CHAPTER 1: INTRODUCTION

In today's modern world robotics is the one of the fastest growing and fascinating field. Autonomous robots becoming increasingly common in various industries. With the increasing demand for automation, robots are being employed in a wide range of applications. Autonomous robots are one of the most fascinating applications of robotics. They have the ability to make their own decisions and perform tasks without any human intervention. One of the critical challenges in developing autonomous robots is to design efficient and effective algorithms that enable the robots to navigate through complex environments while avoiding obstacles and reaching their goals. There are many issues associated with autonomous driving systems in practical applications. However, these issues can be solved in the near future [8].

1.1 Overview of Autonomous Robots

Autonomous robots are designed to perform tasks without any human intervention. They are equipped with sensors, processors, and actuators that enable them to sense their environment, make decisions, and perform tasks. Autonomous robots are being used in a wide range of applications such as manufacturing, logistics, agriculture, healthcare, and space exploration.

1.2 Path Planning and Obstacle Avoidance

Path planning is the process of finding the optimal path from a starting point to a destination point while avoiding obstacles. In autonomous robots, path planning is a critical task that enables the robot to navigate in its environment. Path planning algorithms take into account factors such as the size and shape of the robot, the location of obstacles, and the goal location to generate an optimal path. Obstacle avoidance is one of the most important tasks for various robotics system. Along with obstacle avoidance, path planning is also an important criterion to measure the efficiency of robot in terms of distance travelled by choosing a particular path out of all the available paths. Obstacle avoidance in robots will bring more flexibility in manoeuvring in different environmental conditions and would be much more efficient as continuous human monitoring is not required. So, the robot is designed in a way to traverse through an unknown environment having obstacles at certain coordinates. This problem has several applications in robotics, including autonomous navigation, industrial automation, and military robotics.

The real-time implementation of optimal path planning algorithms involves finding the most efficient path for a robot to travel from its current location to its destination in real-time. This real-time implementation requires the use of algorithms that can quickly compute the optimal path and update it as the robot navigates through its environment.

1.3 Implementation And Methods

The robot is designed using components such as Arduino-UNO micro-controller, Bridge motor driver L298, DC Motor, Omni-direction wheel, jumper wires and 9V-batteries. The overall robot body is designed in compact manner, so it is light-weighted. It will be easily able to move on the floor of a room, tackling all the obstacles of both big and small sizes. The robot uses 2 motors each with its gearbox, 2 wheels per motor and 1 omni-direction wheel which will make the robot change its direction.

Many methods have been developed in this field of robot obstacle avoidance including path-following, wall-following, edge detection and many others. One of the methods uses an ultrasonic sensor mounted in front of robot which detects obstacles in front and send commands to the Arduino micro-controller to direct the robot to another possible path. This robot explores each possible path until it reaches its destination. This is a collision detection technique.

There is another technique called “Collision Avoidance” which is used in this project. In a collision avoidance problem, the end-effector of each manipulator is guided iteratively towards the destination and away from the obstacles using an approximate grid decomposition technique with a greedy Depth-first-Search Algorithm. The Collision Avoidance scheme is an on-line scheme which uses the local knowledge and operates in a cartesian space. The computations for the shortest-path are done in the pre-processing step and this path is used for the traversal by robot.

The advantage of this method is that robot can find a way faster as it has to just traverse from a particular pre-defined path. The environment will be given in the form of grid or matrix where obstacle is represented as a particular integer and empty space as another integer. The algorithm will return the shortest valid path through which robot can move, thereby making the process simpler and faster.

CHAPTER 2: LITERATURE REVIEW

Autonomous mobile robots performing navigation-based tasks in unknown environments require a mechanism to detect and avoid obstacles to ensure their safety and task continuity. The problem of obstacle detection and avoidance for mobile robots is a challenging task and a well-researched topic in robotics. Various sensors and methods have been proposed in previous works to address this problem. Students and practitioners have built different obstacle detection robot mechanisms to overcome this challenge.

Obstacle avoidance is a crucial function for autonomous mobile robots in real-world environments. To detect and avoid encountered obstacles, a range of sensors and methods have been proposed in the literature. However, despite the extensive research, the problem of obstacle detection and avoidance remains a challenging task for mobile robots.

The problem involves finding the most efficient path for a robot to travel from its current location to its destination whole satisfying certain constraints such as avoiding obstacles and minimizing travel time.

2.1 The Obstacle Avoidance System

Obstacle avoidance bot was developed and designed by Aamir attar, Adil Ansari, Abhishek Desai, to design a robot that detects obstacle in the path and follow the instructions provide by the user. So, this system provides an alternate way to the existing system by replacing skilled labour with robotic machinery, which in turn can handle more things in less time with good accuracy and lower cost.

Autonomous vehicles use radar, GPS, computer vision, odometry and lidar to detect it surroundings.

This literature review will provide an overview of some of the most common obstacle avoidance systems.

- **Ultrasonic Sensors:**

Ultrasonic sensors are commonly used in obstacle avoidance systems due to their low cost and ease of implementation. Ultrasonic sensors work by emitting high-frequency

sound waves that bounce off objects in the robot's environment. The sensors then measure the time it takes for the sound waves to return, which is used to determine the distance to the object. However, ultrasonic sensors have limited range and can only detect obstacles within a few meters.

- **Infrared Sensors:**

Infrared sensors are also commonly used in obstacle avoidance systems. They work by emitting infrared light and measuring the amount of light that is reflected back from objects in the environment. Infrared sensors have longer range than ultrasonic sensors, but they can be affected by ambient light and may not work well in bright sunlight.

- **Lidar Sensors:**

LIDAR (Light Detection and Ranging) sensors are becoming increasingly popular in obstacle avoidance systems. LIDAR sensors work by emitting laser beams and measuring the time it takes for the laser beams to reflect back from objects in the environment. LIDAR sensors have long range and high accuracy, making them ideal for use in autonomous vehicles. However, LIDAR sensors are relatively expensive and may not be suitable for use in low-cost robots.

- **Computer Vision:**

Computer vision is another approach to obstacle avoidance that is gaining popularity. Computer vision systems use cameras to capture images of the robot's environment and then use image processing techniques to detect obstacles. Computer vision systems have the advantage of being able to detect a wide range of obstacles, but they can be affected by changes in lighting conditions and may not work well in low light.

- **Hybrid Systems:**

Hybrid obstacle avoidance systems combine multiple sensors and techniques to improve accuracy and reliability. For example, a system may use LIDAR sensors for long-range detection and computer vision for short-range detection. Hybrid systems can provide robust obstacle detection and avoidance, but they can be more complex and expensive to implement.

In conclusion, there are several obstacle avoidance systems available, each with its own advantages and limitations. The choice of system will depend on the requirements of the specific application and the available budget. Ultrasonic sensors and infrared sensors are low-cost options, while LIDAR sensors and computer vision systems offer high accuracy and range. Hybrid systems can provide the best of both worlds but may be more complex and expensive to implement. Ultimately, the goal of obstacle avoidance systems is to ensure the safety and efficiency of autonomous robots and vehicles.

2.2 Obstacle-Avoiding Robot with IR And PIR Motion Sensors

It was developed by Aniket D. Adhvaryu et al. In this robot, platform is general wheeled autonomous platform. These robots can be used in educational field, research etc. Many students used it to learn various skills such as C++, Arduino Uno 1.6.5 compiler, etc.

These robots had very flexible design. It is observed that PIR sensors are more sensitive compared to IR sensors.[1]

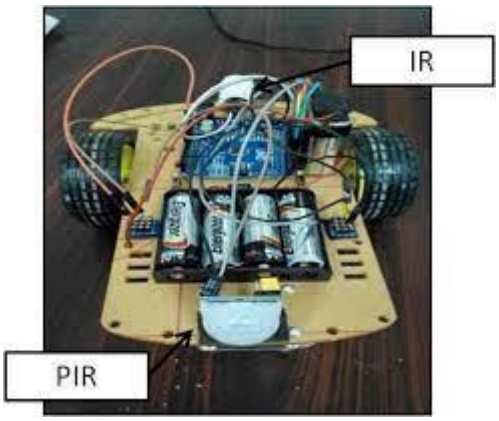


Fig 2.1 - Prototype of obstacle avoiding robot using IR and PIR motion sensors [1]

The obstacle-avoiding robot with IR and PIR motion sensors is a popular and effective system that utilizes both infrared (IR) and passive infrared (PIR) sensors to detect obstacles and avoid collisions. This literature review will provide an overview of the obstacle-avoiding robot with IR and PIR motion sensors, including its design, features, and limitations.

