

**Project Title: Maze Solver**

**Group Name: The Exception Handlers**

**Group Members:**

|  |  |
| --- | --- |
| Name | Student ID |
| Azwad Fawad Hasan | **2020222** |
| Tasnia Tabassum Oishee | **1910040** |
| Tanima Ahamed Tulon | **2030314** |
| Abdun Noor Shihab | **2021871** |

**Course ID:** CSE310L

**Section:** 01

**Submitted By:** Azwad Fawad Hasan

**ID:** 2020222

**Submitted To:** Azfar Hossain

# Abstract:

Autonomous navigation is a crucial feature that enables a mobile robot to autonomously go from one point to another without the aid of a human operator. The robot must find its way through a maze of obstacles by researching, identifying, and charting the environment it is in.

It is possible to research and improve the relevant robot behavior and algorithms. This paper describes a robot maze-solver implementation for a flood-fill algorithm-based maze. The maze's entrances were located using the ultrasonic sensors. The robot was able to explore the maze and select the fastest path among all feasible routes to finish it thanks to PI and the Arduino code, the straight-line correction controller method.

# Introduction:

A key element of mobile robotics is autonomous navigation, which enables the robot to travel freely from one site to another without the assistance of a a servo motor and an ultrasonic sensor. Numerous techniques and algorithms have been developed for this goal, each with pros and cons.

While artificial in terms of the limitations placed on the robot, maze-solving is an organized technique and regulated implementation of autonomous navigation that is occasionally favoured in researching specific elements of the issue. The implementation of a small mobile robot that navigates a maze using the flood-fill technique is described in this article. The maze-solving task is similar to those involved in the found in basic robotic competitions, in which robots compete to solve a maze as rapidly and efficiently as they can. From one corner to the center of a maze, a robot must move quickly. Although it is aware of the starting place and the final destination, it is not aware of any potential roadblocks.

The maze's starting point is on one of its corner cells, and its destination location is formed the correct entry path of the maze route. For exit, only one cell is opened in the maze route.

# Objective:

1. Design and build the robot using Ultrasonic sensor.

2. Understand and implement wall following algorithm.

3. Construct the Arduino based program to solve the maze.

# Components Required:

1. Arduino Uno Board

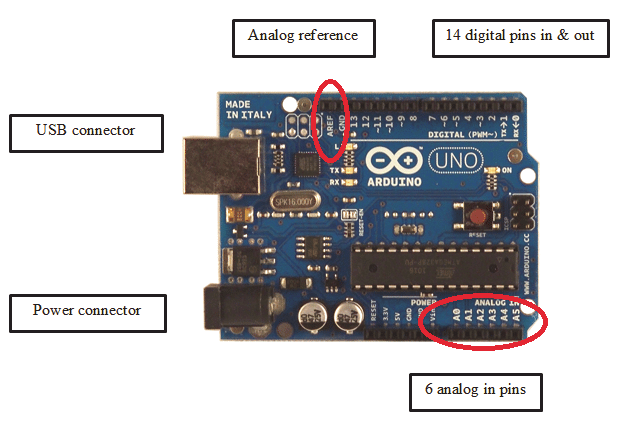


Fig.1 Arduino Uno Board

Microcontroller boards of the Arduino Uno variety were created by Arduino.cc. Arduino is primarily an open-source development board that commonly uses the Microchip ATmega328P microprocessor and is created using Arduino.cc. The board has a fixed number of input/output pins, both virtual and analog, that can be connected to various expansion forums and external circuits.

The board has 6 analog pins and 14 virtual pins that may be programmed using the Arduino IDE, which is an IDE (Integrated Development Environment) that is included with the board. A USB type B cable is used to burn the program. In order to power up the board's strategies, you may either attach a USB connection or a nine volt dc power supply. The range of acceptable voltages is from 7 to 20 volts.

2. Motor Driver

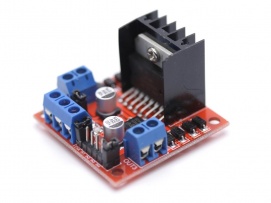


Fig.2 Motor Driver

Motor drives are circuits used to run a motor. In other words, they are commonly used for motor interfacing. These drive circuits can be easily interfaced with the motor and their selection depends upon the type of motor being used and their ratings (current, voltage).

