



Egyptian Russian University

Faculty of Engineering

Mechatronics & Robotics Dep.

Group Project

Embedded Systems - EE314

Supervised by:

Dr. Ahmed Sedik

No.	Name	ID
1	Omar Khaled	171033
2	Omar Alaa	171385
3	Omar Adel	171112
4	Ramez Mohamed	171113
5	Youssef Sherif	171181

Table of Contents

Introduction.....	3
Components	4
Steps of work.....	6
Implementation	10
Schematic Diagram	14
Coding part	15
Conclusion	20

Introduction








In today's scenario, dense network of city streets, disaster prone regions and war prone environments have common navigational difficulties. Specific missions, hazardous surroundings, inhabitable condition serve as a reason for shift to autonomous technology and decision-oriented mechanics. It is well known that the idea of decision-making algorithms, for application like maze solving leads an entrance to a goal, regardless of the challenge of the maze solving being three centuries old, it still holds a vital importance in robotics. If a robot is positioned in an unfamiliar environment, it should have a good decision-making algorithm in order to successfully solve a maze, so to design a robot we are using LSRB algorithm.




Modern robotics technologies are focused on developing self-navigating autonomous robots to automate our day-to-day processes. This means that most of the research focuses on improving sensors and algorithms to build flexible and accurate robots.

The maze solving robot — also known as a micro mouse — 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.

In this project, we will build a simple Arduino maze solving robot using three ultrasonic sensors.

Components

No.	Component name	No. of pieces	Image
1	Ultrasonic Sensor	3	
2	Ultrasonic Sensor mounting bracket	3	
3	Arduino Uno	1	
4	DC motor	4	
5	Jumper wires	25	
6	1.5v DC Battery	4	
7	4 Wheeled robotic car (Model ZK -4WD)	1	

8	Bread board	1	
9	L293D (4 channel H-Bridge)	1	
10	Double face tape	1	

All equipment and components were bought from:

(<https://ram-e-shop.com/>)

All Details in the link above

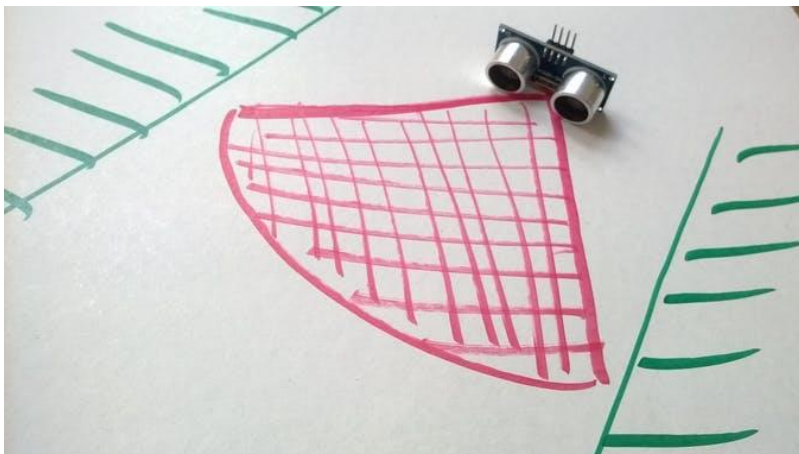
Steps of work

We started to learn to move on the given coordinates and self-supporting wall. Slowly we were preparing for the tasks that await us. Gave way to various problems robot he found a hole in the wall, did not have engines feedback - we manage them on time, the sensors are giving illogical feedback. Like every beginning it was very difficult, but we never gave up. We have found solutions.

Battery and ultrasonic sensors

Our batteries are very quickly consumed, due to the capacity control units were not allowed to put a stronger battery. Most often it happened that amperage quickly falls to zero. We installed a voltage stabilizer 7806 and worked with stronger batteries.

