Maze Solver Bot Using Arduino Microcontroller

Design and Implementation of an autonomous bot to keep the paths on memory and to solve the mazes

Durai Murugan N
Dept. of Electronics and Communication Engineering
College of Engineering Guindy
Chennai, India
Email: duraimurugan122003@gmail.com

Bharath Kannan L
Dept. of Electronics and Communication Engineering
College of Engineering Guindy
Chennai, India
Email: bharathkannan2412@gmail.com

Raja Bharathi N
Dept. of Electronics and Communication Engineering
College of Engineering Guindy
Chennai, India
Email: rajabarathi.in@gmail.com

Rameshkumar S
Dept. of Electronics and Communication Engineering
College of Engineering Guindy
Chennai, India
Email: rameshkumarsakthivel2004@gmail.com

In this paper, the design, implementation and application of a Maze Solver bot using Arduino microcontroller has been explored.

***Abstract*—**Autonomous navigation is a critical technology enabling robots to independently traverse unfamiliar areas, circumventing human limitations and potential dangers. This paper presents the development of an algorithm for enhanced maze navigation, building upon a previous project.

Utilizing ultrasonic range-finders, the robot adeptly detects walls and openings within the maze. The algorithm facilitates the robot's capacity to learn the maze, explore all feasible routes, and efficiently solve it by choosing the shortest path. This work contributes to advancing autonomous navigation algorithms, offering insights into robot behavior and intelligence in dynamic environments.

I. INTRODUCTION

In the realm of robotics, the development of autonomous systems capable of navigating intricate environments is a compelling pursuit. Among the myriad challenges in this domain, the maze-solving problem stands out as a quintessential test of a robot's ability to perceive its surroundings and make intelligent decisions. This project embarks on the design and implementation of a maze-solving robot, leveraging the versatility and programmability of the Arduino microcontroller. The endeavor aims to showcase the integration of sensory input, algorithmic decision-making, and precise actuation to enable the robot's autonomous traversal through complex mazes. By delving into this project, we not only explore the fascinating intersection of robotics and artificial intelligence but also contribute to the ever-evolving landscape of intelligent automation.

II. COMPONENTS USED

A. ARDUINO MICROCONTROLLER

The Arduino Integrated Development Environment (IDE) serves as the foundational software platform for programming Arduino microcontrollers, providing a user-friendly environment for both beginners and experienced developers. Lauded for its simplicity and accessibility, the Arduino IDE empowers enthusiasts to seamlessly write, compile, and upload code to Arduino boards. Its open-source nature fosters a vibrant community that continually expands the library of pre-built functions and code snippets, facilitating rapid prototyping and experimentation. With a straightforward interface and a simplified version of the C++ programming language, the Arduino IDE plays a pivotal role in democratizing electronics and enabling individuals to bring their ideas to life through the seamless integration of hardware and software. Whether crafting intricate robotics projects or simple electronic prototypes, the Arduino IDE stands as a catalyst for innovation, breaking down barriers and

empowering creators to realize their visions in the realm of embedded systems.

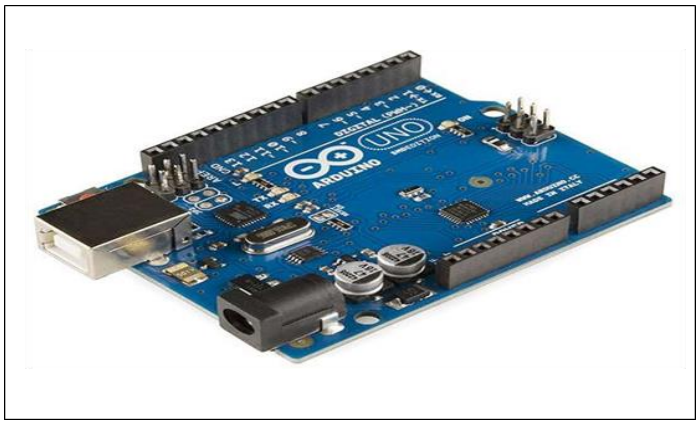

FIGURE 1.1 ***ARDUINO MICROCONTROLLER***

B. LN298N MOTOR DRIVER MODULE

The LN298N is a popular and versatile dual H-bridge motor driver integrated circuit widely employed in robotics and electronics projects. Renowned for its ability to control the speed and direction of DC motors, the L298N is particularly valued in applications requiring precise motor control. Its dual-bridge configuration allows the independent control of two motors, making it suitable for applications such as wheeled robot movement, motorized platforms, and various mechatronic systems. Equipped with built-in diodes for protection against back electromotive force, the L298N is designed for efficiency and reliability. Additionally, its compatibility with microcontrollers like Arduino and Raspberry Pi makes it a preferred choice for hobbyists and engineers seeking a robust solution for motor control in their projects. The L298N's straightforward pin configuration, thermal protection, and robust design contribute to its popularity, positioning it as an integral component in the toolkit of electronics enthusiasts and professionals alike.

