ESP32 Based Line Following Robot

Chethan Srinivasareddy[1] **Sai Bharadhwaj Matha**[2] **Pettugani Venkata Haripriya**[3] **Kolurkavil Rajan Rahul**[4]
Master of Engineering: Artificial Intelligence for Smart Sensors and Actuators, Technische Hochschule Deggendorf, Cham, Germany
chethan.srinivasareddy@stud.th-deg.de[1]
sai.matha@stud.th-deg.de[2]
venkata.pettugani@stud.th-deg.de[3]
rahul.kolurkavil-rajan@stud.th-deg.de[4]

***Abstract*—Line-following robots are one of the simplest and most robust autonomous systems. These robots should detect the predefined path and be able to navigate it with minimal or no human intervention. Typically, the path is designed using black or yellow lines on the surface. An infrared (IR) sensor keeps the robot on course with position-corrective feedback. The previous work presents the design and implementation of an Open-Loop control strategy that causes wobbly movements with undesirable speed. This paper focuses on the improvements in traits of previous robots. The combination of the Closed Loop control feedback method for error reductions can lead to smoother movement and some more sensors for redundancy as well as for object detection. The ESP32 will coordinate between 5 IR sensors and 2 motor controllers to adjust its position with corrective feedback from an IR array of sensors.**

***Index Terms*—PID,Closed Loop,Calibration,enums,state machines,GPIO, Active Low.**

I. INTRODUCTION

THe Industries in this era are becoming increasingly automated for various applications. One such application where these robots can and are making tremendous progress in terms of efficiency and cost-cutting is In-House material handling in the Industries and Logistics domain, delivering the goods [1], automatic watering the plants [2], restaurants [3], agriculture sector [4], [5], fire safety [6], Library inventory management system (LIMS) [7] etc. but these are popular robots commonly used for sorting and moving the goods in warehouses and industrial logistics [8]. In recent events around the world, we have observed how the slightest delay in material transportation across countries can affect common people and countries. Hence, any improvements we can make in this domain will have a larger impact on the big picture. These line-following robots operate in a certain unique physical environment that follows a predefined path in black color with a white or light background. When repeated material handling is carried out on a fixed path, these can be used most effectively. These eliminate human errors and fatigue, as they do not need rest and can perform operations 24x7 with enhanced efficiency and reliability. These are destined to achieve more productivity in a given time at less cost. Semi-automatic or fully-automatic industrial applications can be done with these robots.

Robots have some fundamental building blocks. That includes software, hardware, a microcontroller, a selection of sensors, the control system, and the interface between all these components. The robot must meet certain criteria before being considered fit for production or commercial use. Robots should precisely follow the path with greater control over the speed, i.e., without compromising accuracy over speed and vis-a-vis. Lower power consumption, i.e., power consumption design architecture, should be as efficient as possible. Finally, there is a cost factor and an acceptable reliability level.

It is a three-wheeled robot, with two motor-controlled wheels and one behind a castor wheel for balance and weight distribution. This works in a uniquely designed environment where black paths are constructed on white surfaces. Here, the black line is detected by an IR sensor. This robot is equipped with 5 IR sensors for line detection. Regarding the previous reference report where the robot was equipped with two IR sensors and had issues with smooth movement. We use five arrays of IR sensors for line-following accuracy and redundancy. Considering the robot's safety and the environment it works in, it is equipped to detect obstacles and stop. In addition, it is designed to notify the concerned person.

In this paper, obstacle detection is implemented solely with the IR sensor itself. Although there exist other sensors that are specifically designed for efficient object detection, we have made the deliberate choice to proceed with IR sensors while also acknowledging their limitations. The utilization of IR sensors for object detection in this context is purely for the purpose of research and education. Furthermore, this study aims to demonstrate the capability of these sensors in detecting obstacles and to explore the feasibility of utilizing IR sensors for this task.

In Section [III], we have mentioned how the functional and technical requirements of the robot are being met. In this paper, we have organized it as follows: We have defined the robot's functional and technical requirements, which are described in Section [IV]. The methodology and the concept of execution are described in Section [VI], and the functions of code is explained in the section [VII]. Further, challenges we have faced, and in the latter, the testing and results are explained. In addition to that, we have mentioned challenges and bottlenecks with this model in section [XII], and further potential recommendations for further improvements are followed towards the end.

II. BACKGROUND

The history of line following robot automation is described by from the time of evolution in automation techniques. The use of a line follower robot became an identifiable and a unique device in the 1960s due to the efforts of Joseph Engelberger, George Devol who formed the robotic company called "unimation" characterized the latest trend in the automation of the manufacturing process. We could say that automation is closely tied to world economics. The concept of automation refers to the ability of a machine to perform a given sequence of tasks and meet certain specifications automatically. These sorts of autonomous robots are designed to move along predetermined path that is ideally indicated by white line on black background flat surface. By the large, to develop the model it demands multidisciplinary skills, from mechanical design to mathematics and control theory techniques to computer algorithms is essential. In this project the objectives we are achieved mainly attributed to control theory, electrical and efficient algorithms, and sensor calibration. Along with line following, this project addresses the feasibility of using IR sensors for objection detection and implementation methods.