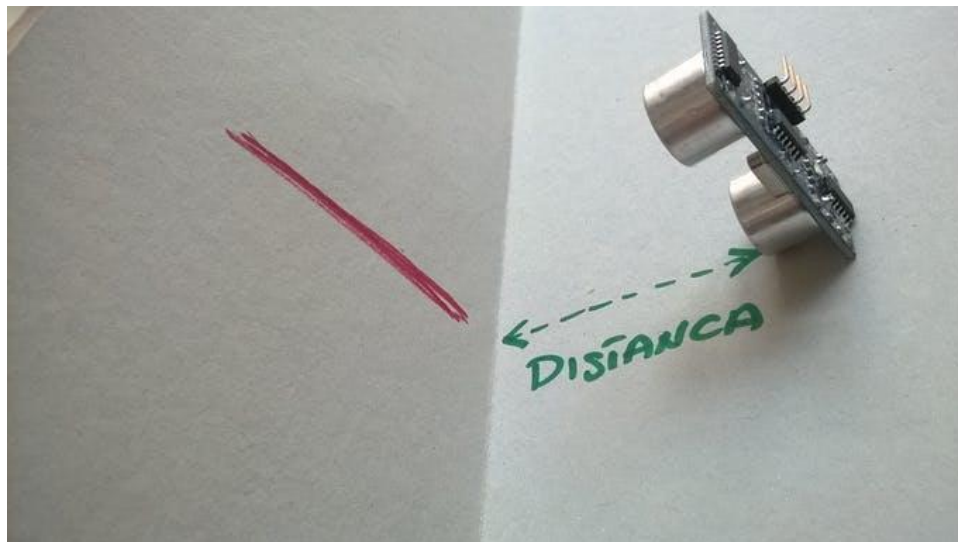
As is already known ultrasonic sensors operate on the principle of radar. Sends audio signals that are reflected from the object and returns back and this time supposed to be a signal to go and return is calculated as the time and distance itself is obtained by the time divided by two. First we set the sensor in a horizontal position because that is the basic position and we thought it would be a good show. We were wrong. So turn the sensors are good for measuring the distance while standing robot because its scope to capture 120°



Therefore, when tracking, the wall was badly registered by the same distance. We tried to angle sensors placed against the wall. We turned it parallel to the wall but at an angle of 45° , 30° and 60° , but nothing significant has changed.

We found the solution!

We turned vertically or sensors so that component that sends a signal down is the one that receives the signal up.



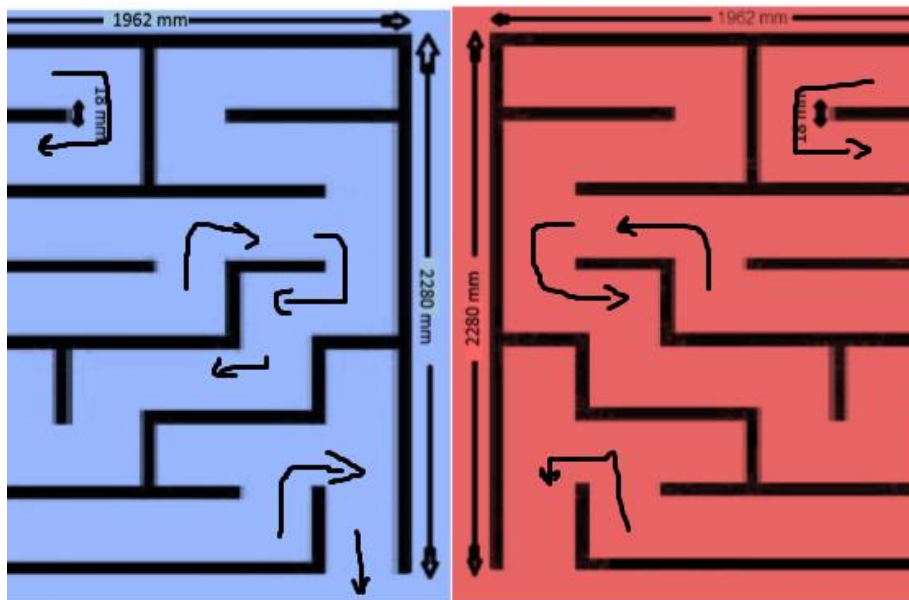
Why Vertical?

Because it is a signal range of lines and measured the exact distance of the robot from the wall. It does not matter whether the robot approaches, moves away or is parallel to the wall. We can always get an accurate signal. Thanks to this we are not running into an invisible hole in the wall and began a nice robot that follows the wall.

