EXP No. 1: maze solving robot.

Prelab work:

- Study the Working of HC- SR 04 Ultrasonic sensor along with its timing Diagram and distance calculation.
- Study the working motor driver L298N and its connection to the Arduino and Motor
-

OBJECTIVE:

1. Design and build the robot using ultrasonic sensors.
2. Understand and implement wall-follower algorithm.
3. Construct Arduino based program to solve the maze.

COMPONENTS REQUIRED

Sl.No	Name of components	Specification	Quantity
1	Arduino board	UNO	1 NO
2	Motor driver	L298N	2 No's
3	DC motors	12 volt	2 No's
4	Ultrasonic sensors	HC- SR 04	3 No's
5	Battery	9 volt	2 No's
6	Tires and ball caster	--	2 No's
7	Chassis	--	1 No
8	Bread board	--	1 No

INTRODUCTION

A maze is a complicated system of paths from entrance to exit. Maze solving problem involves determining the path of a mobile robot from its initial position to its destination while travelling through environment consisting of obstacles. In addition, the robot must follow the best possible path among various possible paths present in the maze.

Maze solving - a seemingly minor challenge for the analytical minds of humans – has generated enough curiosity and challenges for A.I. experts to make their machines (robots) solve any given maze.

Applications of such autonomous vehicles range from simple tasks like robots employed in industries to carry goods through factories, offices, buildings and other workplaces to dangerous or difficult to reach areas like bomb sniffing, finding humans in wreckages etc.

Some maze solving methods are designed to be used inside the maze with no prior knowledge of the maze whereas others are designed to be used by a computer program that can see whole maze at once. We used the former one. Our mazes are simply – connected i.e., without any closed loops.

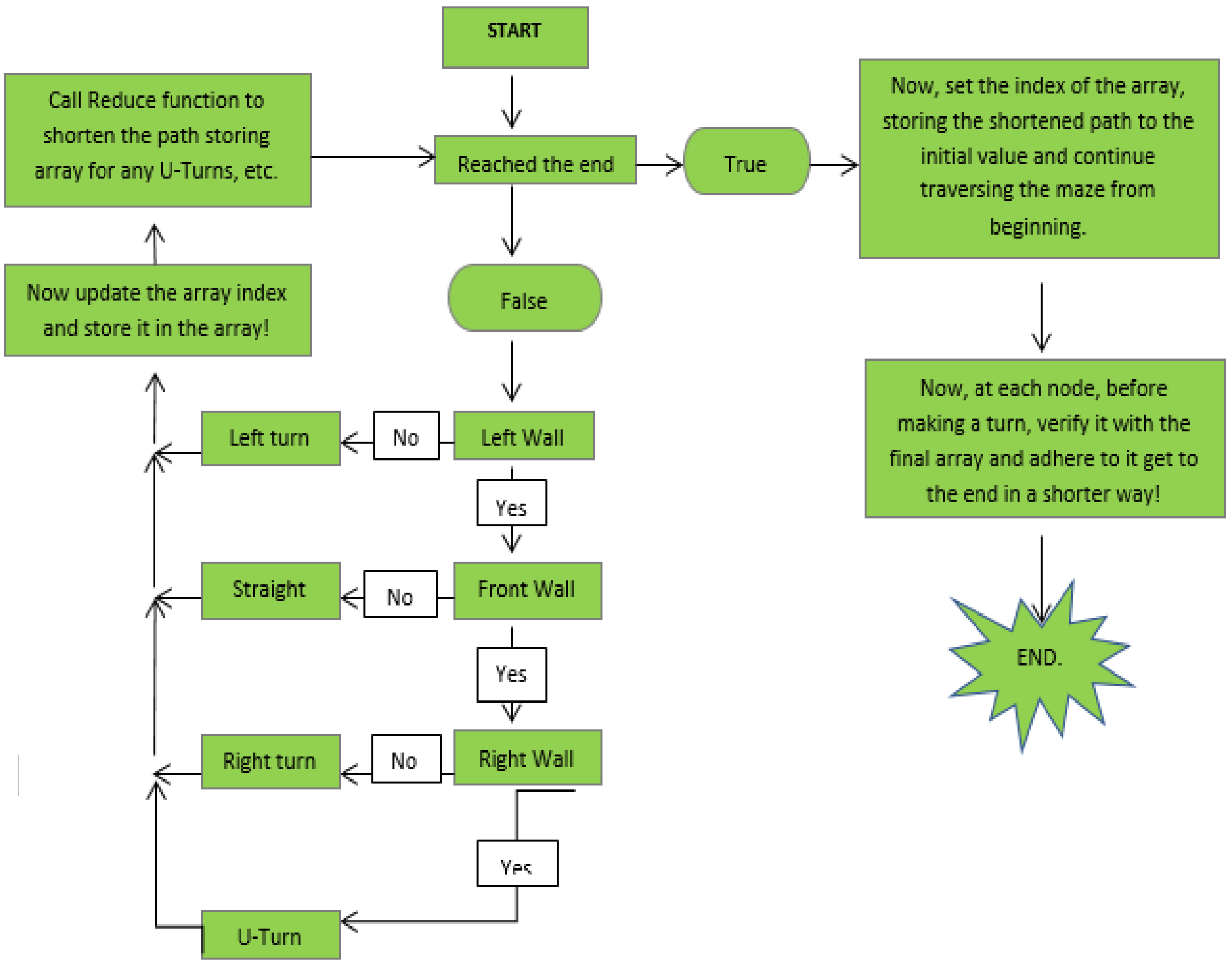
Autonomous maze solving robot that we have built first goes from start to end searching in the maze using wall follower algorithm (left hand rule) and then processes out the shortest possible path by eliminating dead ends and takes that path next time . It does all of that without any human assistance.