In conclusion, the obstacle-avoiding robot with IR and PIR motion sensors is a popular and effective system for obstacle avoidance in autonomous robots. Its

low cost and simplicity make it accessible to hobbyists and enthusiasts, while its ability to navigate through a variety of environments makes it a useful tool for research and development. However, its limitations in range and the ability to detect certain types of obstacles should be taken into consideration when designing and implementing the system.

2.3 Obstacle avoidance robotic vehicle using ultrasonic sensor, android, and Bluetooth for obstacle detection.

It was developed and designed by Vaghela et. Al. While designing object avoidance robotic vehicle various methodologies are analysed and reviewed along with their pros and cons. Ultrasonic sensors was used to design the robot that provide various features such as light weight, portability etc. it needs to acquire more focus in relevant areas of applications like home appliances, wheelchairs, artificial nurses, tabletop screens etc. in a collaborative manner. [2]

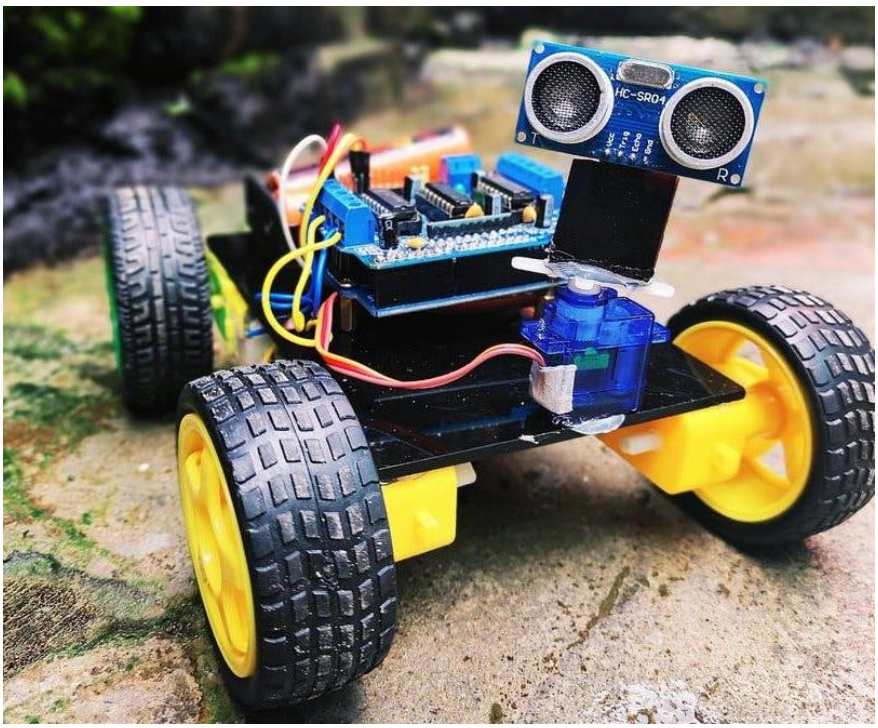


Fig 2.2 - Prototype of obstacle avoiding robot using ultrasonic sensor, android and Bluetooth for obstacle detection [2]

Several algorithms have been proposed in the literature for optimal path planning, and the real-time implementation of these algorithms has been investigated extensively. In this literature review, we will provide an overview of the various algorithms used in optimal path planning and their suitability for real-time implementation.

CHAPTER 3: OBJECTIVES

Following are the objectives of this project –

1. **Designing the Robot:** The first objective of the project is to design a robot using the Arduino-UNO micro-controller, bridge motor driver L298, DC motor, omni-direction wheel, jumper wires, and 9V-batteries. The robot will be designed to avoid obstacles and move towards the desired destination by following the shortest possible path.
2. **Learning about Obstacle Avoidance Algorithms:** The second objective of the project is to learn about various obstacle avoidance algorithms such as recursion, backtracking, depth-first search, Dijkstra algorithm, and A-star algorithm. The project team will study the concepts and working principles of each algorithm in detail and choose the most efficient algorithm for their robot.
3. **Choosing the Best Algorithm:** The third objective of the project is to choose the best algorithm that can compute the shortest possible valid path and avoid obstacles efficiently. The project team will evaluate each algorithm based on its efficiency, accuracy, and reliability and select the best algorithm that meets their requirements.
4. **Interfacing of Arduino and Writing Code:** Once the algorithm is finalized, the fourth objective of the project is to interface Arduino with the robot and write code in Arduino Development Integrated Environment. The code will be written based on the selected algorithm to enable the robot to navigate through the environment and avoid obstacles.
5. **Designing the User Interface:** The fifth objective of the project is to design a user interface with the help of web development techniques. The user interface will give a visual description of the computed optimal path and help the user to monitor the robot's movement.
6. **Testing and Analysis:** The final objective of the project is to test the robot in different environments with obstacles positioned differently each time. The results of each test will be computed and analysed to determine the efficiency and accuracy of the robot. The project team will also evaluate the performance of the robot in different environments and identify areas of improvement.

In summary, the project objectives involve designing a robot that can navigate through an environment, avoid obstacles, and follow the shortest possible path. The project team will evaluate various algorithms and select the best one to achieve this objective. They will interface Arduino with the robot and write code based on the selected algorithm. Additionally, they will design a user interface to visualize the optimal path. The project team will also explore the future aspects of the project and test the robot's efficiency and accuracy in different environments.

CHAPTER 4: PRELIMINARIES

This project involves implementation of optimal path planning for autonomous robot which is designed using various hardware components. The components of the model, block diagram and hardware requirements are described as follows:

4.1 Components

There are various hardware components required to design the structure of the autonomous Robot.

- Arduino UNO
- Motor Driver (L298N)
- DC Motor
- Batteries(9V)
- Wires and connectors

4.2 Block Diagram

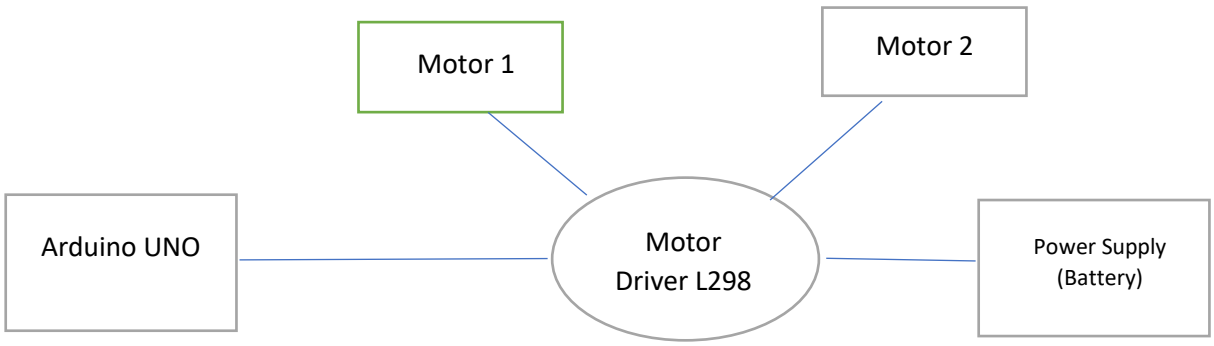


Fig 4.1 – Block Diagram of Model

Following figure shows the basic block diagram of an autonomous robot which comprises of mainly following components -