III. LITERATURE REVIEW

Some of the following papers have done similar works with respect to different aspects of line following robots

1) M. Engin and D. Engin, **"Path planning of a line follower robot,"** This paper presents how data received from an array of IR sensors gives a smooth and steady speed even in a partially structured environment. They have implemented a dynamic PID control algorithm with differential motor control [9]. We can observe how various PID values can affect the performance of the robot.
2) M. Pakdaman and M. M. Sanaatiyan, **"Design and Implementation of Line Follower Robot,"** Line-following robots are semi or fully autonomous robots which follow the line drawn on a white or high-contrast surface. These robots can detect the black line using the IR sensors which are placed in front and directed towards the ground. signals received from these are processed and used for direction control [10].
3) M. Abdul Kader, M. Z. Islam, J. Al Rafi, M. Rasedul Islam and F. Sharif Hossain, **"Line Following Autonomous Office Assistant Robot with PID Algorithm,"** This paper introduces us about error compensating and in the efforts to reduce the wobbling movement in the robot by tuning the PID values [11].
4) G. Benet, F. Blanes, J.E. Simó, P. Pérez, **"Using infrared sensors for distance measurement in mobile robots, Robotics and Autonomous Systems"**. This paper examines the utilization of IR sensors for the purpose of obstacle detection. In comparison to Ultrasonic sensors [12], IR sensors exhibit quicker response rates and are considered as novel, cost-effective sensors. The IR sensor measures the intensity of backscattered light emitted by objects and utilizes this information to calculate distance. Similarly, it establishes a threshold level of light intensity to identify and detect objects. Upon surpassing this threshold, the sensor transmits a signal indicating the detection of an object.
5) F. Kaiser, S. Islam, W. Imran, K. H. Khan, and K. M. A. Islam, **"Line follower robot: Fabrication and accuracy measurement by data acquisition,"**. Here we can see something similar to our work where they used an array of 5 IR sensors but with different microcontrollers. Also, they presented the control algorithm for the line following. Additionally, we can also observe the IR sensor's response to different colors [13], thereby gaining insight into its performance as an obstacle detection sensor.

IV. OBJECTIVES

The main aim of the project is to develop the robot to follow the line along with the defined certain objectives which are categorized as functional and specific objectives

Functional Objectives:

- Smooth movement of the robot with no or less wobbling effect while moving with admissible speed.
- Obstacle detection: Detecting the object using 3 IR sensors and halting the robot if it's in the line of robot's direction and position of the object, sends warning indication via LED light.
- Emergency stop button as a safety feature to half the robot under will of operator in any circumstance.

Technical Objectives:

- Line following accuracy with deviation not more than 20mm along the path.
- Maintaining the consistent speed of 10m/s.
- Obstacle detection range of 10cm by back tuning the IR sensors to do so.
- Robot have ground clearance of 3-4cm.
- Safety feature such as Automatic shut off and warning indicator during operation.

V. PURPOSE OF STUDY

Generally, every product before considering for production and for the use case it must meet many criteria with regards to quality, performance, safety and many more. Anything less than this is unreliable. Adopting unreliable products for application can lead to further chaos. In the previous project it has concerns with the performance and accuracy in line following. This is the main objective of this project where performance and accuracy is an area of concern. In addition to this, to analyze the potential of IR sensor for object detection and implement the same.

VI. METHODOLOGY

In reference to the requirements stated in the previous sections, the design of the line following robot and the control algorithm were devised to meet all the stated requirements for the project. This includes the development of a robust control algorithm to ensure a smooth movement of the robot and a one

of its kind obstacle detection algorithms, only by using the IR sensors instead of any other sensors like ultrasonic sensors, LiDARs or RADARs. The primary goal of the proposed algorithm is to minimize the wobbling nature of the existing robot, smooth out its path following capabilities, and add some additional features like obstacle sensing, safety switch, a LED indicator. Prior to going any further, the drawbacks and challenges from the existing line following robot are explained in detail in sub section A, below.

A. Challenges in existing system

The existing line following robot consists of only 3 IR sensors mounted on the front of the robot in a 'seeing-down' manner, where the sensors are just above the black path on a whitish background and the sensors are calibrated for that distance. The control algorithm doesn't have any controller, except for some if-else statements which run the robot's motors at a set speed hard coded in the firmware. When analysed, the control algorithm deemed to be a naïve implementation of the line following robot. Primarily three main challenges or drawbacks are associated with the existing system, namely follows.

1) Wobbling/Zig-Zag movement: As the number of sensors are only 3 and the control logic is being implemented in if-else statements, the robot was prone to zig-zag movement, correcting its path regularly while moving forward. This wobbling movement of the robot has also implicitly put a constraint on the shape of the path, as the path should be limited to obtuse turns and smooth turns. Any sharp or acute turns can result in robot treading away from the path.

2) Speed inconsistencies: The set speed of the motor tends to vary in this case as, during the traversal of the path, the robot tries to correct itself when the path below changes and especially during the turnings, it exhibits a start-stop motion, which is the root cause for the inconsistency in maintaining the set speed.

3) Open loop control: The control strategy in the existing project doesn't have a controller like PID or Fuzzy logic or a state space control. The if-else conditions set the speed of the left and right motors to a pre-defined value which is hard coded in the firmware. This lack of feedback of robot's speed is a major drawback and a hindrance for the smooth movement of the robot.

B. Proposed design

To overcome the above stated problems, the existing robot design was modified to incorporate more IR sensors viz., 5 for line following and 3 sensors for obstacle detection. The hardware changes were limited to only addition of extra sensors, switch button and indicator LED. The idea behind the control strategy for the proposed model is to implement a closed loop control using PID controllers, for smooth and better path following, increase the number of IR sensors and add additional IR sensors for obstacle sensing, perform a sensor calibration before the run, so as to compensate for any voltage drop. In tandem with improving the control strategy the proposed model also aims at the minimal cost budget for this project, which is restricted to purchasing of few IR sensors, switch, and LED's, refer to Appendix IV for the detailed BOM.