3. Ultrasonic Sensors



Fig.3 Ultrasonic Sensor

Ultrasonic sensors are utilized to detect the gate's approach and departure from the railway crossing, accordingly. They are positioned on both sides of the railway crossing. The ultrasonic sensor provides an accurate distance in centimeters. It has 4 pins- trigger, echo, VCC and ground. The trigger pin is used to send a high frequency sound signal and when the signal hits the train, it reflects back and is received by the echo pin. As we already know the speed of sound, we can calculate the distance of the train by using it. In this system, 3 ultrasonic sensor has been used.

4. Chassis



Fig.4 Chassis

Chassis consist of the motor vehicle's wheels, machinery, and structure, which support the body.

5. DC Motors



Fig.5 DC Motor

A DC motor, also known as a direct current motor, is a type of electrical device that uses direct current to generate a magnetic field that converts electrical energy into mechanical energy. A magnetic field is produced in the stator of a DC motor when it is energized. Magnets on the rotor are drawn to and drawn away by the field, which rotates the rotor. The commutator, which is connected to brushes and the power source, supplies current to the motor's wire windings in order to keep the rotor turning continuously.

6. Battery (9V)



Fig.6 Battery

For this system, we needed 3 batteries so that it can power up the circuit.

7. Breadboard

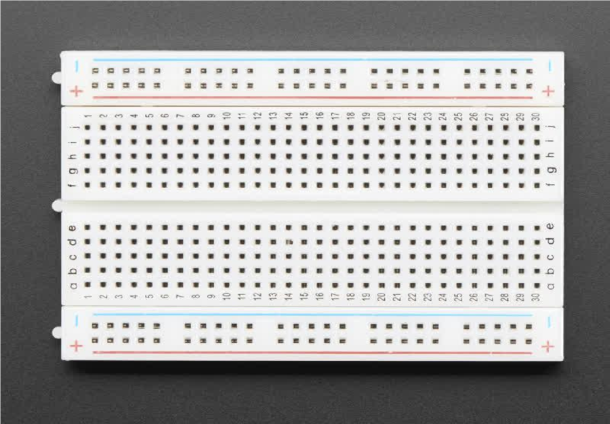


Fig.7 Breadboard, and a mini breadboard

A breadboard is a circuit board that is used to connect electronic components to single-board computers or microcontrollers. The connections aren't permanent; they can be deleted and reinstalled at any time. In, this project a mini breadboard was used

8. Tires and ball caster

A roll of toilet paper

Description automatically generated with low confidence

Fig.8 Ball Caster

A fixed pivot front wheel was used to make sure the chasis is balanced properly and the robot car can move freely