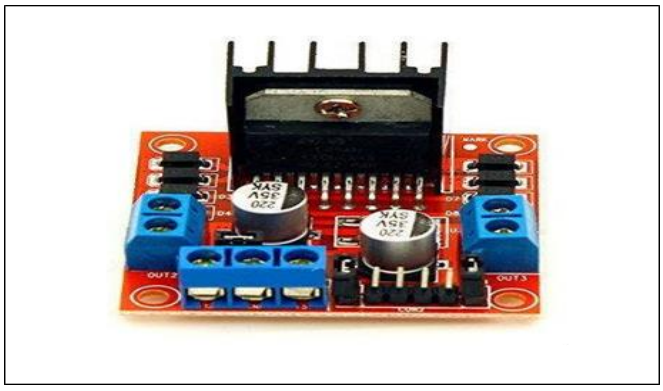

FIGURE 1.2 ***LN298N DRIVER MODULE***

C. ULTRA SONIC SENSOR

The HC-SR04 ultrasonic sensor is a fundamental and widely utilized component in the realm of electronics and robotics. This compact module employs ultrasonic waves to measure distance, making it a popular choice for applications requiring non-contact distance sensing. The HC-SR04 sensor consists of a transmitter and receiver pair, working collaboratively to emit ultrasonic pulses and calculate the time it takes for these pulses to bounce back after hitting an obstacle. This time measurement is then translated into a distance reading, providing precise and real-time information about the sensor's surroundings. With its straightforward interface and compatibility with microcontrollers like Arduino, the HC-SR04 is a go-to choice for projects involving obstacle avoidance, proximity detection, and automation. Its affordability, reliability, and ease of integration contribute to its ubiquity in hobbyist projects and professional applications alike, making the HC-SR04 an essential tool for enthusiasts exploring the realms of sensor-based electronics and robotics.

FIGURE 1.3 ***ULTRASONIC MOTOR***

D. DC MOTOR

The dual shaft 300 RPM motor stands as a robust and versatile electromechanical device, finding extensive utility in the realms of robotics and industrial automation. Renowned for its formidable rotational speed of 300 revolutions per minute, this motor delivers a compelling blend of power and precision. The dual shaft configuration enhances its adaptability, facilitating the concurrent operation of two distinct mechanisms or attachments, thereby broadening its scope for use in intricate applications requiring synchronized motion. This motor's compatibility with popular microcontrollers such as Arduino and Raspberry Pi underscores its seamless integration into a myriad of projects, spanning from mobile robotic platforms to conveyor systems. Its reliability, operational efficiency, and moderate rotational speed render it well-suited for scenarios demanding a judicious balance between torque and agility. In the repertoire of both engineers

and hobbyists, the dual shaft 300 RPM motor emerges as a pivotal component, emblematic of cutting-edge advancements in electromechanical design for contemporary automation endeavors.

FIGURE 1.4 ***DC MOTOR***

III. IMPLEMENTATION OF ALGORITHM

The algorithm employed for maze-solving bots is crucial in determining the efficiency and effectiveness of their navigation through complex labyrinthine structures. One commonly utilized algorithm is the Depth-First Search (DFS) or its variant, the Recursive Backtracker. DFS explores potential paths in a maze by traversing as far as possible along each branch before backtracking, effectively mapping out the entire maze and identifying potential dead ends. Another widely used algorithm is the Breadth-First Search (BFS), which explores the maze layer by layer, ensuring the shortest path is found. Additionally, the A* (A-star) algorithm combines the benefits of both heuristic and systematic search methods, considering the cost of reaching a particular point along with an estimate of the remaining distance to the goal. These algorithms are implemented with the integration of appropriate sensors, enabling the maze-solving bot to interpret its environment and make informed decisions on the optimal path. The selection of the algorithm depends on the specific requirements of the maze-solving task, balancing factors such as computational efficiency and real-time responsiveness.

