightVoltage <1)
{
  sensor = 3;
  s == true;
}


else
{
  s = false;
  sensor = 4;
//  Serial.println(sensor);
}


if (((LF1V>1)&&(LF2V<1))&&(MidVoltage2>1)&&(sensor ==2))
{



  sensor = 5;
}


 if (((LF1V<1) &&(LF2V>1))&&(MidVoltage1>1)&&(sensor==2))
{


  sensor = 6;


}
//Serial.println(sensor);
//Serial.println(r);
// Serial.println(pp);
```

```
 // Serial.println(ppp);
  Serial.println(LeftVoltage);
  Serial.println(LF2V);
  Serial.println(MidVoltage2);
  Serial.println(MidVoltage1);
  Serial.println(LF1V);
  Serial.println(RightVoltage);
 // Serial.println(MidVoltage2);
if (sensor ==2)
{
  myservo2.attach(12);
  myservo1.attach(11);

  straight();
  r = false;
  delay(50);
  st.push(2);
  updateSensors();
//  if (LeftVoltage <1)
//  {
//    //Serial.println(LefrtVoltage);
//    //Serial.println(s);
//    myservo1.attach(11);
//    myservo2.attach(12);
//    left();
//    delay(400);
//    ppp = pp;
//    pp = p;
//    p = 1;
//    st.push(1);
//    s == true;
//  goto bloop;
//}
//  else if ((MidVoltage1 < 1)||(MidVoltage2<1))
//  {
//    while(LeftVoltage >1)
//    {
//    LeftSensor = analogRead(A0);
//     LeftVoltage = LeftSensor * (5.0 / 1023.0);
//    myservo1.attach(11);
//    myservo2.attach(12);
//    straight();
//    st.push(2);
//    delay(400);
//    }
//}
```

```
goto bloop;
}
hoop:
//Serial.println(sensor);
if (sensor ==1)
{

 myservo1.attach(11);
 myservo2.attach(12);
// reverse();
 //delay(1000);
 updateSensors();
 if (( r == true)&&(RightVoltage<1))
 {
 myservo1.attach(11);
 myservo2.attach(12);
 dot();
 }
 updateSensors();

 if (((p ==1)&&(pp==1)&&(ppp==1))&& (((MidVoltage1 < 1)||(MidVoltage2<1))||(LF1V<1)||(LF2V<1)))
 {
   myservo2.attach(12);
   myservo1.attach(11);
   straight();
   ppp = pp;
   pp = p;
   p =2;
   sensor =2;
   l = false;
   r = false;
   delay(50);
   updateSensors();

   goto bloop;
 }
 else if (((p ==1)&&(pp==1)&&(ppp==1))&&(RightVoltage<1))
 {
   right();
  // r = true;
   delay(400);
   ppp = pp;
   pp = p;
   p =3;
   st.push(3);
   //Serial.println(sensor);
```

```
    myservo2.attach(12);
    myservo1.attach(11);
    straight();
    r = false;
    l = false;
    delay(50);
    //delay(10000);
    updateSensors();

    goto bloop;
  }
  else
  {
    left();
    delay(400);
    l = false;
    ppp = pp;
    pp = p;
    p =1;
    st.push(1);
    goto bloop;

  }

  goto bloop;
}

if (sensor ==3)
{

    if (((p ==3)&&(pp==3)&&(ppp==3))&& (((MidVoltage1 < 1)||(MidVoltage2<1))||(LF1V<1)||(LF2V<1)))
  {
    myservo2.attach(12);
    myservo1.attach(11);
    straight();
    ppp = pp;
    pp = p;
    p =2;
    sensor = 2;
    r = false;
    delay(50);
    updateSensors();

    goto bloop;
  }
  if (((p ==3)&&(pp==3)&&(ppp==3))&&(LeftVoltage<1))
```

```
  {
    left();
    l = false;
    delay(400);
    ppp = pp;
    pp = p;
    p =1;
    st.push(1);
  }
  else
  {
    right();
  // Serial.println(sensor);
    r = true;
    delay(400);
    ppp = pp;
    pp = p;
    p = 3;
    st.push(3);
    // sensor =2;
  }
  myservo1.attach(11);
  myservo2.attach(12);

  goto bloop;
}
if (sensor ==4)
{
  rev = true;
  myservo1.attach(11);
  myservo2.attach(12);
  reverse();
  st.pop();
  delay(400);
  RightSensor = analogRead(A0);
  RightVoltage = RightSensor * (5.0 / 1023.0);




  if ((RightVoltage <1)&&(s==false))
  {
    myservo2.attach(12);
    myservo1.attach(11);
    rightadjust();
```