9. Maze:

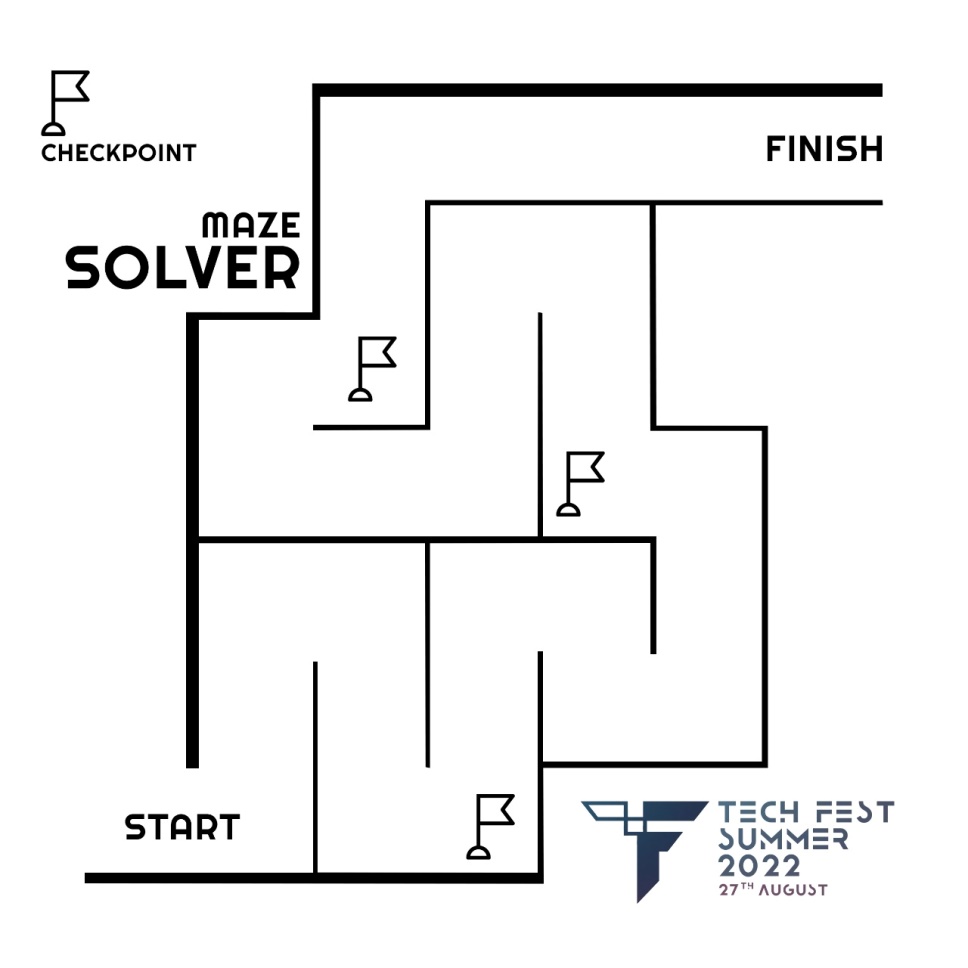
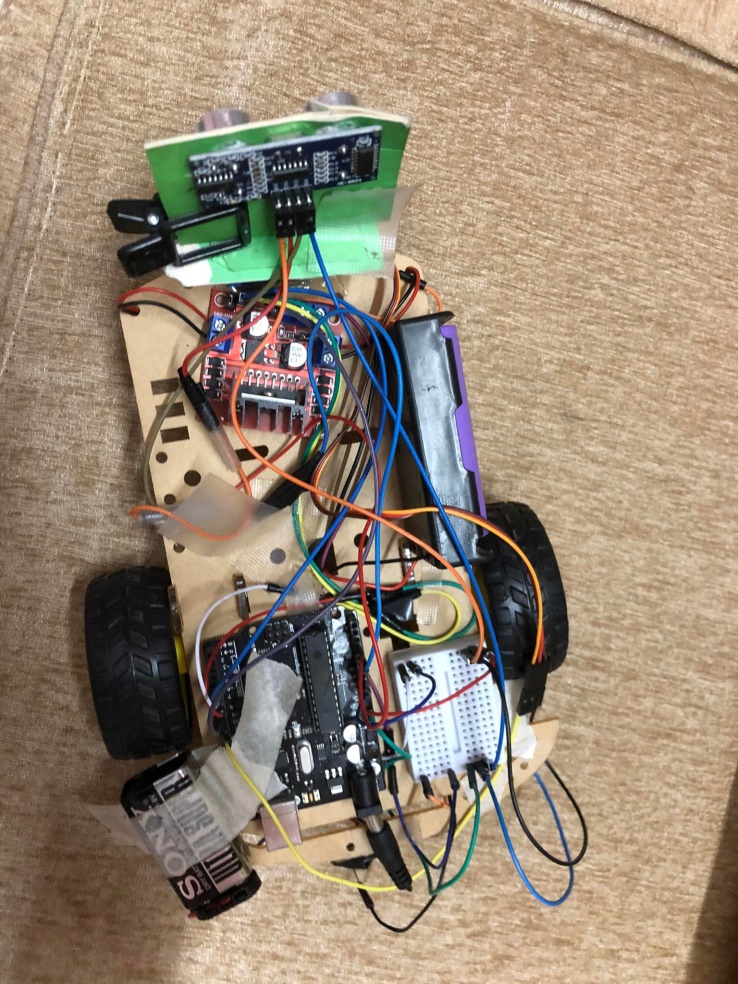
******

Fig.9 Maze

The maze we have used for the robot to solve is the diagram above. As observed in the diagram, the maze has exactly one exit and the robot uses the wall following algorithm to succeed.