Algorithm:

Maze solving robot is one of the most popular autonomous robots. Our maze solving robot make multiple runs in the maze, first it creates a map of maze layout and store it in its memory, then run through a shortest path, this is done through *LSRB* algorithm. The robot will take its direction by following either left or right wall, in our robot we are using *LSRB* algorithm it states that when there is an intersection, turn left if you can, else go straight if you can, else turn right if you can, else turn around because you are at dead end. A program/algorithm must be deployed and able to guide the robot to the end of maze using ANSI C.

In this way this will respond faster and reaches the end in minimum possible time.



Hardware:

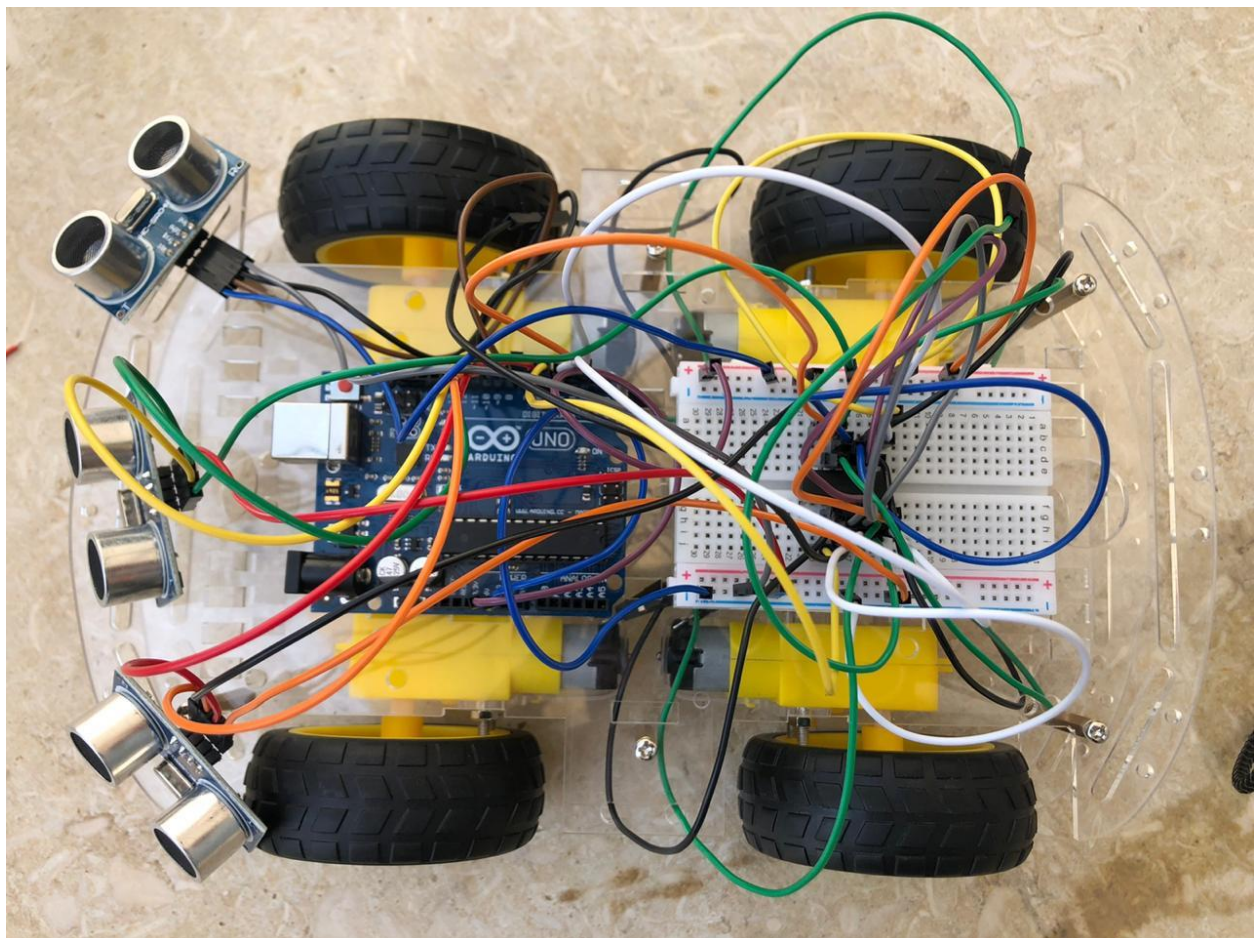
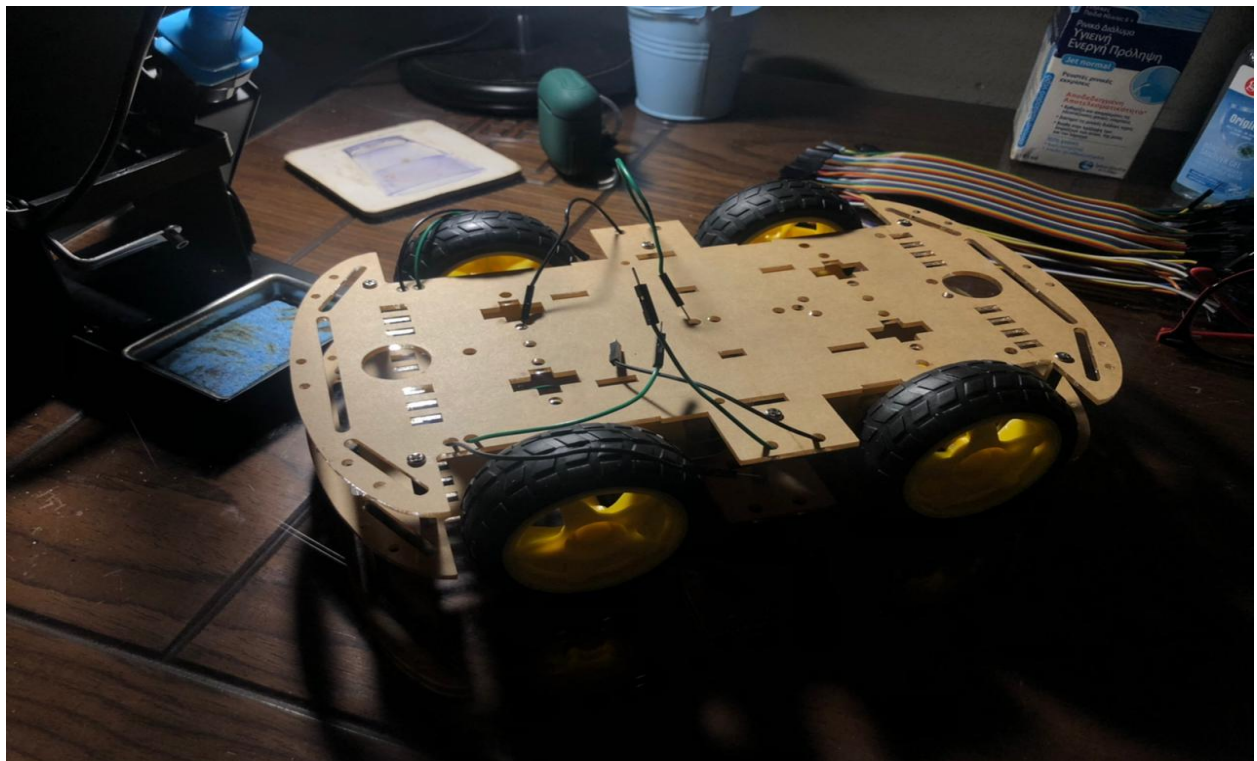
In our robot we are using Arduino Uno board, it acts as the brain of our robot because all the decisions which is taken by the robot to solve the maze is governed by this board and three ultrasonic sensors are used to take the input. The robot is capable of scanning the obstacles with the help of ultrasonic sensors; they transmit ultrasonic waves from its sensor head and again receive the waves reflected back from the object. By measuring the delay of time from the transmission to reception of sonic wave, it detects the position of the object or wall. And then move according to the algorithm avoiding all the obstacles and reach the final destination. Also, this robot has a special feature of navigating through human control for that we are using Bluetooth module, this feature can be used in extreme cases when there is a requirement of human control decision making.