```
  delay(400);


  myservo2.attach(12);
  myservo1.attach(11);
  reverse();
  delay(400);


 // Serial.println(RightVoltage);
  myservo1.attach(11);
  myservo2.attach(12);
  revright();
  r = true;
  delay(400);
  ppp = pp;
  pp = p;
  p =3;
  st.push(3);
  s == true;
  rev = false;
  goto bloop;
}
else if (rev == true)
{
back:
while ((RightVoltage>1) && (LeftVoltage>1))
{
updateSensors();
myservo1.attach(11);
myservo2.attach(12);
reverse();
delay(400);
// st.pop();


  if (((LF1V>1) &&(LF2V<1))&&MidVoltage2>1)
{
  myservo2.attach(12);
  myservo1.attach(11);
  revrightadjust();
  delay(100);


}


  if (((LF1V<1) &&(LF2V>1))&&MidVoltage1>1)
{
myservo2.attach(12);
myservo1.attach(11);
```

```
  revleftadjust();
  delay(100);
  }
  r  = false;
  }
   if ((RightVoltage <1)&&(s==false))
  {

   myservo1.attach(11);
//   myservo2.attach(12);
//   straight
//   ();
//   delay(400);



   // Serial.println(RightVoltage);
   // Serial.println(s);
    myservo1.attach(11);
    myservo2.attach(12);
    revright();
    r =true;
    delay(400);
    ppp = pp;
    pp = p;
    p =3;
    st.push(3);
    s == true;
    goto bloop;
  }
     else if ((LeftVoltage <1)&&(s==false))
  {
    //Serial.println(RightVoltage);

//    myservo1.attach(11);
//    myservo2.attach(12);
//    straight();
//    delay(400);
    myservo1.attach(11);
    myservo2.attach(12);
    revleft();
    delay(400);
    ppp = pp;
    pp = p;
    p =1;
    st.push(1);
    l = false;
```

```
   s == true;
   r = false;
   goto bloop;
  }
  else
  {
  goto back;
  }
  rev =  false;
  }
  goto bloop;
}

if (sensor ==5)
{
  myservo2.attach(12);
  myservo1.attach(11);
  rightadjust();
  delay(400);
  sensor =2;
}

if (sensor ==6)
{
  myservo2.attach(12);
  myservo1.attach(11);
  leftadjust();
  delay(400);
  sensor =2;
}

if (sensor == 7)
{
  myservo2.attach(12);
  myservo1.attach(11);
  revrightadjust();
  delay(400);
 // sensor = 4;
}

if (sensor == 8)
{
  myservo2.attach(12);
  myservo1.attach(11);
  revleftadjust();
  delay(400);
```

```
  //sensor = 4;
}

s = false;
goto bloop;
}




void left()
{
for (float pos = 0; pos<90; pos = pos+1)
{
myservo2.write(180 - pos);// Rotate to 90 degrees
myservo1.write(pos);
delay(2);

}
myservo1.write(90);
delay(600);
myservo1.detach();
//myservo2.attach(12);
myservo2.write(0);
delay(600);
myservo2.detach();
//myservo2.write(90);   // Rotate to 90 degrees
//delay(1000);



myservo2.detach();



}

void revleft()
{
for (float pos = 0; pos<90; pos = pos+0.9)
{
myservo2.write(180 - pos);// Rotate to 90 degrees
myservo1.write(pos);
delay(2);

}
myservo1.write(90);
delay(600);
```

```
myservo1.detach();
//myservo2.attach(12);
myservo2.write(0);
delay(600);
myservo2.detach();
//myservo2.write(90);   // Rotate to 90 degrees
//delay(1000);



myservo2.detach();