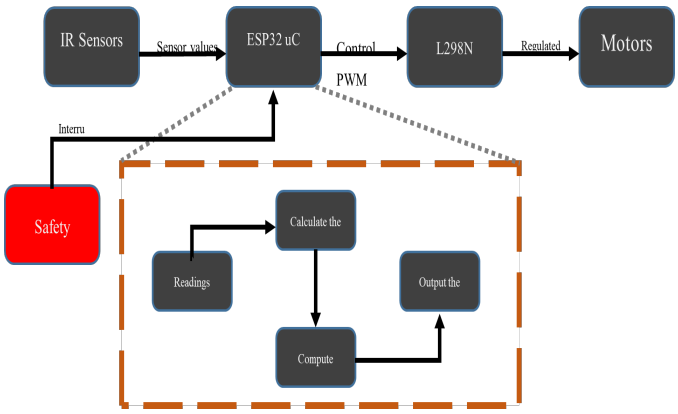

Fig. 1. Flowchart of the proposed design

1) Firmware flowchart: The flow of the firmware is explained in the section. This is a brief overview on how the code works, detailed explanation of code is given in section. The ESP32 microcontroller continuously reads the logic levels of the GPIO pins on which the sensors data signal lines are connected. The readings are then processed by the microcontroller, assigns, and updates the intermediate values like error signals, robot state value, obstacle state value and generates the necessary control signal using the PID controller implemented in the code. This PWM signal is then fed into the motor driver which supplies the necessary control voltage to the motors. The flow of the firmware is explained in the section. This is a brief overview on how the code works, detailed explanation of code is given in "section". The ESP32 microcontroller continuously reads the logic levels of the GPIO pins on which the sensors data signal lines are connected. The readings are then processed by the microcontroller, assigns, and updates the intermediate values like error signals, robot state value, obstacle state value and generates the necessary control signal using the PID controller implemented in the code. This PWM signal is then fed into the motor driver which supplies the necessary control voltage to the motors.

TABLE I
PATH FOLLOW LOGIC TABLE

Sens_L2	Sens_L1	Sens_M	Sens_R1	Sens_R2	Action
0	0	1	0	0	Move straight
0	1	1	0	0	Move to left
0	1	0	0	0	Move to left
1	1	0	0	0	hefty left turn
1	0	0	0	0	Sharp left
0	0	1	1	0	Move to right
0	0	0	1	0	Move to right
0	0	0	1	1	Hefty right turn
0	0	0	0	1	Sharp right
0	0	0	0	0	No Line. rotate
X	X	X	X	X	State unknown

From the Table.1., the state of the robot and the drive state of the robot are updated based on the configuration of the sensor output values. The stop condition is when all the sensors

output a logic HIGH. If in case the sensors' output values are in any unknown configuration other than mentioned in the table, it is treated as an unknown case and the robot moves forward until a valid state is received again. Similarly for the obstacle detection conditions the Table.2 depicts the state and position of the obstacle when placed near the path. The unknown conditions here are treated as a failsafe condition and the robot is brought to a temporary halt.

TABLE II
OBSTACLE SENSING LOGIC TABLE

Sensor_UL	Sens_UM	Sens_UR	Action
0	0	0	No obstacle
1	0	0	Obstacle on extreme left
1	1	0	Obstacle on mid left
0	1	0	Obstacle on mid track
0	1	1	Obstacle on mid right
0	0	1	Obstacle on extreme right

C. Active Low switch

The safety switch is implemented on a active low configuration, which means the GPIO pin that is connected to the switch is internally pulled up to a 3.3V level, while the other end of the switch is connected to the ground. When the switch button is pressed a logic level LOW is read at the GPIO pin. This is very efficient way as no external supply is required separately for the switch. Figure in reference to [14] shows the implementation of the low active strategy.

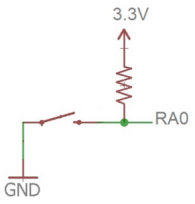

Fig. 2. Active Low Switch

VII. CODE OVERVIEW

The firmware was developed from the scratch and uses minimum number of variables, in lieu enum data structures were defined for the robot state, drive state and obstacle state and these enums are passed to functions and state machines to update the state and take necessary action. Refer to [15] to get an idea on how to code with enums. The entire code is provided in the Appendix II . The firmware starts with defining the line follow IR sensors, obstacle sensing IR sensors, switch button and LED GPIO pins as macros. Defining them as macros allows flexibility to change and improves readability of the code. In the setup () function the modes of these pins are defined, mostly all of them are defined as input pins, while switch button is defined as input but with a pull up, refer section. A couple of arrays were declared and defined, these arrays are used to store the logic levels (1 or 0) of the IR sensor pins. The control parameters, ie., the Kp, Ki, Kd values are initialised. Furthermore, some user defined functions were declared. These functions are used to handle various tasks, brief description is given below.

A. Pid Control():

This function takes the drive state enum value as the argument and assigns it to float variable with name error_val. The enum was defined in such a way that it itself is generated as the error signal. Based upon the error value and the Kp, Ki, Kd control values the function generates a control signal value, which is the PWM value which is passed on to the set_pwm() function.

B. Set_pwm():

This function takes in the control value generated by the pid controller and sets the speed of the left and righ motor accordingly. First the direction of rotation of the motors is set and then the speed in incremented or decremented to the pwm value on top of the set speed value depending upon the left or right wheel.

C. Rotate_robot():

This function is called when there is no line ahead of the motor. The pwm values are preset and the direction of the wheels is opposite to each other. This results in the motor to rotate until it finds the path.

D. Read_line_sensors():

This function reads the value of the line following IR sensors and updates the line_arr variable which is an array to store the values.