IV. WORKING IMPLEMENTATION:

At the heart of our maze-solving robot lies a sophisticated algorithm, meticulously crafted to seamlessly guide the robot through the intricacies of a maze. This algorithm orchestrates the robot's movements by interpreting data from ultrasonic sensors, facilitating real-time decision-making for autonomous navigation.

1. Initiating the Journey:

The algorithm's journey begins with the activation of the ultrasonic sensors, strategically placed on the robot's front, left, and right sides.

These sensors, specifically HC-SR04 ultrasonic modules, emit ultrasonic pulses and measure the time it takes for these pulses to return after bouncing off obstacles. This fundamental principle of echolocation forms the basis for the robot's awareness of its surroundings**.**

2. Obstacle Detection:

The primary objective of the algorithm is to detect obstacles in the robot's path and respond accordingly. The ultrasonic sensors provide distance readings in centimeters for the front (center), left, and right directions.

The algorithm interprets these readings to ascertain the presence of obstacles. If all three directions indicate distances greater than a predefined threshold, the algorithm determines that the path is clear. This prompts the robot to stop, signaling the completion of the maze or the need to assess the next move.

3. Decision-Making Logic:

When an obstacle is detected, the algorithm employs a logical decision-making process to navigate the robot. Each sensor's input triggers specific actions, enabling the robot to adapt to its environment dynamically.

If the right sensor detects an obstacle or fails to provide a valid reading, the algorithm instructs the robot to execute a right turn. This maneuver allows the robot to veer away from the detected obstacle.

Conversely, if the front sensor indicates an obstacle or provides no valid reading, the algorithm commands the robot to move straight ahead. This decision assumes an open path in the forward direction, guiding the robot through the maze.

In the event that the left sensor detects an obstacle or fails to produce a valid reading, the algorithm directs the robot to execute a left turn. This maneuver enables the robot to navigate around potential barriers.

4. Handling Dead Ends:

Dead ends present a unique challenge in maze-solving scenarios. The algorithm is equipped to identify dead ends by analyzing sensor data. When the robot encounters a dead end, indicated by obstacles in all three directions, the algorithm triggers a U-turn.

The U-turn allows the robot to retrace its steps and explore alternative routes, preventing it from becoming stuck in a maze cul-de-sac.

5. Dynamic Adaptability:

A key strength of the algorithm lies in its dynamic adaptability. The robot's decisions are not predetermined but are made in real-time based on the most recent sensor data.

This adaptability ensures that the robot can navigate through changing maze conditions, adjusting its course dynamically as it encounters new obstacles or pathways.

6. Error Handling:

Real-world applications inevitably introduce uncertainties and potential inaccuracies in sensor readings. To counteract this, the algorithm incorporates robust error-handling mechanisms. When faced with invalid or erratic sensor readings, the algorithm employs strategies to filter out unreliable data points. By prioritizing reliable measurements and disregarding outliers, the algorithm maintains a high level of accuracy in decision-making.

In essence, our maze-solving algorithm functions as the intelligence behind the robot's movements. It interprets the language of ultrasonic sensor data, translating it into a series of precise and calculated decisions that propel the robot through the maze. Its simplicity belies its effectiveness, providing a versatile solution for autonomous maze navigation.