WALL FOLLOWING ALGORITHM

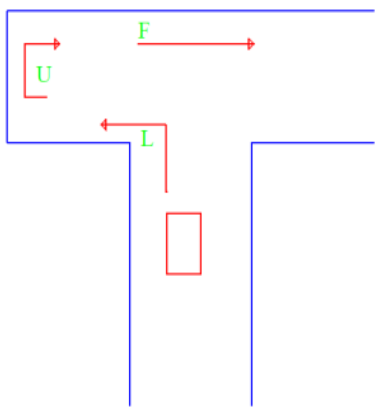
The wall follower, one of the best known rules for traversing mazes, is also known as either left-hand rule or the right-hand rule. Whenever the robot reaches a junction, it will sense for the opening walls and select its direction giving the priority to the selected wall. Here, the selected wall is left! The priority in left hand wall is in the following order: left, straight and right and if all the sides are closed, then U-turn.

The robot uses three ultrasonic sensors attached to it, to determine the distance of the walls in three directions: Front, Left and Right. If the distance of the wall from the bot is less than the set threshold value in a particular direction, then the robot assigns it as closed and similarly if it is greater than the set threshold value, then it assigns it as open. It also uses the left and right wall distances to align itself parallel to the wall, while moving forward.

Flow chart


START
Reached the end
True
Now, set the index of the array, storing the shortened path to the initial value and continue traversing the maze from beginning.
Now, at each node, before making a turn, verify it with the final array and adhere to it get to the end in a shorter way!
END.
Call Reduce function to shorten the path storing array for any U-Turns, etc.
Now update the array index and store it in the array!
False
Left turn
No
Left Wall
Yes
Straight
No
Front Wall
Yes
Right turn
No
Right Wall
Yes
U-Turn