1. Arduino UNO
2. Motor Driver (L298N)
3. DC Motor

4.3 Hardware Requirements

4.3.1 Arduino UNO

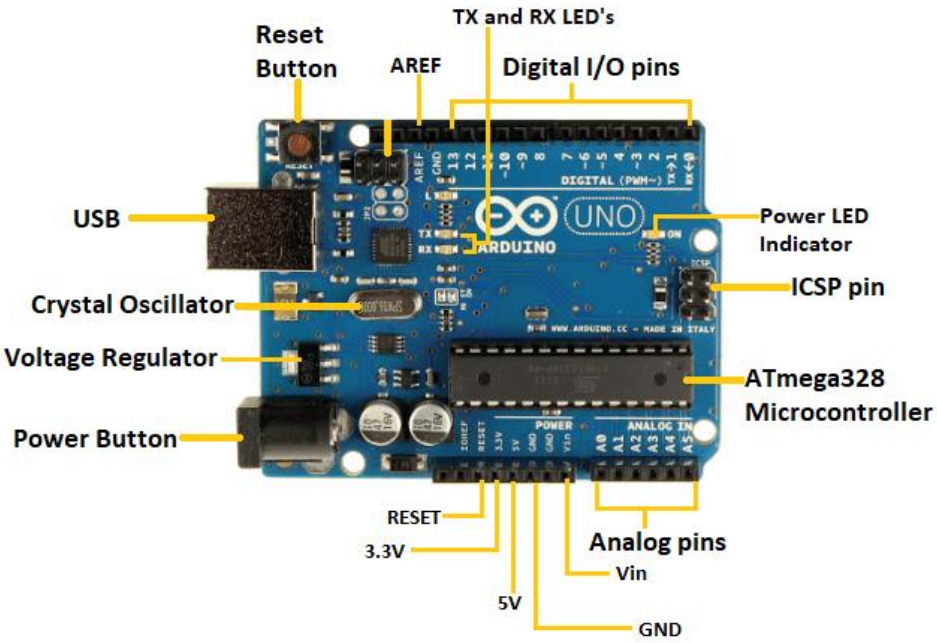


Fig 4.2 - Arduino UNO

The Arduino UNO is an open-source microcontroller board that is popularly used in robotics, automation, and various electronic projects. It is a widely used as an open-source microcontroller board which is based on the **ATmega328P** microcontroller. It was designed by the Arduino team and designed to be easy to use, flexible, and affordable. It has been a popular choice for beginners in the field of electronics and programming, especially for those who are interested in building robots. Its low cost, ease of use, and flexibility make it a preferred choice among hobbyists, students, and professionals.

4.3.1.1 Features And Functionalities

Arduino Uno is a compact board with 14 digital input/output pins, six analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, and an ICSP header. The digital pins on the board can be configured as input or output pins, which makes it possible to interface with a variety of devices such as sensors, switches, LEDs, motors, and servos. The analog inputs on the board can be used to measure voltage levels between 0 and 5 volts, which is useful for reading values from sensors and other analog

devices. The board is powered by a voltage regulator that ensures a stable 5 volts, which makes it easy to use with a wide range of power sources.

4.3.1.2 Applications In Object Avoidance Robots

In an object avoidance robot, the Arduino UNO acts as the central processing unit that controls all the robot's functions, including obstacle detection and avoidance. It receives inputs from sensors and processes the data to determine the location and distance of obstacles. Based on this information, the Arduino UNO sends commands to the motor driver to regulate the robot's movements and avoid obstacles.

Apart from its central role, the Arduino UNO comes with numerous digital and analog input/output pins that make it easier to interface with various sensors, motors, and other electronic components, making it versatile for use in various robot configurations.

Moreover, the Arduino Integrated Development Environment (IDE) provides a user-friendly software platform for programming the Arduino UNO. The IDE includes an extensive library of pre-written code and functions that simplify programming tasks, even for beginners.

Therefore, the Arduino UNO is a critical component in an object avoidance robot due to its control and processing capabilities, as well as its versatility and user-friendliness.

In conclusion, the Arduino Uno is a powerful and versatile microcontroller board that is popular in the field of robotics and automation. Its features and functionalities make it a great choice for beginners and advanced users alike. The board's ease of use and accessibility have made it a popular choice for a variety of electronic projects, including object avoidance robots. With its ability to control motors and servos and read values from sensors, the Arduino Uno provides the essential tools for building a successful object avoidance robot.

4.3.2 Motor Driver (L298N)

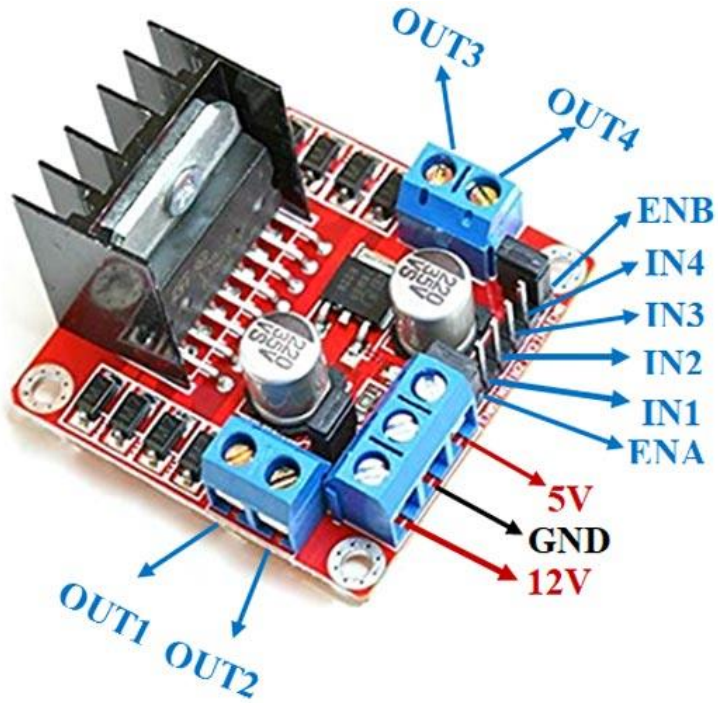


Fig 4.3 - Motor Driver (L298N)

A motor driver is an electronic device that controls the speed and direction of a motor. The L298N is a popular motor driver IC that can be used to control two DC motors or a single stepper motor. It is widely used in robotics and automation projects. The L298 Motor Driver is a popular dual H-bridge motor driver IC that can control the direction and speed of two DC motors independently. It is commonly used in robotics projects, including object avoidance robots, to control the movement of the robot.

4.3.2.1 Features and Specifications:

The IC has four input pins, which are used to control the motors. Each input pin can be set to high or low, which determines the direction of the motor. The IC also has two enable pins, which can be used to control the speed of the motor. The L298N motor driver has a maximum current rating of 2A per channel, which makes it suitable for driving a wide range of motors.

The L298 Motor Driver plays a crucial role in an object avoidance robot by receiving commands from the microcontroller, usually an Arduino, and driving the motors accordingly. It acts as a bridge between the microcontroller and the motors, allowing the microcontroller to control the speed and direction of the motors.

The L298 Motor Driver has two sets of input pins for each motor, IN1 and IN2 for Motor A, and IN3 and IN4 for Motor B. These pins receive digital signals from the microcontroller that determine the direction and speed of the motor. For example, a high signal on IN1 and a low signal on IN2 would make Motor A spin in one direction, while a high signal on IN2 and a low signal on IN1 would make it spin in the opposite direction.

The L298 Motor Driver also has two sets of output pins for each motor, OUT1 and OUT2 for Motor A, and OUT3 and OUT4 for Motor B. These pins connect to the positive and negative terminals of the motors.

4.3.2.2 Wiring

The L298N motor driver can be connected to the Arduino Uno using six wires. Two wires are used to connect the power supply to the motor driver, two wires are used to connect the motor to the motor driver, and two wires are used to connect the motor driver to the Arduino Uno. The wiring of the L298N motor driver is relatively simple and can be easily understood by referring to the datasheet.

4.3.2.3 Applications in Robotics

The L298N motor driver is widely used in robotics projects, particularly in projects that require the control of two DC motors. The motor driver is commonly used in line-following robots, obstacle avoidance robots, and autonomous vehicles. The motor driver provides the essential tools for controlling the speed and direction of the robot's motors, which is essential for navigating the robot around obstacles and through complex environments.

4.3.2.4 Role in Object Avoidance Robots

The object avoidance robot is a robot that is designed to navigate its environment while avoiding obstacles. The robot uses sensors to detect obstacles in its path and adjusts its movement accordingly. The L298N motor driver plays a crucial role in object avoidance robots, as it is used to control the speed and direction of the robot's motors.

The Arduino Uno is commonly used in conjunction with the L298N motor driver to build object avoidance robots. The Arduino Uno provides a platform for programming the logic of the robot and controlling the sensors.

In summary, the L298 Motor Driver is an essential component of an object avoidance robot as it enables the microcontroller to control the direction and speed of the motors. It is easy to use and has several input and output pins that make it versatile for various robotic applications.

In conclusion, the L298N motor driver is a powerful and versatile device that is popular in the field of robotics and automation. Its features and functionalities make it a great choice for controlling the speed and direction of DC motors. The L298N motor driver is commonly used in conjunction with the Arduino Uno to build object avoidance robots. With its ability to control the speed and direction of the motors and read values from sensors, the L298N motor driver provides the essential tools for building a successful object avoidance robot.

4.3.3 Dual Shaft DC Motor

Fig 4.4 Dual Shaft DC Motor

A DC motor is a type of electric motor that converts electrical energy into mechanical energy. It is commonly used in robotics and other automation applications where precise control over motor movement is required. In an object avoidance robot, DC motors play a critical role in controlling the movement of the robot and helping it avoid obstacles.