E. Steering_action():

This function has some comparison statements, which takes in the line_arr array as argument and compares the value of each element in the array and updates the state of the robot and its driving state. The reason for not using a bitmask is the number of total possible states is 32 out of which only 10 states are relevant for this proposed algorithm. Hence bitmasking would again require additional conditional statements which leads to unnecessary statements.

F. Read_obstacle_sensors():

This function reads the values of the IR sensors defined for obstacle detection and updates the obstacle_arr array.

G. Obstacle_action():

Based on the values of each element of the obstacle_arr array, the array values are compared in a bunch of if-else statements as per the Table.2 and updates the obstacle state of the robot.

H. Halt_robot():

This function halts the robot by setting the PWM values of each motor to xero. This also has an infinite while loop that keeps on executing, turning the LED on and off every 1s. To exit this infinite loop the microcontroller has to be hard reset.

I. Temporary_halt():

Like the halt_robot() function, instead of permanently stopping the robot, this function temporarily halts the robot to a stop, especially during obstacle detection and goal stop cases. While halt_robot() needs to be attended by the user, this function automatically starts the robot when the calling condition is no longer prevailing.

With the help of these user defined functions, in the void loop() function, the sensor values are read and the state of the robot is updated. If there is any obstacle, the robot state is again updated to obstacle state and the robot is halted, else the previous robot state is used in the state machine to switch to different states in a switch case and the robot's motion is defined accordingly. The sampling rate is set to 10ms by assigning a delay. The Kp, Ki, Kd values are determined by trial-and-error method and are further explained in the testing section[X].

VIII. SOFTWARE AND HARDWARE REQUIREMENTS

SOFTWARE REQUIREMENTS

- Operating System: Windows 7 or above
- IDE: Arduino

A. Introduction

The Arduino software plays a crucial role in the development and programming of the line following robot. This section outlines the software requirements for utilizing Arduino in the project, enabling efficient programming and control of the robot's functionality.

1) Arduino IDE: The Arduino Integrated Development Environment (IDE) is a software platform used for programming Arduino boards.
2) Installation: To utilize Arduino, the Arduino IDE must be installed on the computer. The Arduino website https://www.arduino.cc/ provides the necessary software and installation instructions compatible with different operating systems.
3) Code Development: The Arduino IDE supports the development of code using the Arduino programming language (based on C/C++). Users can write custom code or modify existing Arduino code libraries to control the line following robot's behaviour.
4) Libraries: Arduino libraries are prewritten code modules that simplify complex tasks and provide additional functionality. Depending on the project requirements, relevant libraries for interfacing with sensors, motors, and other components may need to be installed and utilized.
5) Arduino Board Selection: The Arduino IDE allows users to select the appropriate Arduino board for their project. The specific Arduino board model used in the line following robot project should be selected within the IDE to ensure compatibility and proper compilation of the code.

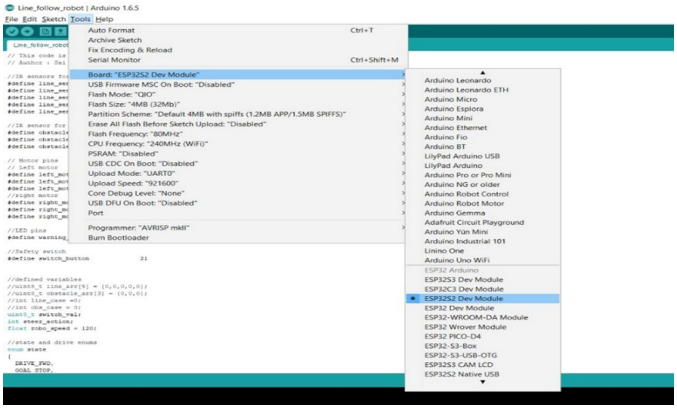

Fig. 3. The picture above clearly shows the required setup followed for our line following robot setup

B. IR Module Testing

- To test the IR module using the ESP32 microcontroller, follow these steps:Initialize a global variable 'IR_module_GPIO_pin' to represent the GPIO pin of the ESP32 as pin 2.
- In the 'setup()' function, initialize the serial communication with a baud rate of 9600.
- Set the IR Sensor pin as input and the LED pin as output.
- Print a message to the serial monitor to verify its functionality.
- In the 'loop()' function, use the 'digitalRead()' function to read the sensor pin and save the result in the 'sensorStatus_Module' variable.
- Check if the sensor's output is high or low using an 'if' statement.
- If the output is high, print "Motion Ended in the module!" to the serial monitor.
- If the output is low, print "Motion Detected in the module!" to the serial monitor.
- Repeat the loop to continuously monitor the IR module's status and display it in the serial monitor window.

HARDWARE REQUIREMENTS

1) Robot Car Kit: The Robot Car kit provides an excellent foundation for Arduino or Raspberry Pi-based robot projects. Its robust acrylic chassis features numerous mounting holes. Powered by 2 geared motors, it operates on a voltage range of 3-9V. The front includes a metal ball joint for easy and precise turning. The kit also includes an on/off switch and a battery holder.

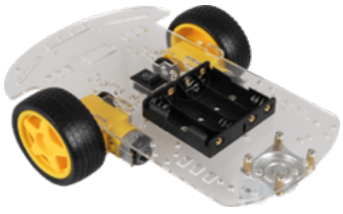

Fig. 4. Robot Car Kit

2) ESP32 Module: The ESP32 microcontroller development board offers convenient prototyping capabilities with simple programming options through Lua script or the Arduino IDE. Its breadboard-compatible design allows for easy integration. The board features 2.4GHz dual-mode Wi-Fi and Bluetooth capabilities. With 512kb SRAM and 4MB memory integrated, it provides ample storage for your projects. The board also includes various data interfaces such as UART, I2C, SPI, DAC, and ADC. The package includes the ESP circuit board.