Implementation

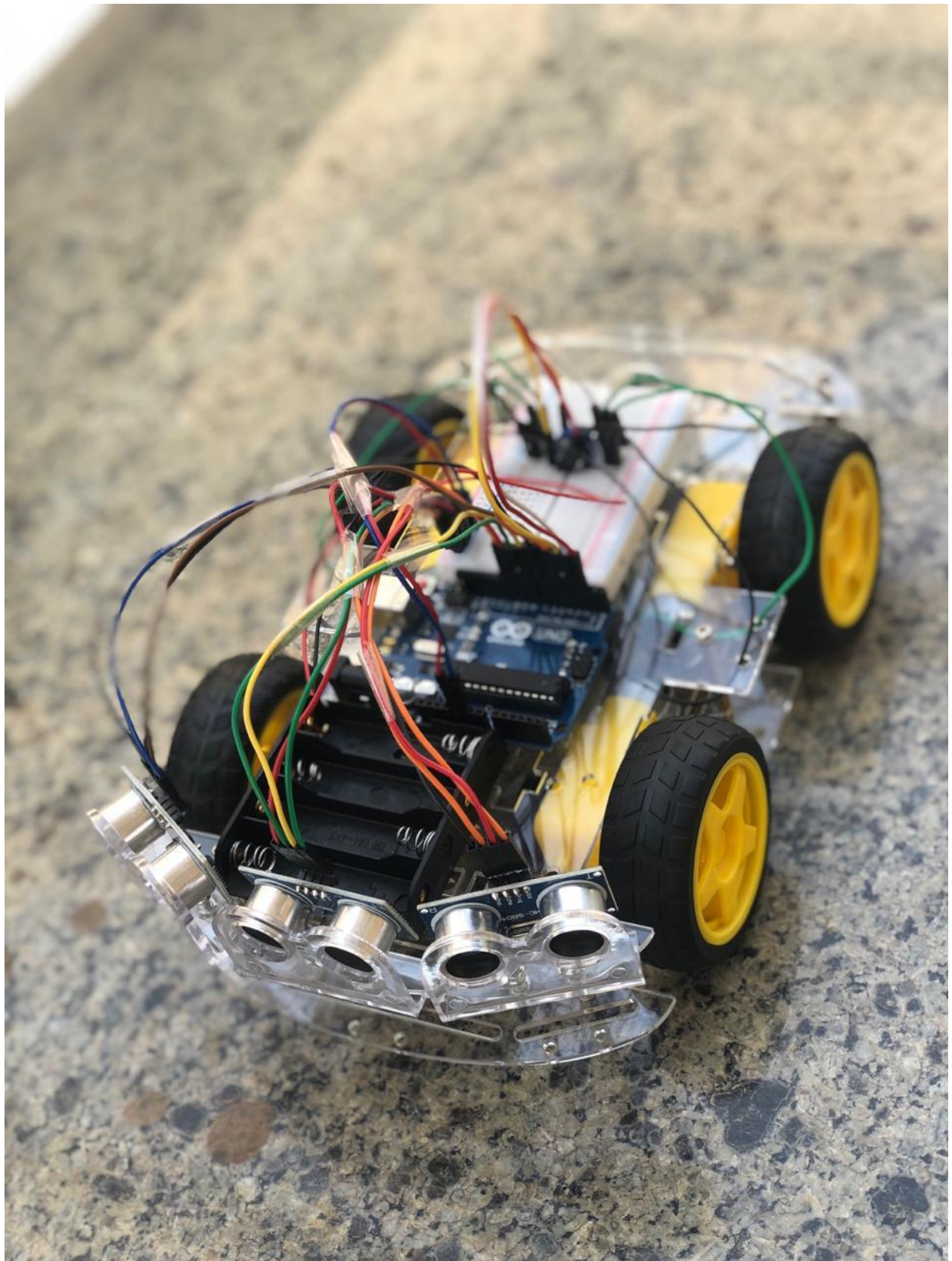
Firstly, we start fixing the motors in the robot body and then attach the wheels into each motor

And then fix the bottom part of the body with upper. After that we start attaching the sensors and the Arduino Uno in the upper part of the body