4.3.3.1 Features And Functionalities

The 2 Shaft DC motor is a small and lightweight motor that is designed to rotate in either direction when a voltage is applied to it. The motor has two shafts, one on each end, that can be used to attach gears, wheels, or other components. The motor's speed and direction can be controlled by adjusting the voltage and polarity of the applied current. The motor's rotation speed is directly proportional to the voltage applied to it, while the direction of rotation is determined by the polarity of the applied current.

4.3.3.2 Role of DC Motor in Object Avoidance Robot

In an object avoidance robot, DC motors are typically connected to a motor driver, such as the L298 motor driver, which allows the microcontroller to control the direction and speed of the motor. The DC motor has two terminals: a positive terminal and a negative

terminal. When a voltage is applied across these terminals, the motor's shaft rotates in one direction.

To control the speed and direction of the DC motor, the motor driver receives signals from the microcontroller and applies them to the motor's terminals. The motor driver typically has four output pins: OUT1, OUT2, OUT3, and OUT4. OUT1 and OUT2 are used to control the direction and speed of motor A, while OUT3 and OUT4 are used to control the direction and speed of motor B. By modulating the voltage across these output pins, the motor driver can control the speed and direction of the motor.

To connect the DC motor to the motor driver, the following connections are required:

- **Positive terminal:** Connect to the OUT1 or OUT3 pin of the motor driver.
- **Negative terminal:** Connect to the OUT2 or OUT4 pin of the motor driver.

To power the DC motor, the following connections are required:

- **Positive terminal:** Connect to the VCC pin of the motor driver.
- **Negative terminal:** Connect to the GND pin of the motor driver.

In summary, the DC motor is a crucial component in an object avoidance robot as it enables the robot to move and avoid obstacles. By connecting the DC motor to a motor driver, the microcontroller can control the speed and direction of the motor, allowing the robot to move in a controlled and precise manner.

In conclusion, the 2 Shaft DC motor is a versatile and reliable device that is widely used in the field of robotics and automation. Its features and functionalities make it a great choice for controlling the movement of a robot. The 2 Shaft DC motor is commonly used in conjunction with the Arduino Uno to build object avoidance robots. With its ability to control the movement of the robot and read values from sensors, the 2 Shaft DC motor provides the essential tools for building a successful object avoidance robot.

4.4 Algorithm Used for Object Avoidance

Object avoidance is an important aspect of robotics that involves detecting and avoiding obstacles in the environment. This can be achieved through various algorithms and techniques, including recursion and backtracking. In this article, we will discuss how recursion and backtracking can be used to compute object avoidance for a robot in detail.

4.4.1 Recursion In Object Avoidance

Recursion is a technique that involves calling a function within itself repeatedly until a certain condition is met. In object avoidance, recursion can be used to break down the path planning problem into smaller sub-problems. The robot can use a recursive function to check if the current path is free of obstacles. If the path is blocked, the function can call itself with a new starting point until it finds a clear path. This process can be repeated until the robot finds a path that leads to the goal without any obstacles in the way.

The main advantage of using recursion in object avoidance is that it allows the robot to explore different paths and backtrack when it reaches an obstacle. This can be useful in complex environments where the robot needs to make quick decisions to avoid obstacles. Recursion can also handle dynamic environments where obstacles may move or change position.

To implement recursion in object avoidance, the robot needs to have a map of the environment with obstacles and the goal location. The robot can then use a recursive function to explore the different paths from the starting point to the goal. The function can use a depth-first or breadth-first search algorithm to explore the paths and backtrack when it reaches an obstacle. The robot can also maintain a list of visited nodes to avoid revisiting them.

One of the challenges of using recursion in object avoidance is that it can be computationally expensive, especially for large environments with many obstacles. The robot may also get stuck in a local minimum or maximum, where it is unable to find a clear path to the goal. To mitigate these challenges, the robot can use heuristics or machine learning algorithms to guide its path planning process.

4.4.2 Backtracking In Object Avoidance

Backtracking is a technique that involves exploring all possible solutions to a problem by trying different paths until a solution is found. In object avoidance, backtracking can be used to explore different paths and backtrack when it reaches an obstacle. The robot can use a backtracking algorithm to maintain a list of visited nodes and explore all possible paths from each node until it reaches the goal or finds a path that avoids obstacles.

The main advantage of using backtracking in object avoidance is that it allows the robot to explore all possible paths and find the optimal path to the goal. Backtracking can also handle complex environments with multiple obstacles and dynamic obstacles that move or change position.

To implement backtracking in object avoidance, the robot needs to have a map of the environment with obstacles and the goal location. The robot can then use a backtracking algorithm to explore the different paths from the starting point to the goal. The algorithm can maintain a list of visited nodes and explore all possible paths from each node until it reaches the goal or finds a path that avoids obstacles. The robot can also use heuristics or machine learning algorithms to guide its path planning process.

One of the challenges of using backtracking in object avoidance is that it can be computationally expensive, especially for large environments with many obstacles. The robot may also get stuck in a local minimum or maximum, where it is unable to find a clear path to the goal. To mitigate these challenges, the robot can use pruning techniques or machine learning algorithms to reduce the search space.

4.4.3 Recursive Backtracking in Object Avoidance

Recursive backtracking is a combination of recursion and backtracking that can be used in object avoidance. In recursive backtracking, the robot uses a recursive function to explore different paths and backtrack when it reaches an obstacle. The robot can also maintain a list of visited nodes and explore all possible paths from each node until it reaches the goal or finds a path that avoids obstacles.

The main advantage of using recursive backtracking in object avoidance is that it combines the benefits of recursion and backtracking. It allows the robot to explore different paths and backtrack when it reaches an obstacle while also exploring all

possible solutions to find the optimal path to the goal. Recursive backtracking can handle complex environments with multiple obstacles and dynamic obstacles that move or change position.

The algorithm continues until it finds a path that leads to the goal without any obstacles in the way or until all possible paths have been explored. The robot can use the path found by the algorithm to navigate to the goal while avoiding obstacles.

One of the challenges of using recursive backtracking in object avoidance is that it can be computationally expensive, especially for large environments with many obstacles. The robot may also get stuck in a local minimum or maximum, where it is unable to find a clear path to the goal. To mitigate these challenges, the robot can use pruning techniques or machine learning algorithms to reduce the search space.

In conclusion, recursion and backtracking are powerful techniques that can be used in object avoidance to navigate robots through complex environments. By using a recursive backtracking algorithm, the robot can explore different paths and backtrack when it reaches an obstacle while also exploring all possible solutions to find the optimal path to the goal. However, the use of recursion and backtracking in object avoidance can be computationally expensive and require advanced algorithms to reduce the search space and optimize the path planning process.

CHAPTER 5: METHODOLOGY AND WORKING

This project implements a recursion and backtracking algorithm to generate the shortest path in a grid with obstacles placed at certain coordinates. A grid is defined which is composed of cells that can have two possible combinations: 0 and 1. A cell with a value of 0 indicates an invalid cell that the robot cannot pass through, while a cell with a value of 1 indicates a valid cell that the robot can traverse.

The process requires the information related to the static obstacle environment which is the input of the problem and the algorithm determines the most optimal path through which the robot should move in an early stage even before the robot starts moving. In real-life example, an obstacle environment with grids of 0s and 1s is a representation of an image or video frame that needs to be observed. In this case, the 1s can represent areas of the image where there is no object present or where the object is of no interest, while the 0s can represent the object or objects that need to be detected and analyzed and from which the robot cannot pass through.

Additionally, obstacle environments with grids of 0s and 1s can be used in games and simulations, where they can represent various obstacles and barriers that players or agents need to navigate around. These environments can also be used in planning and optimization problems, such as determining the optimal path for a vehicle to follow through a complex terrain.

The code takes a pre-defined path in the form of a 2D matrix consisting of 0s and 1s. By implementing recursion and backtracking, the algorithm computes all possible paths within the grid. Among all the valid obtained paths, the algorithm selects the one with the shortest length. The result is represented as a string consisting of direction commands.

For example, if the result is "DRLUD," it means that the robot should first move downward, then to the right, followed by a left movement, followed by an upward movement and finally a downward movement again.

By using recursion, the algorithm explores different possible paths by making decisions at each step and backtracking whenever it encounters an obstacle or reaches a dead end. This process continues until all possible paths have been explored, allowing the algorithm to find the shortest path from the starting point to the destination in the grid.