Fig. 5. ESP32 Module

3) Motor Driver Module (L298N): The MotoDriver2 expansion board allows easy connection to a single-board computer, providing control and power supply for two DC motors. It eliminates the need for additional power supplies or excessive cables. The board supports a voltage range of 5V to 35V for motor control. The scope of delivery typically includes the expansion board and necessary connectors.

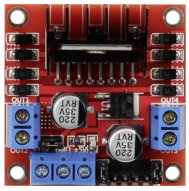

Fig. 6. Motor Driver Module (L298N)

4) IR Infrared Module: The infrared module detects proximity using infrared technology. It can be connected to microcontroller platforms like Arduino for reliable obstacle detection based on reflected infrared light. The module has two LEDs indicating the switching level: High for obstacle detected and Low for no obstacle detected. Sensitivity can be adjusted using a potentiometer, allowing detection at 3-30cm. The module has three pins: OUT for switching output, GND for ground connection, and VCC for voltage supply (3.3V5V). It provides easy connection and is an effective obstacle detection solution.
5) Push Button: A button or switch for manual control or input.It is a NO (Normally Open) contact type with a reinforced design. It has a red colour and is capable of withstanding temperature ranges from -25 to +85 degrees Celsius. The button is rated for 50,000 switching cycles, ensuring durability and reliable operation.

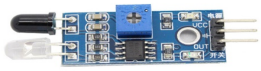

Fig. 7. IR Infrared Module

Fig. 8. Push Button

6) Other Requirements:
- Cable Set: Assorted cables and connectors for wiring.
- LEDs: Light-emitting diodes for status indication.
- Accessories: Items like tape, zip ties, etc., for assembly and mounting.

IX. CIRCUIT PICTORIAL DIAGRAM, SCHEMATIC AND POWER DISTRIBUTION

CIRCUIT PICTORIAL DIAGRAM

The circuit pictorial diagram refers to a visual representation of the robot's electrical circuitry, showing the various components and how they are connected. It provides a detailed illustration of the physical layout of the circuit, including wires, connectors, and components. The circuit pictorial diagram helps in understanding the wiring connections and the overall structure of the electrical system. The diagram emphasizes the placement of components and the paths of the wires, allowing for a clear understanding of how the electrical connections are made and the physical layout of the circuit. The circuit incorporates various components, including eight IR sensors, a warning LED, two motors, a microcontroller (ESP32), and the L298N motor driver IC. The following sections provide a detailed description of the pictorial diagram, highlighting the interconnections and functionalities of each component.

1) IR Sensors:
- Eight IR sensors are strategically placed along the robot's body to detect the line and the object.
- Each sensor is represented by the appropriate symbol in the schematic.
- The sensor outputs are connected to the input pins of the microcontroller.

2) Warning LED:
- A warning LED is included to indicate when the robot deviates from the line.
- The LED symbol is placed in the schematic, and it is connected to a digital output pin of the microcontroller.

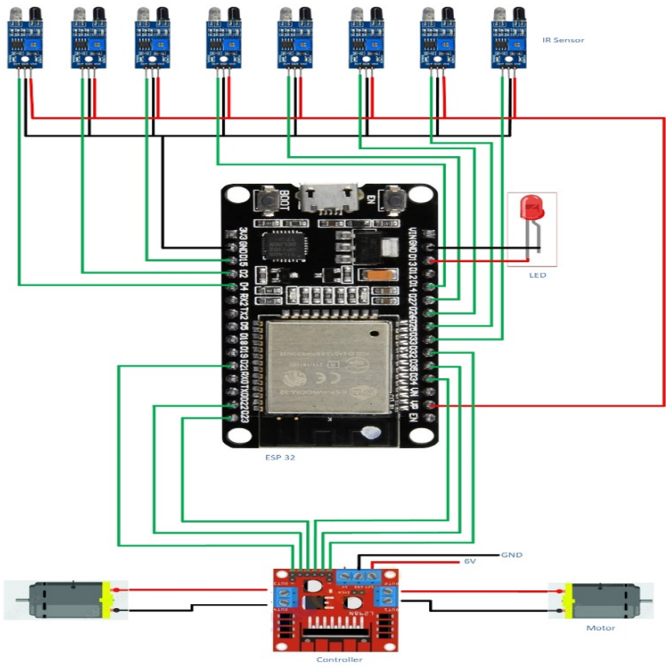

Fig. 9. Circuit Pictorial Diagram

3) Motors:
   - Two motors are responsible for driving the robot's movement.
   - The motors are symbolized in the schematic, and their connections are shown.
   - The L298N motor driver IC is used to control the motors' speed and direction.
4) Microcontroller (ESP32):
   - The microcontroller acts as the brain of the robot, processing sensor data and controlling motor movements.
   - The ESP32 symbol is placed in the schematic, and its connections to other components are depicted.
5) L298N Motor Driver IC:
   - The L298N IC is utilized to interface the microcontroller with the motors.
   - The IC symbol is included, and its connections to the microcontroller and motors are illustrated.

Schematic Power Distribution of Line Following Robot

The schematic power distribution diagram illustrates the flow of electrical power within the line following robot. It provides a clear representation of how the power source is distributed to various components, ensuring proper voltage and current supply. This section presents a comprehensive description of the power distribution schematic, emphasizing the connections and paths of power flow.