# Maze Robot

******

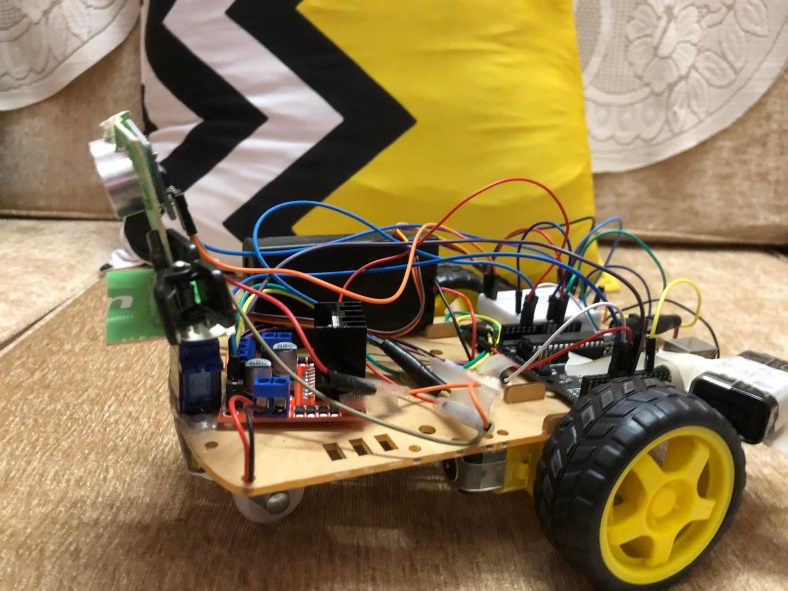
******

Fig.10 Maze Robot

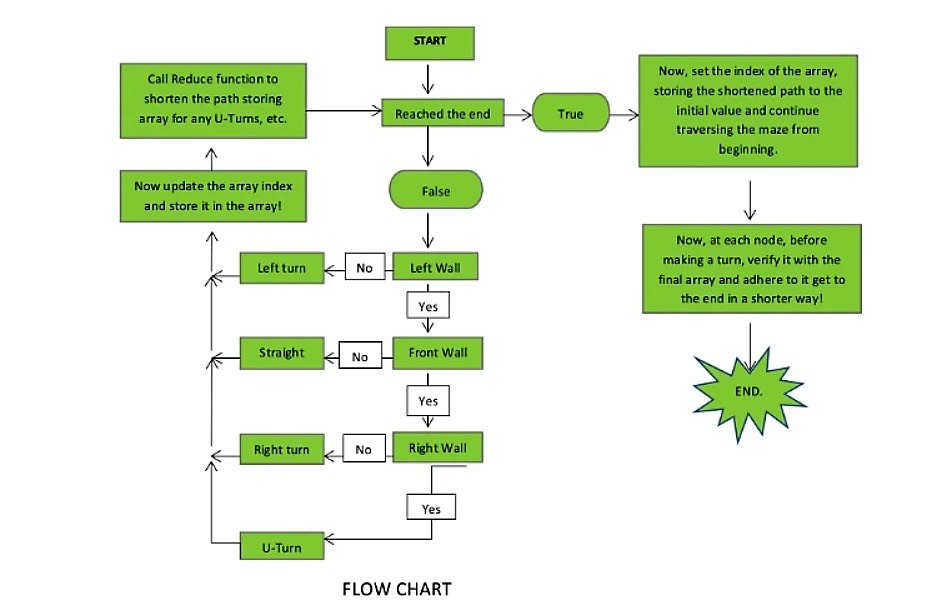
# Wall Following Algorithm

The wall follower is one of the most well-known maze navigational rules. Other names for it include the left-hand rule and the right-hand rule. The robot will determine which direction to go at a fork in the road by scanning for any gaps in the surrounding walls. Since the right wall has been selected in this instance, a U turn will take precedence over the right, straight, and left walls if all of them are closed. Three ultrasonic sensors that are attached to the robot—one on each side and one in the front—allow it to measure the distance between the walls in all three directions.