Flowchart For Basic Overview of Project

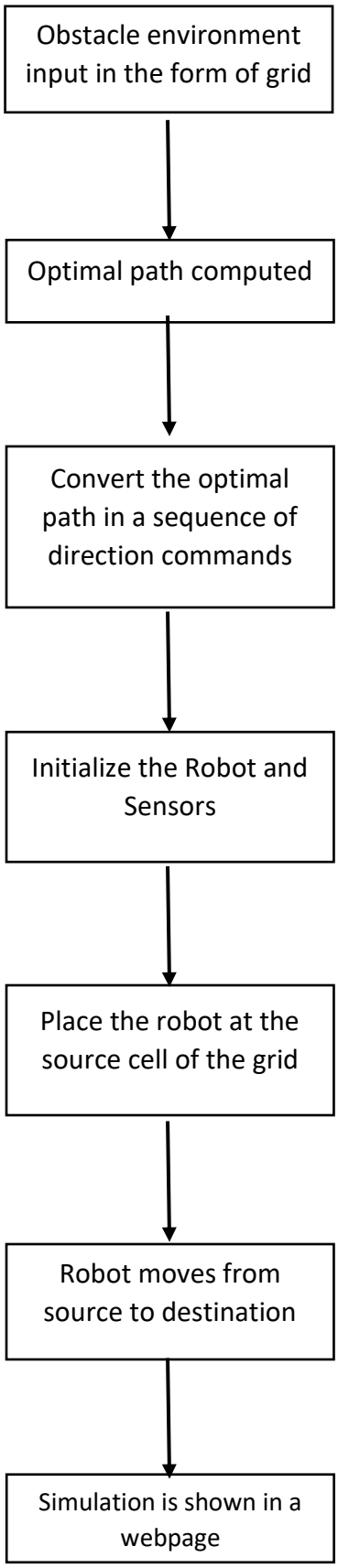


Fig 5.1 Flowchart for Basic Overview of Project

The basic block diagram of hardware for the implementation of the project is as shown:

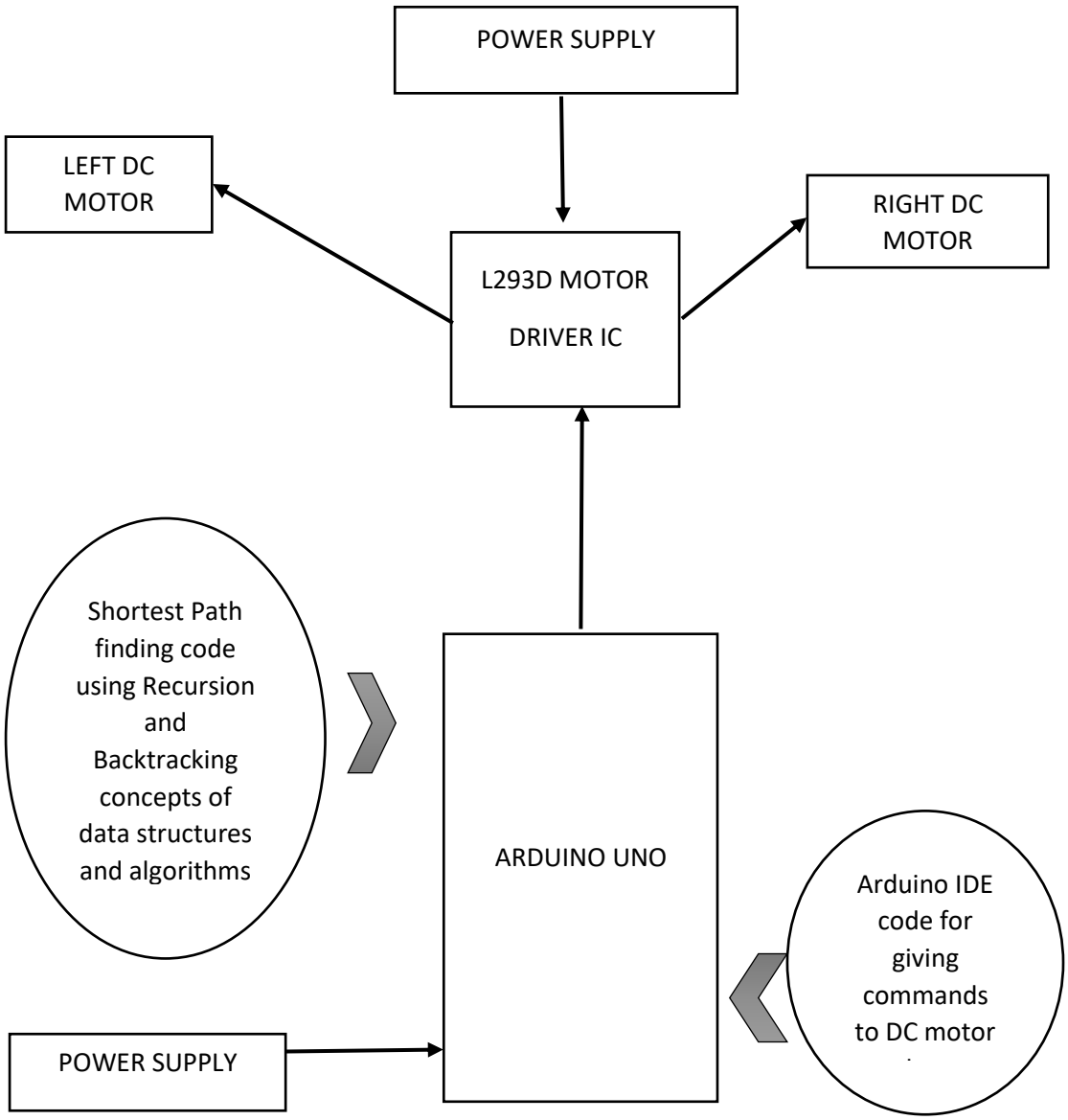


Fig 5.2 Block Diagram for Hardware Implementation

Power is supplied from four 9 volts battery pack to Arduino UNO and L298 motor driver IC which then gives command to right and left DC motor. Shortest path finding code using recursion and backtracking is written and fed into Arduino UNO microcontroller which gives command to motor driver.

The basic circuit diagram of hardware for the implementation of the project is as shown:

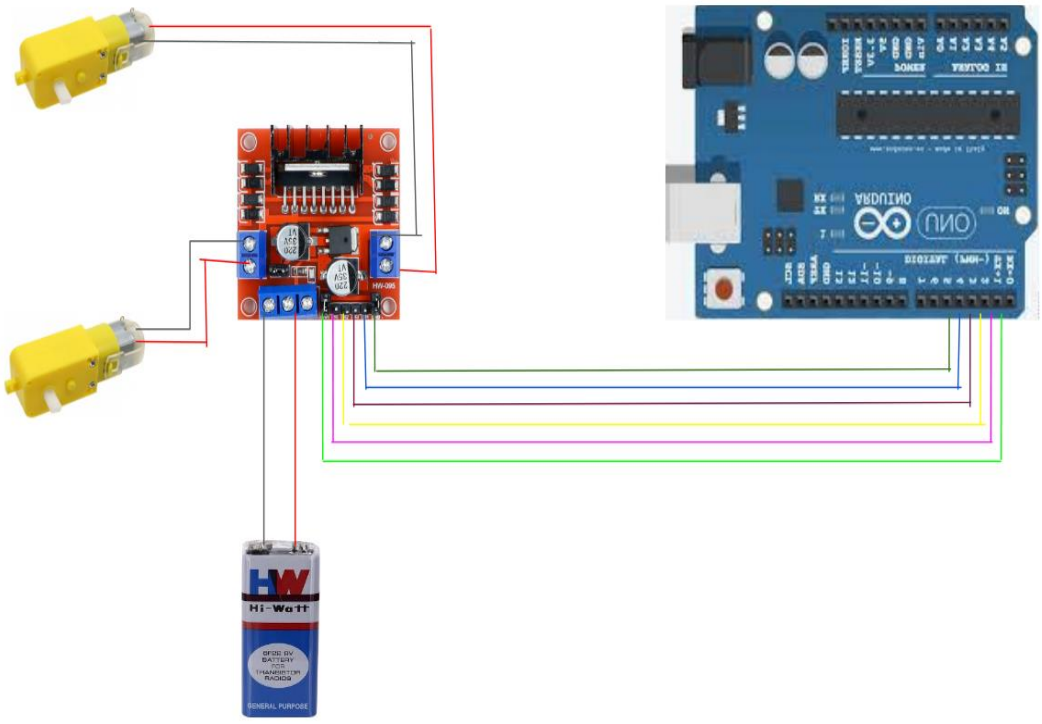


Fig 5.3 Circuit diagram for hardware implementation

Power supply is connected to L298 motor driver in VCC and GND pin. Each DC motor's two wires is connected to L298 motor driver which controls clockwise and anticlockwise

rotation of wheels. Enable and Input pins of motor driver are connected respectively to Arduino UNO digital pins.