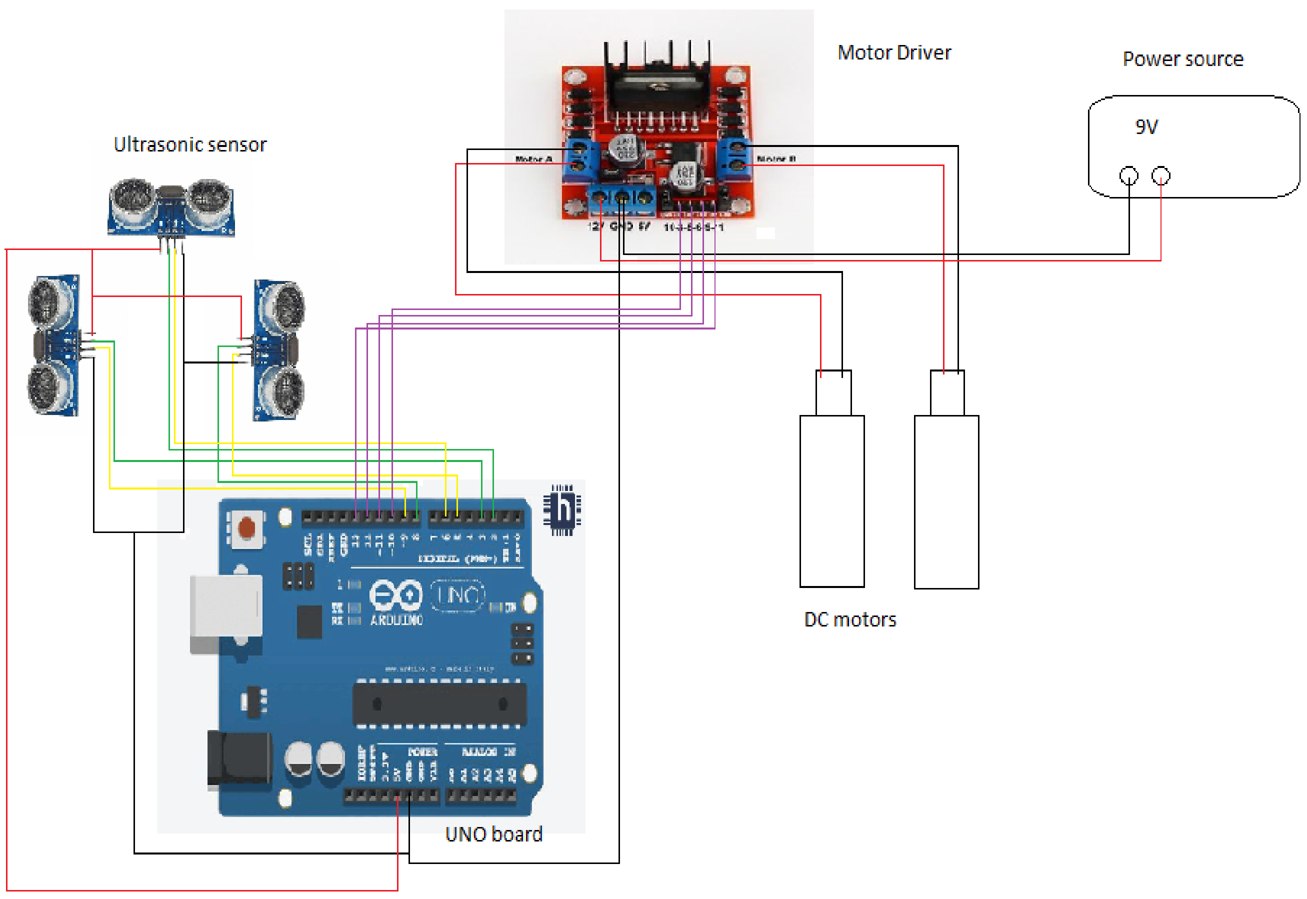
As the bot traverses the maze, its path is stored in an array and is simultaneously reduced by calling the function 'reduce', if it can be shortened. For example, if we consider the following part of the maze: at the node where it moved forward over right, this turn along with its two previous turns (U-turn and a left turn respectively) can be shortened to 'R'! That is, 'LUF = R' and now the array is dynamically modified and the modified path is updated into it. After reaching the end, the bot again starts from the beginning and this time, at each node it checks with the shortened array and follows it to go to the end in a much shorter path!



In this way, while traversing the maze the bot remembers as well as shortens and adheres it to reach the end in a much shorter path!

As it is said, the wall follower algorithm is amateur programmers' favorite! This method works perfectly well for those mazes whose start and end is wall connected or in other words for those mazes which are simply connected, but gets stuck in loops. There are many algorithms which have successfully overcome this problem of getting stuck in loops, but most of them have one condition, that the maze be known priorly. Examples of some of the algorithms are: Flood Fill Algorithm which is extensively used in 'Micromouse Competitions', the Pledge algorithm, the Tremaux's algorithm, etc.

Circuit Diagram:



CODE

The below code contains code both for Maze Solving and Shortest path Simplification.

```
#define fthold 12     // Threshold value in front direction
#define rthold 20     // Threshold value in right direction
#define lthold 20     // Threshold vlaue in left direction

const int t = 1050;  // Time alotted for taking 90 degrees for 9V!
int tfr =2750;       // Initial Time for which it moves forward when it chooses
forward over right.
int timefr;          // Dynamically set time..for which it moves forward,when it
chooses forward over right.
int tlbef=440;       // Time for which it moves forward before taking left turn.
int tlaf = 1150;     // Time for which it moves forward after taking left turn.
int nf =0;           // Number of times it chooses straight over right.
int nlr=0;           // Number of times it takes left turn.
bool found=false;    // If its true, it indicates that the bot has reached the
end!
int dir[100];        // Array for storing the path of the bot.
int i=-1;            // For the indices of the dir array.
int j=0;             // Implies the number of intersections bot passed through.

// Front US sensor.
const int trigPinf = 2;
const int echoPinf = 6;

// Right US sensor.
const int trigPinr = 8;
const int echoPinr = 5;

 // Left US sensor.
const int trigPinl =  3;
const int echoPinl =  9;

//Booleans for recognising the walls. True if resp sensor distance is less than
the resp threshold vlaue.
bool fsensor;       // For the front US sensor.
bool rsensor;       // For the right US sensor.
bool lsensor;       // For the left US sensor.

// Sorts and returns the median value of a five element array.
float middleval(float arr[])
{
   for(int p=0;p<4;p++)
    {
        for(int q=0;q<4;q++)
        {
            if(arr[q]>arr[q+1])
            {
                int temp = arr[q];
                arr[q] = arr[q+1];
                arr[q+1] = temp;
            }
        }
    }
  return arr[2]; // Median value.
}

// Moves the bot in the forward direction
void gofront()
{
    // Moves forward adjusting its path

    float  ldist1 = leftdist();
    float lconst = ldist1;

     while(ldist1<=5)     // Should turn a little to its right
     {
       digitalWrite(10,HIGH);
       digitalWrite(11,LOW);
       digitalWrite(12,LOW);
       digitalWrite(13,LOW);

       delay(t/65);

       ldist1 = leftdist();
       if(abs(lconst - ldist1)>=0.8||(ldist1>=3.6)){break;}
     }

   float rdist1 = rightdist();
   float  rconst = rdist1;

  while(rdist1<=5.4)         // Should turn a little to its left
    {
     digitalWrite(10,LOW);
     digitalWrite(11,LOW);
     digitalWrite(12,HIGH);
     digitalWrite(13,LOW);

     delay(t/65);

     rdist1 = rightdist();
     if(abs(rconst - rdist1)>=0.9){break;}
    }

   if(leftdist()>=7.2)       // Will move little to its left if its too far from
the left wall
    {
     digitalWrite(10,LOW);
     digitalWrite(11,LOW);
     digitalWrite(12,HIGH);
     digitalWrite(13,LOW);
     delay(t/30);
    }

    digitalWrite(10,HIGH);
    digitalWrite(11,LOW);
    digitalWrite(12,HIGH);
    digitalWrite(13,LOW);
  }

// Returns the dist of wall in front of it
float frontdist()
{
 float gapf;float ticktockf;

 digitalWrite(trigPinf,LOW);
 delayMicroseconds(2);
 digitalWrite(trigPinf,HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPinf,LOW);

ticktockf = pulseIn(echoPinf,HIGH);    // in one cm there are 29 microseconds.
 gapf = ticktockf*0.0344/2;

 return gapf;
}

// Returns the distance the wall to its right side
 float rightdist()
 {
   float gapr;float ticktockr;
   digitalWrite(trigPinr,LOW);
   delayMicroseconds(2);
   digitalWrite(trigPinr,HIGH);
   delayMicroseconds(10);
   digitalWrite(trigPinr,LOW);

   ticktockr = pulseIn(echoPinr,HIGH);
   gapr = ticktockr*0.0344/2;

   return gapr;
 }

// Returns the distance of the wall to its left side
float leftdist()
 {
   float gapl;float ticktockl;

   digitalWrite(trigPinl,LOW);
   delayMicroseconds(2);
   digitalWrite(trigPinl,HIGH);
   delayMicroseconds(10);
   digitalWrite(trigPinl,LOW);

   ticktockl = pulseIn(echoPinl,HIGH);
   gapl = ticktockl*0.0334/2;

   return gapl;
 }

// Reduces the path if it can be shortened and dynamically modifies the path
storing array.
void reduce(int dir[], int &pt)
{
 int i=pt;