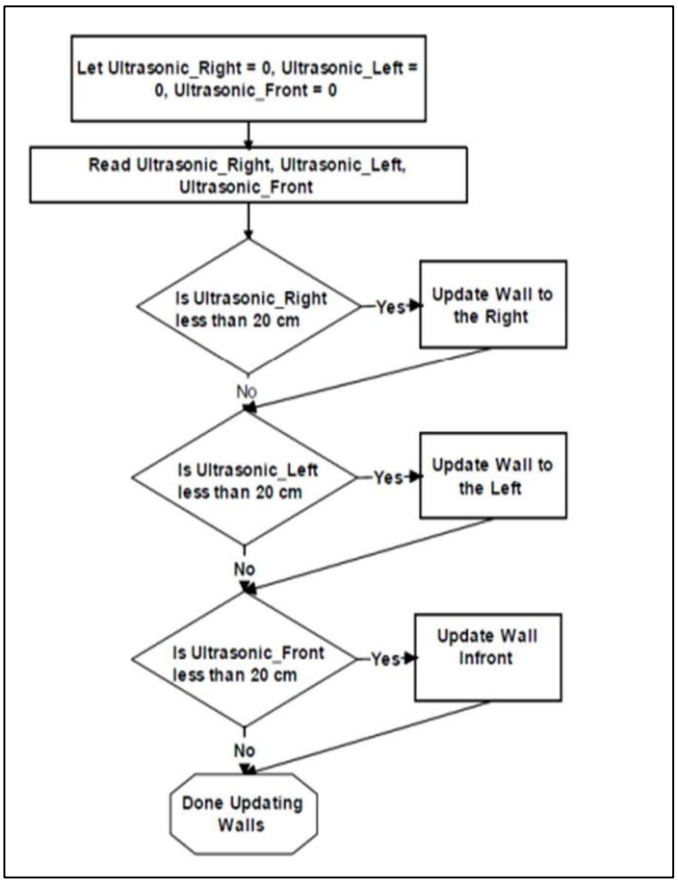

FIGURE 1.5 ***FLOW CHART IMPLEMENTATION OF WORKING***

V. APPLICATIONS

1. **Industrial Automation:**

The bot's navigation algorithms find application in industrial automation, aiding tasks like warehouse management where precise navigation through complex environments is essential.

2. **Pipeline Instructions:**

In industries like infrastructure and utilities, the bot can be employed for inspecting pipelines and structures with intricate layouts, ensuring thorough and efficient inspections.

3. **Autonomous Vehicles and Drones:**

The bot's navigation techniques can be adapted for autonomous vehicles and drones, contributing to their ability to navigate complex and dynamic environments.

4. **Smart Homes and IOT:**

Integration into smart home systems and IoT devices allows the bot to contribute to home automation, guiding robotic devices and monitoring security in maze-like home environments

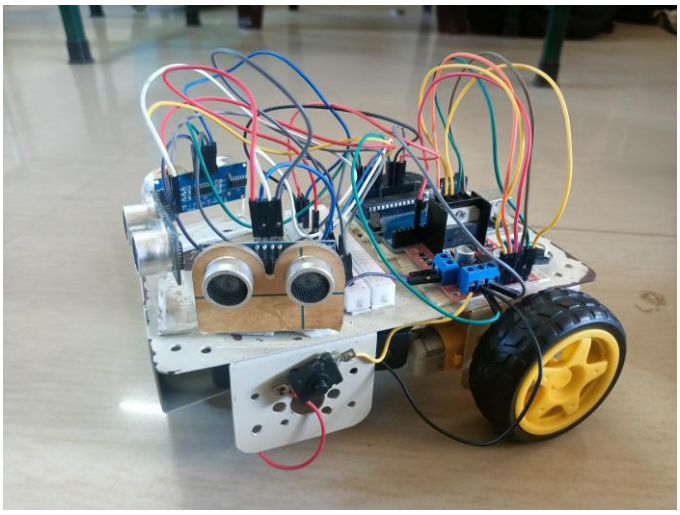

FIGURE 1.6 ***FINAL OUTPUT***

VI. CONCLUSION:

In conclusion, the Maze Solver Bot has demonstrated its exceptional problem-solving capabilities and adaptability, marking a significant stride in the realm of robotics. Its successful navigation through various mazes underscores its efficiency and effectiveness, positioning it as a valuable tool for education and robotics competitions. The bot's potential applications in industrial automation, search and rescue operations, entertainment, and more, highlight its versatility.

Moreover, its modular design suggests a promising avenue for customizable solutions, making it a potential asset across diverse industries. As the Maze Solver Bot charts new paths in the integration of artificial intelligence and robotics, it not only solves mazes but also points toward a future where autonomous systems can address complex challenges in innovative ways.