We have divided the entire space into m*n matrix with each cell of equal standard size where each block or cell is either empty or has obstacle in it. We need to find the shortest path between a given source cell and destination cell

avoiding the collision with any obstacle. An example of environment is shown below, where shaded block represents, it has obstacle in it.

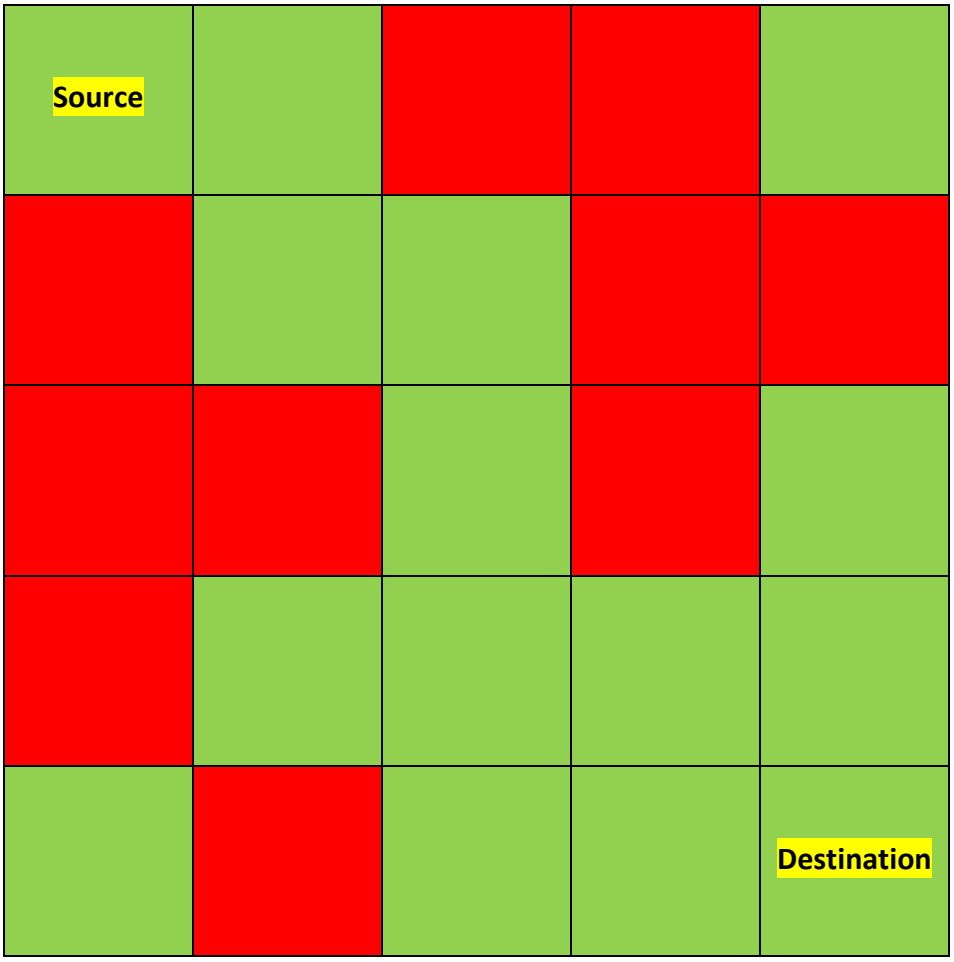


Fig 5.4 Description of obstacle environment

To solve this problem, we have used Recursion and Backtracking techniques of data structures and algorithms. The overall environment of place is pre fed into the code and then using recursion it iterates over all the possible paths. Backtracking is used in order to avoid checking the same path again. After analyzing all the possible paths, code finds out the shortest or most optimal path. Once the shortest path is found it gives the direction commands that robot need to follow in order to reach destination from source.

This algorithm can be defined in detail as follows:

1. Start from the source block or cell in the pre fed environment matrix and explore all possible paths using recursion.
2. Check if destination block is reached in the environment and backtrack avoiding the path already checked if destination is not reached.
3. Keep track of visited cells using array or vector.

4. Compare all the paths which successfully made it till destination and find out the shortest or optimal path.
5. Generate direction commands as right, left, up and down for robot to follow this shortest path and reach destination successfully avoiding all the collision.

Now after getting the direction commands from code to reach destination from source through shortest path avoiding all the collisions, we need to give these commands to our robot's wheel's DC motor for this we write move function and other required code in Arduino uno microcontroller, which finally makes our robot to reach destination.

5.1 Algorithm

The aim of the algorithm is to perform two main tasks. First to find the shortest path and second, to move the robot accordingly from source to destination.

5.1.1 Determination Of Shortest Path

It consists of getallpath() function which takes the following arguments –

1. Grid in the form of a matrix (Obstacle Environment)
2. Variable n which is the ratio of size of matrix to size of 1st row of matrix
3. Current row index where the robot is present
4. Current column index where the robot is present
5. A temporary string variable for storing current string
6. A string variable for storing final result
7. An int variable for storing minimum length

This function getallpath() is a recursive function which accepts the above arguments and it first checks if the current position is valid (not out of bounds or a blocked cell) and returns if not. If the current position is the bottom right corner of the matrix, the function checks if the current path is shorter than the current minimum path length and updates the final String and minimum value (length) accordingly.

The function then recursively calls itself on the neighbouring cells (up, left, bottom, right), adding the corresponding letter to the current string each time. It also marks the current cell as blocked (so that same path is not computed again) before making the recursive calls, and unmarks it after the recursive calls are done.

Also it is self-explanatory that the function assumes that final string and minimum value are initially set to a higher value (e.g., 100000 in this case) before the first call which ensures that any path found on the first call will be shorter than the initial value and it is stored in the array.

Overall, this function uses backtracking algorithm to explore all possible paths in the matrix and keep track of the shortest paths found according to length.

Out of all the paths found, all the paths with minimum length are stored in an array.

There are two conditions from which the shortest path string is obtained –

1. String length – Out of all the possible paths, those with shortest length are chosen

2. Rotations – Out of all shortest paths with same length, those with less rotations are chosen

If two paths have same length and same rotations, then any of them can be considered.

5.1.2 Implementation of shortest path on Robot

It consists of pathImplementation() function which takes a string as an argument, which is the shortest path for the given environment. In this function we iterate over each character in the string and based upon which, we perform one mandatory action of moving the robot forward and rotating it if required. For moving the robot we have mover() function and for rotating we have anticlockRotate() function which rotates the function in the anticlockwise direction and clockRotate() function which rotates the robot in the clockwise direction.

5.1.2.1 Arduino built-in functions

For working of our functions we use built in functions of arduino

5.1.3.1 digitalWrite()

The **digitalWrite()** function is a built-in function in the Arduino programming language that allows the user to set the output value of a specific digital pin on the microcontroller. This function is commonly used in conjunction with electronic components such as LEDs, motors, relays, and other devices that require digital signals to operate.

The function is used to send a digital signal to a specific pin on the Arduino board. The user specifies the pin number and the value they want to send to that pin. The digital value can be either HIGH or LOW. When a digital pin is set to HIGH, the voltage on the pin is typically 5 volts. When a digital pin is set to LOW, the voltage on the pin is typically 0 volts.

The syntax of the digitalWrite() function is straightforward and easy to use. The function takes two arguments: the first is the pin number, and the second is the value to be set on that pin.

In this syntax, "pin" is the digital pin number that the user wants to send the digital signal to, and "value" is the value to be sent to the pin, which can be either HIGH or LOW.

The digitalWrite() function plays a crucial role in the Arduino programming language because it allows the user to control the state of digital pins on the microcontroller board. This, in turn, allows the user to control electronic

components such as LEDs, motors, and relays, and even interface with other electronic devices.

5.1.2.1 digitalRead()

The **digitalRead()** function in Arduino is used to read the state of a digital pin on the microcontroller board. It takes one argument, which is the pin number to be read, and returns a value that indicates whether the pin is in a HIGH or LOW state. This function is important for obtaining information from electronic components such as buttons, switches, and sensors, which can be used to control the behaviour of the microcontroller or send data to external devices.

5.1.2.2 analogWrite()

The analogWrite() function in Arduino allows you to generate a PWM (Pulse Width Modulation) signal on a specific pin. PWM is a technique where a digital signal is rapidly switched between HIGH and LOW states at a certain frequency and duty cycle, which effectively simulates an analog output.