 if(i>=2)
         {
  //RUL = U..
  if((dir[i-1]==3)&&(dir[i-2]==2)&&(dir[i]==1))
                                  {
                                    dir[i-2]=3;
                                    pt = pt -2;
                                  }
  //LUL = F..
  else if((dir[i-1]==3)&&(dir[i-2]==1)&&(dir[i]==1))
                                  {
                                    dir[i-2]=0;
                                    pt = pt -2;
                                  }
  //LUR = U..
  else if((dir[i-1]==3)&&(dir[i-2]==1)&&(dir[i]==2))
                                  {
                                    dir[i-2]=3;
                                    pt = pt -2;
                                  }
 //FUF = U..
 else if((dir[i-1]==3)&&(dir[i-2]==0)&&(dir[i]==0))
                                  {
                                    dir[i-2]=3;
                                    pt = pt -2;
                                  }
 //FUL = R..
 else if((dir[i-1]==3)&&(dir[i-2]==0)&&(dir[i]==1))
                                  {
                                    dir[i-2]=2;
                                    pt = pt -2;
                                  }
 //LUF = R..
 else if((dir[i-1]==3)&&(dir[i-2]==1)&&(dir[i]==0))
                                  {
                                    dir[i-2]=2;
                                    pt = pt -2;
                                  }
return;
     }
}

// Stops the bot
void stopit()
{
   digitalWrite(10,LOW);
   digitalWrite(11,LOW);
   digitalWrite(12,LOW);
   digitalWrite(13,LOW);
}

// When it has to move forward according to the shortest path.(At some
intersection)
void frontturn()
{

     for(int n=1;n<=8;n++)
     {gofront();delay((timefr)/8);}


    digitalWrite(10,LOW);
    digitalWrite(11,LOW);
    digitalWrite(12,LOW);
    digitalWrite(13,LOW);
    delay(1000);
}

// When it has to take a right turn according to the shortest path.
void rightturn()
{  stopit();delay(1000);
   float prevfdist = frontdist();

 //while( abs(frontdist()-prevfdist)<=(prevfdist/2)-1)
  for(int n=1;n<=5;n++)
  {gofront();delay(260);}

    digitalWrite(10,HIGH);
    digitalWrite(11,LOW);
    digitalWrite(12,LOW);
    digitalWrite(13,HIGH);

    delay(t);

   // gofront();delay(2400);
   float prevfrdist = frontdist();

   while( abs(frontdist()-prevfrdist)<=18)
 /* for(int n=1;n<=10;n++)*/
   {gofront();delay(300);}

    digitalWrite(10,LOW);
    digitalWrite(11,LOW);
    digitalWrite(12,LOW);
    digitalWrite(13,LOW);
    delay(1000);
}


void setup()  // put your setup code here, to run once:
{
  // US pins setup..
  pinMode (trigPinf,OUTPUT);
  pinMode (echoPinf,INPUT);
  pinMode (trigPinr,OUTPUT);
  pinMode (echoPinr,INPUT);
  pinMode (trigPinl,OUTPUT);
  pinMode (echoPinl,INPUT);
  pinMode( 4,INPUT);        // FOR THE IR SENSOR...

  // Motor pins.
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);

  Serial.begin(9600); //staartingg serial communication...9600 bits per second.

// dir[0] = 0; // initial direction..?
}

void loop()    // put your main code here, to run repeatedly
 {
         if(nlr==7)
                      {
                        found=true;   // Reached the end.

                        for(int i=0;i<=2;i++){Serial.print(dir[i]);}
                        i=-1;j=0; nlr=0;      // Back to start again..


                        // Stops the bot for 30 seconds after reaching the end.
                          digitalWrite(10,LOW);
                          digitalWrite(11,LOW);
                          digitalWrite(12,LOW);
                          digitalWrite(13,LOW);
                          delay(30000);
                      }

  float fdist;    float rdist;    float ldist;      // front, right, and left
distances.
  float fduration;float rduration;float lduration;  // front, right, and left
travel time in echoPin.
  float fdur[5];  float rdur[5];  float ldur[5];    // Arrays which store the
values of five durations... We will take only the median value(afer sorting)
with error bearing capacity of 40%.
  float ldista[5];


// For the front US sensor..
  for(int i=0;i<=4;i++)
  {
   digitalWrite(trigPinf,LOW);         // Clearing the trigPin.
   delayMicroseconds(5);
   digitalWrite(trigPinf,HIGH);        // Setting the trigPin HIGH for 10