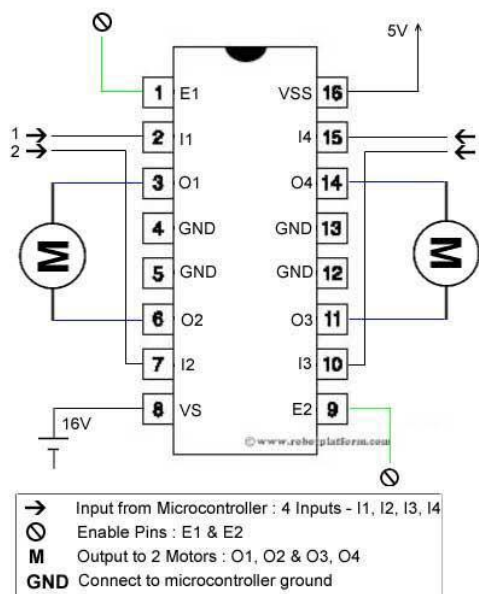
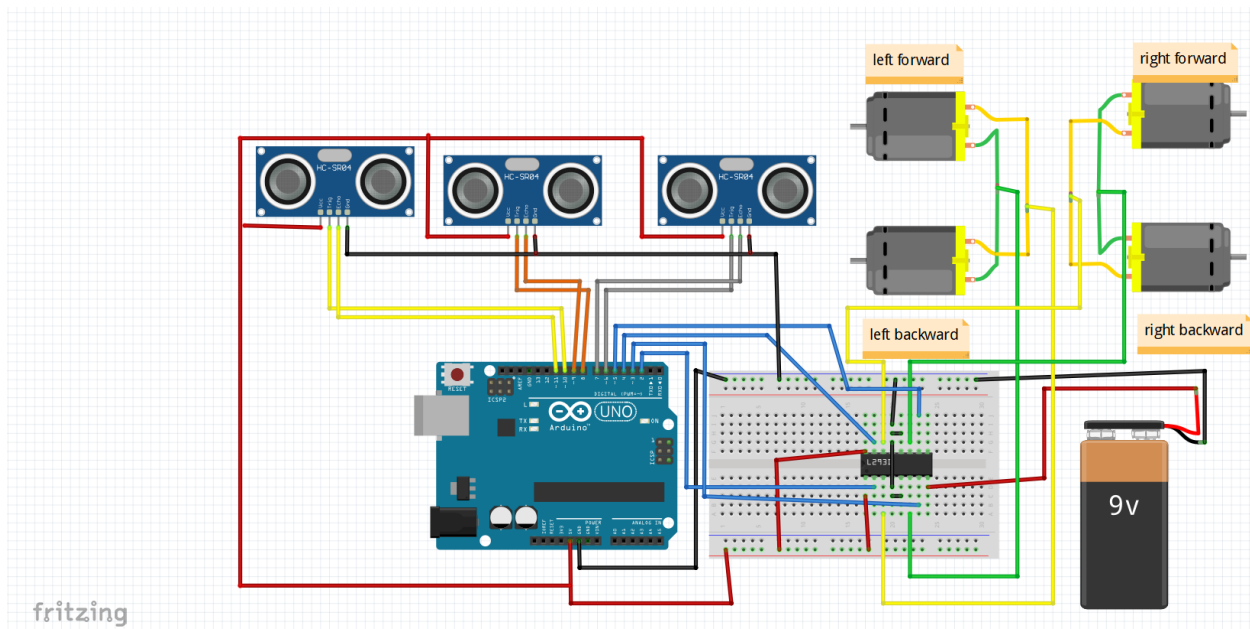