To use the analogWrite() function, you need to provide two parameters:

pin: This is the number of the pin that supports PWM. These pins are typically marked with a "~" symbol on Arduino boards.

value: This parameter represents the duty cycle or intensity of the PWM signal. It should be an integer between 0 and 255, where 0 means the signal is always off (0% duty cycle), and 255 means the signal is always on (100% duty cycle).

5.1.2.3 serial.print()

The Serial.print() function is used in Arduino to send data from the microcontroller to the serial port, which can be connected to a computer or another device. It allows you to send text, numbers, or other data types for debugging purposes, data logging, or communication with external devices.

5.1.2.4 millis()

The millis() function in Arduino is a useful function for timing and creating delays without blocking the execution of other code. It returns the number of milliseconds that have passed since the Arduino board started running its current program.

The millis() function is often used in combination with variables and conditional statements to create non-blocking delays or time-based events. It allows you to

perform actions at specific intervals without using the delay() function, which would halt the program execution.

5.1.3 Moving and Rotation functions

5.1.3.1 Moving function – mover()

In this we set pin in1 and in2 high for set duration of time depending on the dimension of the block in the environment. After the robot moves by the distance of block, we set the both analog signals (enA, enB) to zero.

Rotation functions – clockRotate() and anticlockRotate()

ClockRotate() function

To rotate the robot, we set robot on a fixed position and rotate one of the wheels in forward direction depending upon the requirement for a set duration of time and stop wheel after it rotates by 90 degrees.

In this we set digital input 4 (in4) high which rotates the right wheel (frame of reference behind the robot) and set analog input of right wheel high for a set duration of time, which rotates the robot in clockwise direction for 90 degrees after which we set both analog inputs to zero, so that no movement takes place.

AnticlockRotate()

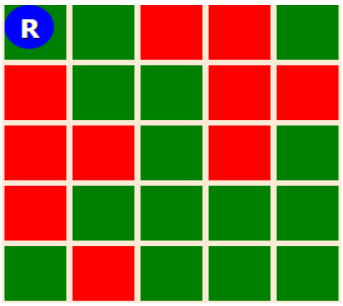
In this we set digital input 1 (in1) high which rotates the left wheel (frame of reference behind the robot) and set analog input of left wheel high for a set duration of time, which rotates the robot in anti-clockwise direction for 90 degrees after which we set both analog inputs to zero, so that no movement takes place.

CHAPTER – 6: RESULTS AND SIMULATION

- The project involved designing a robot from scratch using various electronic components such as Arduino-UNO micro-controller, Ultra-sonic sensor, Bridge motor driver L298, DC Motor, Servo motor, Omni-direction wheel, jumper wires, and 9V-batteries. The team assembled and programmed the robot to perform specific tasks, such as navigating through obstacles.
- During the project, the team also delved into various obstacle avoidance algorithms and concepts such as recursion, backtracking, depth-first search, and Dijkstra Algorithm. This helped them to understand the different ways the robot can navigate through obstacles efficiently.
- To optimize the robot's performance, the team precomputed the shortest path before deployment. This approach allowed the robot to avoid the conventional approach of finding a path at each obstacle during runtime, thereby reducing computational time and enhancing the robot's efficiency.
- In addition, the team designed a user interface using HTML, CSS, and JavaScript, which provided a visual representation of the optimal path computed by the robot. This allowed the user to interact with the robot and monitor its performance.
- Finally, the team tested the robot in various environments, with obstacle positions changed each time. The results were computed and analysed to evaluate the robot's performance and identify areas for improvement.

Consider the following static environment with obstacles in a grid as follow –

[1, 1, 0, 0, 1],
[0, 1, 1, 0, 0],
[0, 0, 1, 0, 1], ==>
[0, 1, 1, 1, 1],
[1, 0, 1, 1, 1]



1 or green represents a valid cell (Robot can pass through this cell)

0 or red represents an invalid cell (Robot cannot pass through this grid)

Algorithm computes all the possible valid paths along with directional commands. Following table shows all the possible paths along with the total time robot will take to reach from source to destination.

Table 6.1 Valid Paths Specifications

S No.	Path	Length	Rotations
1	RDRDDRRD	8	5
2	RDRDDRDR	8	6
3	RDRDDDRURD	10	7
4	RDRDDDRR	8	4

From the above table, we determine the path according to two conditions –

1. Choosing the path with minimum length
2. Choosing the path with minimum rotations

If multiple paths have same minimum length and minimum number of rotations then any of them can be chosen.

After the shortest path string has been determined then it is fed to the Arduino which makes the robot move along the directional commands along the shortest path. Robot's movements can be more efficient than testing each possible path. By using the shortest path, the robot can navigate the matrix with the fewest number of steps and avoid unnecessary movements that would waste time and energy.

The project also provided a visual representation of the shortest path algorithm's working through a website. The environment is depicted as a grid or 2D matrix, where each cell is coloured either red or green.

- Red cells represent invalid cells that the robot cannot pass through.
- Green cells represent valid cells that the robot can traverse.

Website is implemented with HTML, CSS, and JavaScript:

- HTML is used to structure the webpage and define the grid layout.
- CSS is used to style the grid and assign colours to the cells (red and green).

- JavaScript is used to implement the logic of the algorithm, compute the shortest path, and animate the robot's movement.

Visualization of the Robot's Movement in the grid:

After computing the shortest path in the form of a string with direction commands (e.g., "DRLU" for down, right, left, upward), the website animates the robot's movement based on these commands.

The animation can be achieved by manipulating the CSS properties of the robot element or by dynamically updating its position on the grid.

For example, if the shortest path is " RDRDDDRR ", the website can visually show the robot moving right , then down, then right, then down, then down, then down, then right and then right.

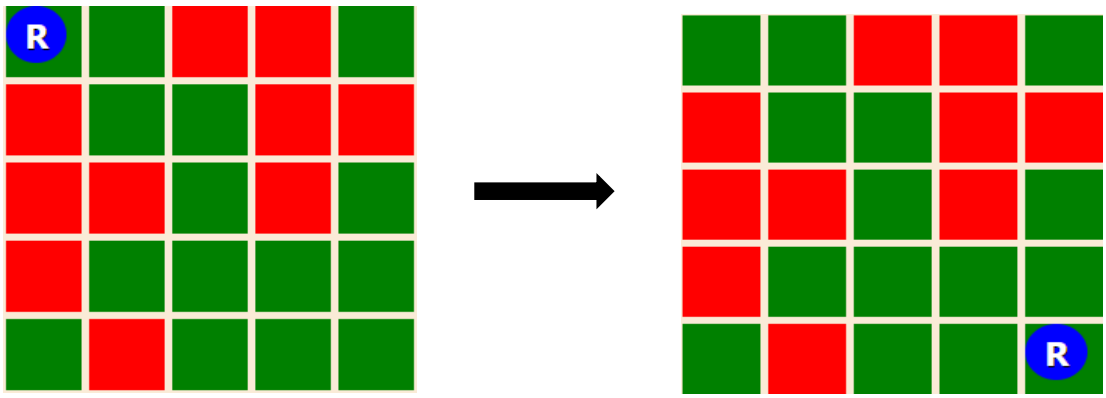


Fig 6.1 -Visual Description of Robot and obstacle environment through website [3]

By combining the visual representation of the grid environment, the colours assigned to cells, and the animation of the robot's movement, the website provides a high-level overview of how the algorithm works and how the robot navigates through the grid to find the shortest path.