microseconds..sends some 8cycle sonics.
```

```
   delayMicroseconds(10);
   digitalWrite(trigPinf,LOW);

  fdur[i] = pulseIn(echoPinf,HIGH);    // Returns the time for which the wave
travelled.
  }

  fduration = middleval(fdur);
  fdist = fduration*0.0344/2;          // Distance of the wall in the forward
direction

  /*Serial.print("frontdistance: ");
  Serial.println(fdist);*/

 // for the right US sensor...
   for(int i=0;i<=4;i++)
   {
   digitalWrite(trigPinr,LOW);
   delayMicroseconds(5);
   digitalWrite(trigPinr,HIGH);
   delayMicroseconds(10);
   digitalWrite(trigPinr,LOW);

   rdur[i] = pulseIn(echoPinr,HIGH);
   }

   rduration = middleval(rdur);
   rdist = rduration*0.0344/2;         // Distance of the wall to its right.

 /* Serial.print("rightdistance: ");
 Serial.println(rdist);*/

 // for the left US sensor....
   for(int i=0;i<=4;i++)
   {
   digitalWrite(trigPinl,LOW);
   delayMicroseconds(5);
   digitalWrite(trigPinl,HIGH);
   delayMicroseconds(10);
   digitalWrite(trigPinl,LOW);

   ldur[i] = pulseIn(echoPinl,HIGH);

   }

   lduration = middleval(ldur);
   ldist = lduration*0.0344/2;        // Distance of the wall to its left side

 /* Serial.print("leftdistance: ");
   Serial.println(ldist);*/

 if((fdist>=125)||(rdist>=150)||(ldist>=400)) {return;}    // Cancelling out any
error values...goes back to void loop().
```

```
  fsensor = false;rsensor = false;lsensor = false;          // Setting up the
booleans.

  if(rdist<=rthold) rsensor = true;
  if(ldist<=lthold) lsensor = true;
  if(fdist<=fthold) fsensor = true;



                                                                       // Left
Wall Following Algorithm!

 // If left is closed-
if((lsensor==true))
  {                 // for a U-turn..

   if((rsensor==true)&&(fsensor==true))
       { j=j+1;
         i=i+1;
         dir[i] = 3;
         reduce(dir,i);

         digitalWrite(10,HIGH);
         digitalWrite(11,LOW);
         digitalWrite(12,LOW);
         digitalWrite(13,HIGH);
         delay(2*t);
       }

// If Front is open..
 else if(fsensor==false)
  {
   if((rsensor==false)&&(frontdist()>=40))    // If both front and right are
open..
    {
      i = i+1;
      j=j+1;

     if((found==true)&(dir[i]!=0))   // After reaching the end ... checks the s
                                 {
                                   rightturn();return;
                                 }
   else
     {
      if(found==false){
                       dir[i] = 0; // moving forward over right...
                       reduce(dir,i);
                      }
      /*Serial.print("for the jth turn ..");Serial.print(" =");Serial.print(j);
      Serial.print(" the i value is ");Serial.print(i);Serial.print("and the dir
is ..");Serial.println(dir[i]);*/


      timefr = tfr + 65*nf;
      nf=nf+1;
      stopit();delay(1000);

      for(int g=1;g<=10;g++){gofront();delay(timefr/10);}

      stopit();delay(1000);
     }
    }

else {gofront();delay(300);}   // Else moving forward .. only front is open.
 }

//for a right turn..
 else
   {
     i = i+1;
     j=j+1;
     dir[i] = 2;
     reduce(dir,i);

     float prevfdist = frontdist();

     while( abs(frontdist()-prevfdist)<=(prevfdist/2)-2)
{gofront();delay(300);if(frontdist()<=4.5){break;}}

     digitalWrite(10,HIGH);
     digitalWrite(11,LOW);
     digitalWrite(12,LOW);
     digitalWrite(13,HIGH);
     delay(t);

     float prevfrdist = frontdist();

   while( abs(frontdist()-
prevfrdist)<=15.2){gofront();delay(300);if(frontdist()<=4.5){break;}}
   }


}


else
 {
   //for a left turn..
   i=i+1;
   j=j+1;

   if((found==true)&&(dir[i]!=1)){

if((dir[i]==2)&&(rightdist>=10)){rightturn();return;}
                                  else if((dir[i]== 0)&&(fsensor==false))
{frontturn();return;}
                                 }

   else{
    dir[i]=1;       // Left turn..
    nlr=nlr+1;
    reduce(dir,i);  //calling reduce function to shorten the path
dynamically..if path is not yet completed

    {gofront(); delay(tlbef);}

    digitalWrite(10,LOW); // takes a left turn..
    digitalWrite(11,LOW);
    digitalWrite(12,HIGH);
    digitalWrite(13,LOW);
    delay(2*t);

    for(int n=1;n<=8;n++) { gofront();delay(tlaf/8);}

    stopit();delay(1000);
       }
  }

delay(320);
}
```

Result