The ESP32 board and microcontroller in the line following robot are powered via a USB connection, providing a stable 5V power supply. Eight sensors are parallelly connected to the same power supply, ensuring consistent power delivery for their operation. The H-bridge motor controller receives an external supply of 6V, achieved by connecting four 1.5V batteries in series. The warning LED is supplied with a power source of 3.3V, enabling its proper functioning.

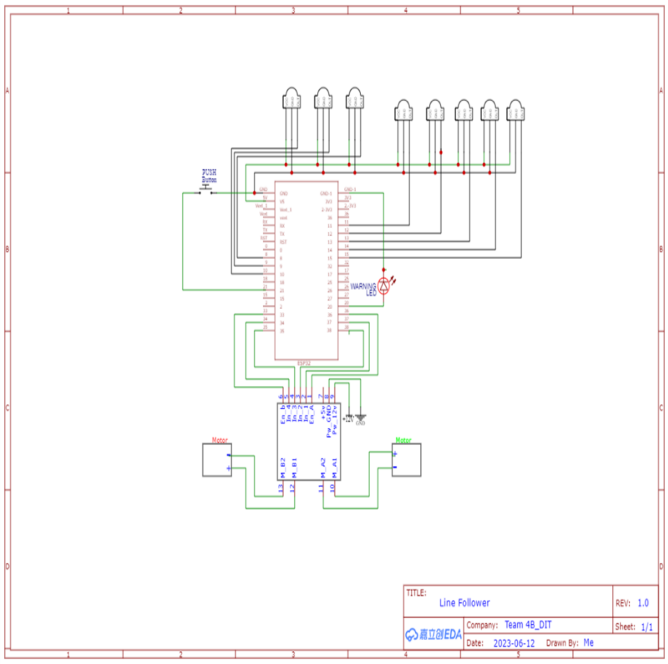

Fig. 10. Circuit Pictorial Diagram

X. Sensor Calibration

Sensor calibration is the process of adjusting sensor settings and parameters to ensure accurate and reliable measurements, aligning them with known reference values or conditions. It optimizes sensor performance, minimizes errors, and enhances the overall accuracy and precision of sensor readings [16].

1) Need For Sensor Calibration: Sensor calibration is crucial for both accurate line detection and reliable object detection in a line following robot. Here's an explanation of the importance of sensor calibration and how it enhances the robot's performance:
   - Accurate Line Detection: Proper sensor calibration ensures precise line detection, which is essential for the robot to stay on track and follow the desired path. By calibrating the sensors, you can optimize their sensitivity and threshold values, allowing them to accurately detect the contrast between the line and the surrounding surface. This ensures that the robot responds appropriately to the line's position, maintaining precise alignment and preventing deviations or errors in its movement.
   - Smooth Robot Movement: Sensor calibration minimizes false readings and erratic behavior, enabling reliable response to line changes. Fine-tuning sensor sensitivity and responsiveness allows precise motor

control and smooth transitions between line segments, enhancing overall performance.
- Optimization of Performance: Sensor calibration optimizes the line following robot's performance by aligning sensor readings with desired behavior and adapting them to specific environmental conditions. This enhances the robot's accuracy in following lines, making informed decisions based on reliable sensor inputs for improved and reliable operation.

2) Step-by-Step Calibration Procedure: In this project, we employed 8 infrared (IR) sensors, where 3 were used to identify objects and 5 were used to differentiate between objects that were black or white in colour. When set up for object detection, a robot can use the IR sensor to avoid running into walls. When set up for black-and-white colour detection, a line-tracing robot can use the IR sensor to follow either a black or white line on the floor [16].
   - STEP 1: To get the infrared sensor to work properly, one must first calibrate it. First, locate the signal LED, which may be turned on and off with the potentiometer. This is the signal LED that will be used in the calibration and detecting steps below.
   - STEP 2: **Object detection calibration**
     - The IR sensor can identify an object that is less than 9 inches away.
     - Point the infrared sensor at a wall about 2 feet away.
     - Turn the potentiometer completely counterclockwise.
     - When an object is placed within 9 inches of the signal LED, it will illuminate.
     - Turn the potentiometer clockwise until the signal LED briefly illuminates.
   - STEP 3:**Calibration for black path detection**
     - To calibrate the IR sensor for black path detection, place it about 1 to 2 inches above a black surface.
     - Turn the potentiometer completely counterclockwise.
     - When the sensor is above a white surface, the signal LED turns on, and when it is above a black surface, it turns off.
     - Turn the potentiometer clockwise until the signal LED briefly illuminates.
     - Keep the distance between the IR sensor and the black surface constant. If you need to adjust the distance, you must re-calibrate.

XI. TESTING

This testing's objective is to assess the line-following robot's functionality and performance. The robot is made to move autonomously along a predetermined course by following to a white surface and a black line. The tests are designed to evaluate the robot's proficiency in precisely detecting the line, upholding good alignment, and reacting to various line patterns and curves. The experiments were conducted in a regulated indoor setting with regular lighting. The test track was made of a white surface with a 2 inch wide, irregularly shaped black line running through it. The track has straight stretches, angular turns, T-junctions, and intersections.

To ensure a thorough assessment of a line-following robot's performance, you can take into account a variety of test cases. Here are some examples of the various test case categories we took into account.

- Path
- Obstacles
- PID Values

Test Case 1: Path

1) Path 1: For evaluating the robot's capacity to follow lines, we conducted tests along several distinct courses. Section [X] already provides an explanation of the calibration procedure required to identify black and white colors.