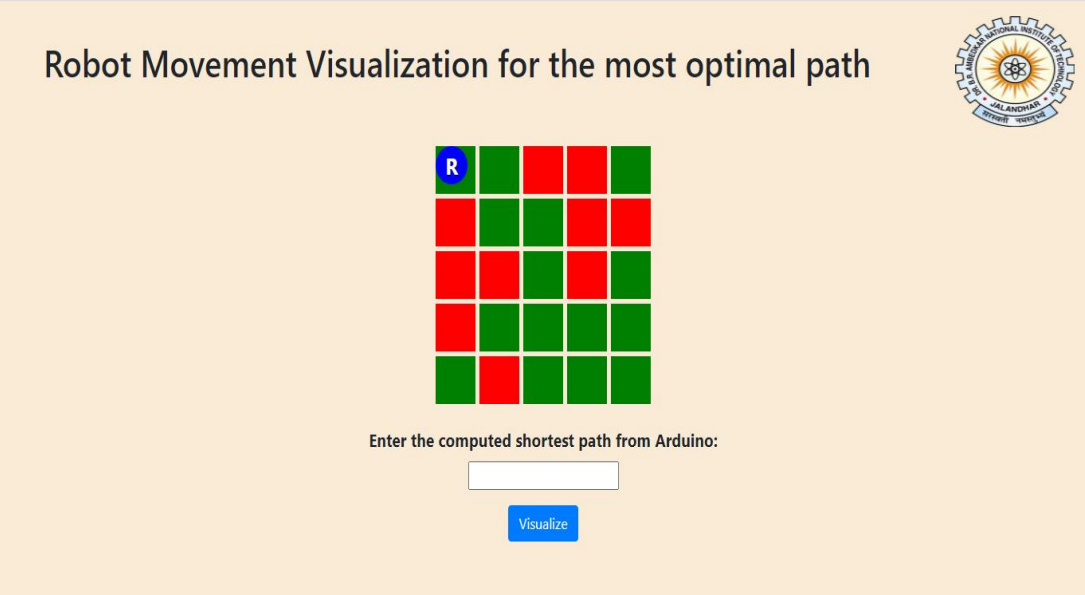
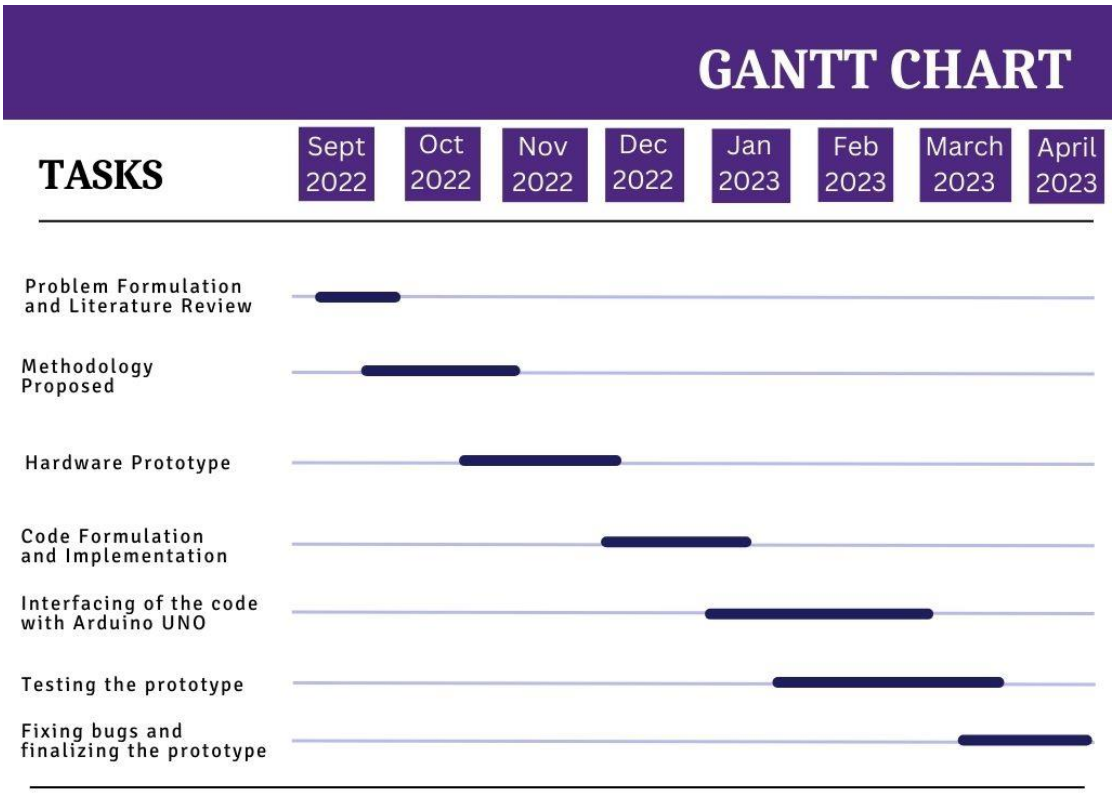


Fig 6.2 Visual description for website implemented for simulating robot's movement

GANTT CHART


GANTT CHART
TASKS
Sept 2022
Oct 2022
Nov 2022
Dec 2022
Jan 2023
Feb 2023
March 2023
April 2023
Problem Formulation and Literature Review
Methodology Proposed
Hardware Prototype
Code Formulation and Implementation
Interfacing of the code with Arduino UNO
Testing the prototype
Fixing bugs and finalizing the prototype

REFERENCES

[1] Sharma, Rahul. "Design and implementation of path planning algorithm for wheeled mobile robot in a known dynamic environment." IJRET: International Journal of Research Engineering and Technology 2.06 (2013): 967-970.

[2] Chaari, Imen, et al. "Design and performance analysis of global path planning techniques for autonomous mobile robots in grid environments." International Journal of Advanced Robotic Systems 14.2 (2017): 1729881416663663.

[3] Ismail, A. T., Alaa Sheta, and Mohammed Al-Weshah. "A mobile robot path planning using genetic algorithm in static environment." *Journal of Computer Science* 4.4 (2008): 341-344.

[4] https://iopscience.iop.org/article/10.1088/1757-899X/152/1/012064/pdf

[5] Ajeil, Fatin Hassan, et al. "Grid-based mobile robot path planning using aging-based ant colony optimization algorithm in static and dynamic environments." *Sensors* 20.7 (2020): 1880.

[6] https://srituhobby.com/how-to-make-a-multi-function-arduino-robot/

[7] GitHub link : https://harsh99786.github.io/ShortestPathFinder.github.io/

[8] Hu, X., Chen, L., Tang, B., Cao, D., & He, H. (2018). Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles. Mechanical Systems and Signal Processing, 100, 482-500. https://doi.org/10.1016/j.ymssp.2017.07.019

[9] https://ieeexplore.ieee.org/abstract/document/4339335

[10]https://ieeexplore.ieee.org/abstract/document/6761521/citations?tabFilter=papers#citations

ANNEXURE

The complete code for the model to find path….

Moving function – mover()

```
void mover(){
   int starttime = millis();
   int endtime = starttime;
   int loopcount = 0;
   while ((endtime - starttime) <= 1500) // do this loop for up to 1500mS
   {
     digitalWrite(in1,HIGH);
     digitalWrite(in2,LOW);
     digitalWrite(in3,LOW);
     digitalWrite(in4,HIGH);
     analogWrite(enA,130);   //right
     analogWrite(enB,255);  //left
     loopcount = loopcount+1;
     endtime = millis();
   }
     analogWrite(enA,0);
     analogWrite(enB,0);
}
```

Rotation functions – clockRotate() and anticlockRotate()

```
void clockRotate(){
   int starttime = millis();
   int endtime = starttime;
   while ((endtime - starttime) <= 500) // do this loop for up to 500mS
   {
     digitalWrite(in1,LOW);
     digitalWrite(in2,LOW);
     digitalWrite(in3,LOW);
     digitalWrite(in4,HIGH);
```

```
    analogWrite(enA,0);
    analogWrite(enB,500);
    endtime = millis();
   }
    analogWrite(enA,0);
    analogWrite(enB,0);
}

void anticlockRotate(){
   int starttime = millis();
   int endtime = starttime;
   while ((endtime - starttime) <= 500) // do this loop for up to 500mS
   {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
    analogWrite(enA,500);
    analogWrite(enB,0);
    endtime = millis();
   }
    analogWrite(enA,0);
    analogWrite(enB,0);
}
```

Path Implementation function – pathImplementation()

```
void pathImplementation(String path) {
  int n = path.length();
  for (int i = 0; i < n; i++) {
    char current = path.charAt(i);
    if (i == 0) {
      if (current == 'L') {
        anticlockRotate();
      } else if (current == 'R') {
```

```
    clockRotate();
   } else if (current == 'U') {
    anticlockRotate();
    anticlockRotate();
   }
  }
  else if (path.charAt(i) != path.charAt(i - 1)) {
   char previous = path.charAt(i - 1);
   if (current == 'U') {
    if (previous == 'L') {
     anticlockRotate();
    } else {
     clockRotate();
    }
   }
   else if (current == 'D') {
    if (previous == 'L') {
     clockRotate();
    } else {
     anticlockRotate();
    }
   }
   else if (current == 'R') {
    Serial.print(i);
    if (previous == 'U') {
     anticlockRotate();
    } else {
     clockRotate();
    }
   } else if (current == 'L') {
    {
     if (previous == 'U') {
      clockRotate();
     } else {
```

```
            anticlockRotate();
          }
        }
      }
    }
    mover();
  }
}
```