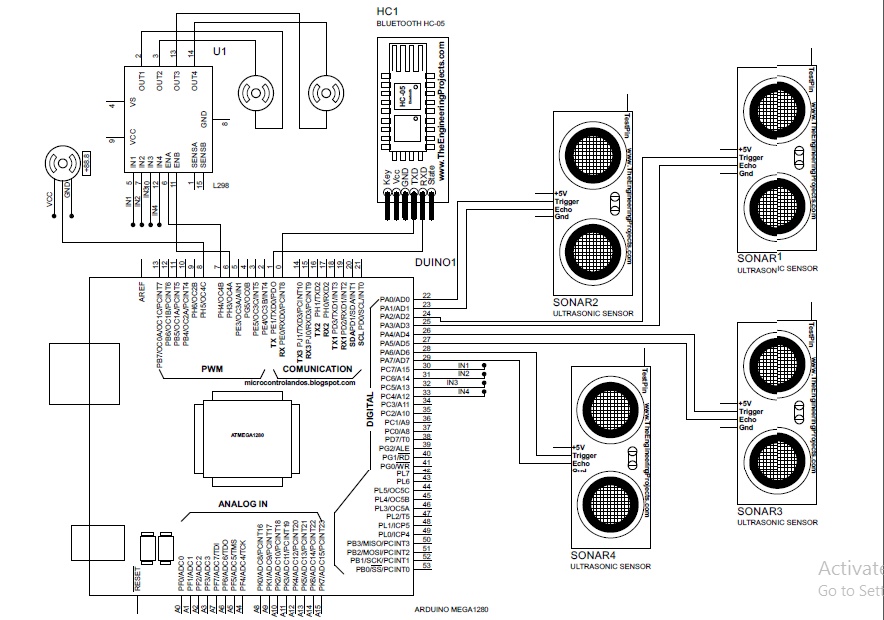
The bot moves through the maze while recording its path in an array and, if possible, reducing it at the same time by calling the function "reduce." For instance, if we take a look at the next part of the maze, this turn and its two predecessors (a U-turn and a left turn, respectively) can all be referred to as "R" turns! That is, "LUF = R," and the dynamically updated array now contains the new path. After reaching the destination, the bot restarts its journey from the beginning, but this time it compares each node to the condensed array and follows it to get there much faster! Thus, as the robot maneuvers through the maze, it learns, shortens, and adheres it to take a much shorter route to the conclusion!

As has been mentioned, the wall follower algorithm is a favorite among inexperienced programmers. This method works incredibly well for mazes whose start and end are wall connected, or, to put it another way, for mazes that are only connected but get caught in loops. There are many algorithms that have successfully avoided getting stuck in loops, but most of them operate in a similar way: as the bot navigates the maze, it keeps track of where it is and shortens and adheres the path to the end in a much shorter route. The Flood Fill Algorithm, which is frequently used in Intermediate level robot making Competitions, the Pledge Algorithm, the Tremaux's Algorithm, and others are examples of some of the algorithms.

# Flowchart



# Detailed Pin Diagram



# Code

The code can be viewed from this link or from the github repo.

https://github.com/AzwadFawadHasan/Maze\_SolverBot\_ArduinoProject

<https://pastebin.pl/view/7339c8a7>

Code Starts

#include <Servo.h> //Servo motor library. This is standard library

#include <NewPing.h> //Ultrasonic sensor function library. You must install this library

//our L298N control pins

const int LeftMotorForward = 7;

const int LeftMotorBackward = 6;

const int RightMotorForward = 4;

const int RightMotorBackward = 5;

//sensor pins

#define trig\_pin A1 //analog input 1

#define echo\_pin A2 //analog input 2

#define maximum\_distance 200

boolean goesForward = false;

int distance = 100;

NewPing sonar(trig\_pin, echo\_pin, maximum\_distance); //sensor function

Servo servo\_motor; //our servo name

void setup(){

pinMode(RightMotorForward, OUTPUT);

pinMode(LeftMotorForward, OUTPUT);

pinMode(LeftMotorBackward, OUTPUT);

pinMode(RightMotorBackward, OUTPUT);

servo\_motor.attach(10); //our servo pin

servo\_motor.write(115);

delay(2000);

distance = readPing();

delay(100);

distance = readPing();

delay(100);

distance = readPing();

delay(100);

distance = readPing();

delay(100);

}

void loop(){

int distanceRight = 0;

int distanceLeft = 0;

delay(50);

if (distance <= 20){

moveStop();

delay(300);

moveBackward();

delay(400);

moveStop();

delay(300);

distanceRight = lookRight();

delay(300);

distanceLeft = lookLeft();

delay(300);

if (distance >= distanceLeft){

turnRight();

moveStop();