Fig. 11. Path 1

2) Path 2:

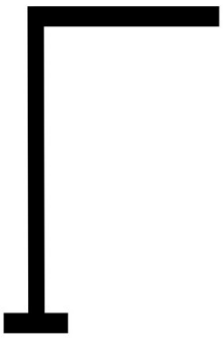

Fig. 12. Path 2

The robot can maneuver the 90-degree turn with ease and stay in line with the curved portion of the line. The outcome shows whether the robot maintains good alignment and following the curve without noticeable drift or deviations. In order to maintain stability and avoid overshooting or deviating off the path, the system can modify its speed during the turn. The outcome shows whether the robot slows down or speeds up as necessary for a precise and stable turn. The system can correctly and precisely make a 90-degree turn without going over

the line or making the turn too wide. The outcome shows whether the robot executed the turn within the desired tolerances and successfully followed the line's curved course.

3) Path 3:

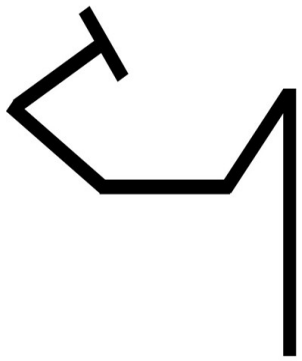

Fig. 13. Path 3

The robot can accurately follow the straight-line segments of the path without significant deviations. The robot can effortlessly follow the curved portion of the line while negotiating turns with angles less than 45 degrees. While staying in alignment with the curved segment of the line, the robot can manage bends with angles greater than 45 degrees also. The robot can successfully maneuver around obtuse angles in the path, maintaining accurate alignment and following the curved section of the line. The outcome shows if the robot performs these turns precisely and within the desired tolerances.

4) Path 4:

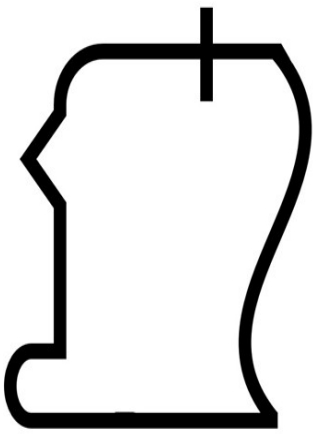

Fig. 14. Path 4

The robot can precisely follow the entirety of this predetermined path, including straight lines, turns at various angles, U-turns, and smooth curves, without experiencing any appreciable faults or deviations.

Test case 2: Obstacles

This segment can be divided into three sections based on where the obstacles are placed in relation to the path.

Obstacle placed in

- Left
- Middle
- Right

Section [X] already provides an explanation of the calibration procedure required to detect object. We don't anticipate our system to detect objects in the color black because we tuned the five sensors to follow a black line, and these three sensors (for object detection) are situated close to those sensors.

1) Obstacle position Left:

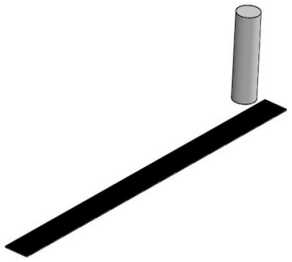

Fig. 15. Obstacle position Left

In the scenario pictured above, the system's left-hand IR sensor will pick up the object when it is 9 inches away from it. As a result, the system will go for immediate halt.

2) Obstacle position Middle:

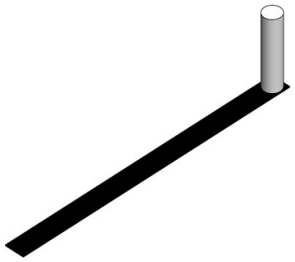

Fig. 16. Obstacle position Middle

In the scenario pictured above, the system's center IR sensor will pick up the object when it is 9 inches away from it. As a result, the system will go for immediate halt.

3) Obstacle position Right:

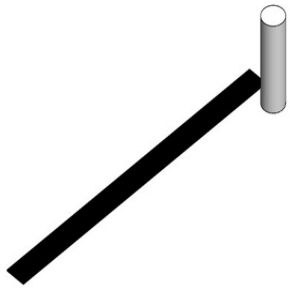

Fig. 17. Obstacle position Right

In the scenario pictured above, the system's right-hand IR sensor will pick up the object when it is 9 inches away from it. As a result, the system will go for immediate halt.

Test case 3: PID values

The PID (Proportional-Integral-Derivative) values utilized in a line-following robot's control system have a significant impact on the robot's behavior. How the robot reacts to deviations from the ideal line path is determined by the PID parameters. An overview of how various PID values may impact a line-following robot's behavior is described below:

1) Kp=5, Ki=2.5, Kd=0
   Line deviations will be responded to strongly and aggressively due to the high proportional gain (Kp=5). Systematic errors like offset or steady drift will be somewhat compensated by the integral gain (Ki=2.5). The robot won't actively slow down its response to sudden changes in the position of the line if the derivative gain is zero (Kd=0).
2) Kp=7.5, Ki=3.5, Kd=1
   With these set of values, a line-following robot displays an extremely aggressive response to line deviations, good compensation for systematic errors, and active damping of rapid changes.
3) Kp=8.5, Ki=4, Kd=0.1
   A line-following robot with PID settings of Kp=8.5, Ki=4, and Kd=0.1 demonstrates an extremely aggressive response to line deviations, significant compensation for systematic mistakes, and a slight dampening of quick shifts.
4) Kp=10, Ki=5, Kd=0.05 An highly aggressive response to line deviations, significant correction for systematic mistakes, and a modest damping of rapid changes are all displayed by a line-following robot with PID settings of Kp=10, Ki=5, and Kd=0.05. For the line-following application, this set was utilized.

Using the trial and error process, all of the above mentioned values were collected, including the final set of PID values.