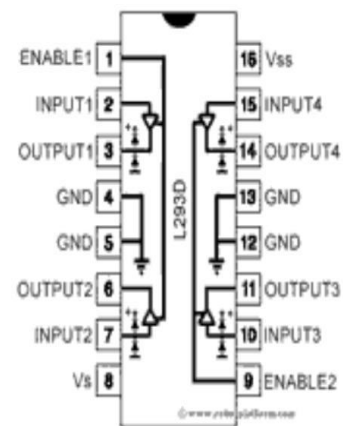




Schematic Diagram



Introduction to L293D IC



Coding part

Project_Code

```
const int trigPin1 = 6; //front
const int echoPin1 = 7;
const int trigPin2 = 10; //left
const int echoPin2 = 11;
const int trigPin3 = 8; //right
const int echoPin3 = 9;
const int in1 = 2;
const int in2 = 3;
const int in3 = 4;
const int in4 = 5;

#define DIS 7

void setup()
{
    pinMode(trigPin1, OUTPUT);
    pinMode(echoPin1, INPUT);
    pinMode(trigPin2, OUTPUT);
    pinMode(echoPin2, INPUT);
    pinMode(trigPin3, OUTPUT);
    pinMode(echoPin3, INPUT);
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(in3, OUTPUT);
    pinMode(in4, OUTPUT);
}
```

```
void loop()
{
  if (FrontSensor () > DIS && RightSensor () > DIS && LeftSensor () > DIS)
  {
    forward();
  }
  //else if ( FrontSensor () > DIS && RightSensor () < DIS && LeftSensor () < DIS)
  //{
  //forward();
  //}
  else if ( FrontSensor() < DIS && RightSensor () < DIS && LeftSensor () < DIS) // obstacle infront of all 3 sides
  {
    reverse ();
    delay(500);
    if ((LeftSensor()) > (RightSensor()))
      turn_left();
    else
      turn_right();
    //then reverse
  }
  else if (FrontSensor() < DIS && RightSensor () < DIS && LeftSensor () > DIS) // obstacle on right and front sides
  {
    turn_left (); // turn left side
  }
  else if (FrontSensor() < DIS && RightSensor () > DIS && LeftSensor () < DIS) // obstacle on left and front sides
  {
    turn_right (); // turn right side
  }
  else if (FrontSensor() < DIS && RightSensor () > DIS && LeftSensor () > DIS) // obstacle on front sides
  {
    turn_left ();
    delay(500);
  }
}
```



```
    forward();// then turn right //*****
}
else if (FrontSensor() > DIS && RightSensor () > DIS && LeftSensor () < DIS) // obstacle on left sides
{
    turn_right(); // then turn right and then forward
    delay(500);
    forward();
}
else if (FrontSensor() > DIS && RightSensor () < DIS && LeftSensor () > DIS) // obstacle on right sides
{
    turn_left (); // then turn left and then right
    delay(500);
    forward();
}
else
{
    forward();
}
}
void forward ()
{
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
}
void turn_left ()
{
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
}
void turn_right ()
```

```
void turn_right ()
{
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);

}
void reverse ()
{
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);

}
void stop()
{
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);

}
long FrontSensor ()
{
    long dur;
    digitalWrite(trigPin1, LOW);
    delayMicroseconds(2); // delays are required for a succesful sensor operation.
    digitalWrite(trigPin1, HIGH);
    delayMicroseconds(10); //this delay is required as well!
    digitalWrite(trigPin1, LOW);
    dur = pulseIn(echoPin1, HIGH);
    return (dur / 58); // convert the distance to centimeters.
}
```

```
}  
  
long FrontSensor ()  
{  
    long dur;  
    digitalWrite(trigPin1, LOW);  
    delayMicroseconds(2); // delays are required for a succesful sensor operation.  
    digitalWrite(trigPin1, HIGH);  
    delayMicroseconds(10); //this delay is required as well!  
    digitalWrite(trigPin1, LOW);  
    dur = pulseIn(echoPin1, HIGH);  
    return (dur / 58); // convert the distance to centimeters.  
}  
  
long RightSensor ()  
{  
    long dur;  
    digitalWrite(trigPin2, LOW);  
    delayMicroseconds(2); // delays are required for a succesful sensor operation.  
    digitalWrite(trigPin2, HIGH);  
    delayMicroseconds(10); //this delay is required as well!  
    digitalWrite(trigPin2, LOW);  
    dur = pulseIn(echoPin2, HIGH);  
    return (dur / 58); // convert the distance to centimeters.  
}  
  
long LeftSensor ()  
{  
    long dur;  
    digitalWrite(trigPin3, LOW);  
    delayMicroseconds(2); // delays are required for a succesful sensor operation.  
    digitalWrite(trigPin3, HIGH);  
    delayMicroseconds(10); //this delay is required as well!  
    digitalWrite(trigPin3, LOW);  
    dur = pulseIn(echoPin3, HIGH);  
    return (dur / 58); // convert the distance to centimeters.  
}
```

Conclusion

We finally ready to run the code and see the robot avoiding obstacles and walls.