}

else{

turnLeft();

moveStop();

}

}

else{

moveForward();

}

distance = readPing();

}

int lookRight(){

servo\_motor.write(50);

delay(500);

int distance = readPing();

delay(100);

servo\_motor.write(115);

return distance;

}

int lookLeft(){

servo\_motor.write(170);

delay(500);

int distance = readPing();

delay(100);

servo\_motor.write(115);

return distance;

delay(100);

}

int readPing(){

delay(70);

int cm = sonar.ping\_cm();

if (cm==0){

cm=250;

}

return cm;

}

void moveStop(){

digitalWrite(RightMotorForward, LOW);

digitalWrite(LeftMotorForward, LOW);

digitalWrite(RightMotorBackward, LOW);

digitalWrite(LeftMotorBackward, LOW);

}

void moveForward(){

if(!goesForward){

goesForward=true;

digitalWrite(LeftMotorForward, HIGH);

digitalWrite(RightMotorForward, HIGH);

digitalWrite(LeftMotorBackward, LOW);

digitalWrite(RightMotorBackward, LOW);

}

}

void moveBackward(){

goesForward=false;

digitalWrite(LeftMotorBackward, HIGH);

digitalWrite(RightMotorBackward, HIGH);

digitalWrite(LeftMotorForward, LOW);

digitalWrite(RightMotorForward, LOW);

}

void turnRight(){

digitalWrite(LeftMotorForward, HIGH);

digitalWrite(RightMotorBackward, HIGH);

digitalWrite(LeftMotorBackward, LOW);

digitalWrite(RightMotorForward, LOW);

delay(500);

digitalWrite(LeftMotorForward, HIGH);

digitalWrite(RightMotorForward, HIGH);

digitalWrite(LeftMotorBackward, LOW);

digitalWrite(RightMotorBackward, LOW);

}

void turnLeft(){

digitalWrite(LeftMotorBackward, HIGH);

digitalWrite(RightMotorForward, HIGH);

digitalWrite(LeftMotorForward, LOW);

digitalWrite(RightMotorBackward, LOW);

delay(500);

digitalWrite(LeftMotorForward, HIGH);

digitalWrite(RightMotorForward, HIGH);

digitalWrite(LeftMotorBackward, LOW);

digitalWrite(RightMotorBackward, LOW);

}

# Problem Faced

1. We rotate the robot using Dc motor. So, we make the robot take 90 degrees left or right turn by controlling the speed of Dc motor. But in reality, the speed of the motor depends highly on the nature of the surface and voltage it gets from the battery. So, in different surface the motor was rotating at different angles. It causes the robot to deviate from the trajectory it should maintain. As a result, the robot often make collision with the wall and can’t show the proper attributes.
2. Speed of the two Dc motors were different. So, we have to provide different speed for them even if in time of moving forward. So, in different surface it often deviates from the straight line. Although we make the correction by moving the robot a little bit left or right by taking reading from the sensors, but it cannot mitigate the deviation always. So the robot deviated from its trajectory also make inappropriate left or right turns and hit the wall. Moreover, the robot goes different distance in different instance although it should maintain a constant distance.
3. The robot take decisions by taking value from the left, right and from sensors and thus detect dead ends, and free paths. If it deviates from the original straight line and then the sensors reading are the diagonal distance from the wall which are not current. So it sometimes thought this distance to be a free path and make turn and thus collide.
4. The servo motor, when powered up always rotates a little automatically although we made the rotation standstill. Providing VCC to servo using software didn’t provide solution in this regard.

# Conclusion

As a conclusion, the maze solving algorithm have successfully been implemented in the robot

and the objectives of the project have been achieved. The first algorithm was wall following

algorithm. The basic method shows a good result for solving the maze.

So, an efficient method has been used to find the shortest path that is flood fill algorithm

method. After applying all methods, the robot was trained in a real maze. Several tests have been

run to ensure the best performance of the robot.

This project helps to improve various important information about robotics, knowledge about

many decision-making algorithms. It’s also helped to learn about many electronics components such as motor driver, sensors, etc. This gained knowledge will have a

significant impact on future work.