ARDUINO CODE:

```
const int in1=3;
const int in2=4;
const int enA=9;
const int in3=13;
const int in4=12;
const int enB=10;
const int trig1=A0;//centre
const int echo1=A1;
const int trig2=A2;//left
const int echo2=A3;
const int trig3=A4;//right
const int echo3=A5;
long distance1;
long time1;
long distance2;
long time2;
long distance3;
long time3;
//int obsdist;
void setup() {
// put your setup code here, to run once:
pinMode(in1,OUTPUT);
pinMode(in2,OUTPUT);
pinMode(in3,OUTPUT);
pinMode(in4,OUTPUT);
pinMode(enA,OUTPUT);
pinMode (enB,OUTPUT);
pinMode(trig1,OUTPUT);
pinMode(echo1,INPUT);
pinMode(trig2,OUTPUT);
pinMode(echo2,INPUT);
pinMode(trig3,OUTPUT);
pinMode(echo3,INPUT);
Serial.begin(9600);
}

void go_straight(){
digitalWrite(in1,LOW);
digitalWrite(in2,HIGH);
analogWrite(enA,60);
digitalWrite(in3,LOW);
digitalWrite(in4,HIGH);
analogWrite(enB,60);
delay(150);//need to be checked
}

void turn_left() {
//Stop left and run only right tyre
digitalWrite(in1,LOW);
digitalWrite(in2,LOW);
analogWrite(enA,0);
digitalWrite(in3,LOW);
digitalWrite(in4,HIGH);
analogWrite(enB,60);
delay(700);
digitalWrite(in1,LOW);
digitalWrite(in2,LOW);
analogWrite(enA,0);
digitalWrite(in3,LOW);
digitalWrite(in4,LOW);
analogWrite(enB,0);
delay(500);//need to be checked

void turn_right() {
//Stop right and run only left tyre
digitalWrite(in1,LOW);
digitalWrite(in2,HIGH);
analogWrite(enA,60);
digitalWrite(in3,LOW);
digitalWrite(in4,LOW);
analogWrite(enB,0);
delay(700);//need to be checked
digitalWrite(in1,LOW);
digitalWrite(in2,LOW);
```

```
analogWrite(enA,0);
digitalWrite(in3,LOW);
digitalWrite(in4,LOW);
analogWrite(enB,0);
delay(500);
}

void turn_180() {
//Stop left and run only right
tyre(backwards)
digitalWrite(in1,LOW);
digitalWrite(in2,LOW);
analogWrite(enA,0);
digitalWrite(in3,HIGH);
digitalWrite(in4,LOW);
analogWrite(enB,60);
delay(1600);//need to be checked
digitalWrite(in1,LOW);
digitalWrite(in2,LOW);
analogWrite(enA,0);
digitalWrite(in3,LOW);
digitalWrite(in4,LOW);
analogWrite(enB,0);
delay(500);
}

void stop_() {
//Stop left and run only right
tyre(backwards)
digitalWrite(in1,LOW);
digitalWrite(in2,LOW);
analogWrite(enA,0);
digitalWrite(in3,LOW);
digitalWrite(in4,LOW);
analogWrite(enB,0);
delay(5000);//need to be checked
}
void loop() {
//Measure distance of obstacle from front
sensor
digitalWrite(trig1,HIGH);
delayMicroseconds(10);
digitalWrite(trig1,LOW);
time1=pulseIn(echo1,HIGH);
distance1=time1*0.017;
Serial.print("distance1 in cm");
Serial.println(distance1);
digitalWrite(trig2,HIGH);
delayMicroseconds(10);
digitalWrite(trig2,LOW);
time2=pulseIn(echo2,HIGH);
distance2=time2*0.017;
Serial.print("distance2 in cm");
Serial.println(distance2);
//Measure distance of obstacle from right
sensor digitalWrite(trig3,HIGH);
delayMicroseconds(10);
digitalWrite(trig3,LOW);
time3=pulseIn(echo3,HIGH);
distance3=time3*0.017;
Serial.print("distance3 in cm");
Serial.println(distance3);

if (distance3>17 && distance2>17
&&distance1>17){
// Stops in OPEN SPACE
stop_();
}

else if(distance3>17 || distance3==0){
//If right side is open to move, turn right
digitalWrite(in1,LOW);
digitalWrite(in2,LOW);
analogWrite(enA,0);
digitalWrite(in3,LOW);
digitalWrite(in4,LOW);
```

```
analogWrite(enB,0);
delay(1000);
turn_right();
}

else if (distance1>25 || distance1==0 ){
//If right side is closed and straight
route is free to move, go straight
go_straight();
}

else if ( (distance2>17 || distance2 ==0)){
//If right side and straight both are
closed and left side is free to move, turn
left.
digitalWrite(in1,LOW);
digitalWrite(in2,LOW);
analogWrite(enA,0);
digitalWrite(in3,LOW);
digitalWrite(in4,LOW);
analogWrite(enB,0);
delay(1000);
turn_left();
}

else {
//If all sides are closed, turn backwards
turn_180();
  }
}
```