}

void straight()
{
updateSensors();
//myservo1.attach(11);
//myservo2.attach(12);
//if(LeftVoltage>1)
//{
for (float pos = 0; pos<90; pos = pos+3.5)
{

int LeftSensor = analogRead(A5);
float LeftVoltage = LeftSensor * (5.0 / 1023.0);
 if (((LF1V>1)&&(LF2V<1))&&(MidVoltage2>1))
 {
   sensor = 5;
 }

  if (((LF1V<1) &&(LF2V>1)) &&(MidVoltage1>1))
 {
   sensor = 6;
 }

myservo2.write(pos);// Rotate to 90 degrees
myservo1.write(175 - pos);
delay(2);

}

myservo2.detach();
myservo1.detach();
delay(400);
}
```

```
//}


void reverse()
{
updateSensors();
for (float pos = 0; pos<90; pos=pos+6)
{

myservo2.write(180 - pos);// Rotate to 90 degrees
myservo1.write(pos);
delay(2);

}
//delay(500);
myservo2.detach();
myservo1.detach();



}



void right()
{
for (float pos = 0; pos<90; pos=pos+1)
{
myservo2.write(180 - pos);// Rotate to 90 degrees
myservo1.write(pos);
delay(2);

}

myservo2.write(90);
delay(550);
myservo2.detach();
myservo1.write(270);// Rotate to 90 degree
delay(550);
myservo1.detach();
}

void revright()
{
for (float pos = 0; pos<90; pos=pos+0.9)
{
myservo2.write(180 - pos);// Rotate to 90 degrees
myservo1.write(pos);
delay(2);
```

```
}

myservo2.write(90);
delay(650);
myservo2.detach();
myservo1.write(270);// Rotate to 90 degree
delay(650);
myservo1.detach();
}
void rightadjust()
{
myservo2.write(0);
delay(100);
//myservo2.detach();
myservo2.write(90);// Rotate to 90 degree
delay(100);


myservo1.write(180);// Rotate to 90 degree
delay(10);
myservo2.write(90);
delay(10);
//myservo2.detach();

myservo1.detach();
myservo2.detach();

}

void revrightadjust()
{
myservo2.write(0);
delay(150);
//myservo2.detach();
myservo2.write(90);// Rotate to 90 degree
delay(150);


myservo1.write(180);
delay(85);
//myservo2.detach();
myservo2.write(90);// Rotate to 90 degree
delay(85);
myservo1.detach();
myservo2.detach();
```

```
}

void leftadjust()
{
myservo1.write(180);
delay(100);
//myservo2.detach();
myservo1.write(90);// Rotate to 90 degree
delay(100);

myservo2.write(180);// Rotate to 90 degree
delay(2);
myservo1.write(90);
delay(2);
//myservo2.detach();

myservo1.detach();
myservo2.detach();

}

void revleftadjust()
{
myservo1.write(180);
delay(175);
//myservo2.detach();
myservo1.write(90);// Rotate to 90 degree
delay(175);



myservo2.write(0);
delay(100);
//myservo2.detach();
myservo1.write(90);// Rotate to 90 degree
delay(100);
myservo1.detach();
myservo2.detach();

}

void dot()
{
int popped;
myservo1.write(90);
delay(5);
```

```
myservo1.detach();
myservo2.detach();
myservo2.write(90);
delay(50000);
myservo2.detach();
while (!st.isEmpty())
{
popped = st.pop();
moves.push(popped);
//Serial.print(popped);
}
shortest();
}

void shortest()
{
int popped;
while (!moves.isEmpty())
{
popped = moves.pop();
if (popped ==1)
{
if (sensor ==1)
{

  myservo1.attach(11);
  myservo2.attach(12);
  left();
  delay(400);
}
else if (sensor ==2)
{

  myservo1.attach(11);
  myservo2.attach(12);
  straight();
  delay(400);
}

else if (sensor ==3)
{

  myservo1.attach(11);
  myservo2.attach(12);
  right();
  delay(400);
```

```
}


}
}
//Serial.print(popped);
}
```