XII. CHALLENGES AND BOTTLENECKS

Though the control algorithm is robust and incorporates all the necessary cases including the failsafe cases, there are few challenges faced with the hardware, especially the energy flow of the entire robot. Since the proposed algorithm aims at implementing an improved version of the existing robot with a completely new firmware developed from scratch, on the existing hardware, with additional sensors, the batteries were not able to provide enough current to all the components that are drawing from it.

1) L298N driver issues
   Though the L298N motor driver is widely used for controlling motors, it has an inherent voltage drop of 1.5 2V over the supply voltage. This is due to the drop in voltage of the switching transistors used in the H-bridge of the motor driver. This issue was verified by us and it was found that the output from the batteries was around 5.9 6V as each battery is rated 1.6V and we have 4 batteries in series. But at the motor driver output there is a significant drop in voltage of 1.8V on no-load condition, refer to the figures.

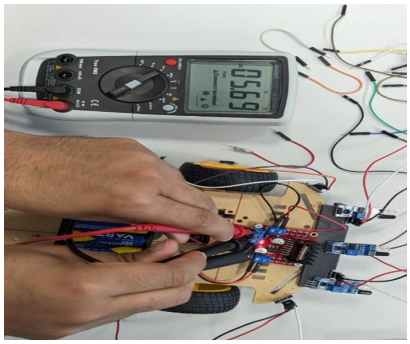

Fig. 18. Battery Output

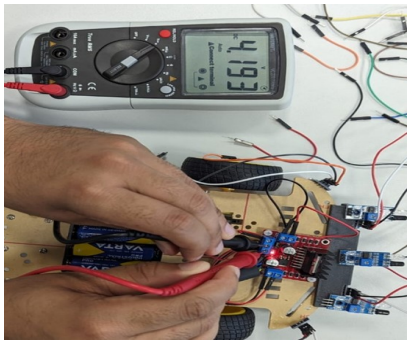

Fig. 19. Motor Driver Output

2) Current insufficiency
   Since the proposed model utilises 8 IR sensors, a motor driver, a LED, the current supplied by the power source is too less, which causes issues when calibrating the IR sensors. Also the batteries drained quick, major portion drawn by the motors, while the voltage drop was even further down to 3.2V which is very less, as the IR sensors are rated for 5V which is explained in detail in the hardware components section.

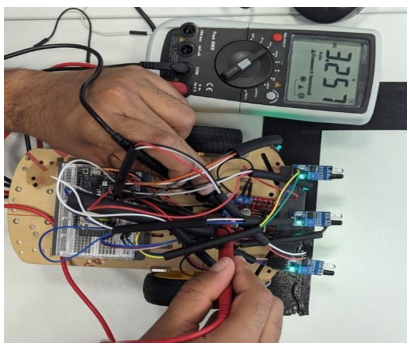

Fig. 20. Overall Supply Drop

3) Quality of the IR sensors
   The IR sensors received from the procurement team are deemed to be not from standard vendors as there are multiple issues while calibrating the sensors.

XIII. APPLICATIONS AND USE CASES

A line-following robot is a popular application of robotics and can be used in various industries and scenarios. Here are some applications and use cases for a line-following robot:

1) **Industrial Automation**:To move materials or components from one location to another on production and assembly lines, line-following robots can be used. They can move around the building by following a marked path on the floor and carry out tasks like delivering parts, moving objects, or helping with assembly.
2) **Logistics and Warehousing**: Line-following robots can be used to automate material handling tasks in warehouses and distribution centers. They can pick up and deliver packages, move objects along marked paths on the warehouse floor, and streamline the movement of goods inside the building. This can boost productivity, cut down on mistakes, and simplify the order fulfillment procedure.
3) **Robotic delivery systems**: In urban areas, mile delivery can be done by line-following robots. These robots can move through crowded areas, avoiding obstacles, and delivering packages to customers' doorsteps by following painted lines or markers on the sidewalks or designated paths. They provide a low-cost, environmentally friendly option for small-scale deliveries.
4) **Agriculture**: Applications for line-following robots in agriculture include crop monitoring and harvesting. These robots can navigate through rows of crops, gather information regarding plant health, precisely apply fertilizers or pesticides, and even carry out autonomous harvesting tasks by following the lines in the fields. In the agricultural sector, this automation can increase efficiency and lower labor costs.
5) **Inspection and upkeep**: Line-following robots with sensors and cameras can be used for inspection and upkeep tasks in sectors like infrastructure, oil and gas, and power generation. These robots can track pipes, buildings, or lines while collecting visual information and spotting anomalies or damage. They can assist with spotting potential problems, carrying out regular inspections, and carrying out maintenance operations in challenging or dangerous environments.

These are the general applications of the line following robot.it is important to remember that the actual implementation and customization of the robot's functionalities will depend on the particular requirements and context of the application.

XIV. FUTURE IMPROVEMENTS

1) **Color Recognition**: Expanding the robot's ability to recognize and follow lines of various colors is one improvement that could be made. This can be done by incorporating extra sensors, like RGB cameras or color sensors, and creating algorithms to recognize and track lines of different colors.

2) **Line Detection Algorithm Optimization**:Optimization of the line detection algorithm can result in a performance that is more accurate and dependable. It is possible to investigate methods like image processing, edge detection, or machine learning algorithms to improve the robot's capacity to recognize the line even in difficult lighting situations or complex environments.

3) **Adaptive Line Following**: By creating adaptive line-following algorithms, the robot will be able to modify its speed, turning radius, or behavior in accordance with the properties of the line. This could make navigation smoother and more effective by improving its capacity to manage curves,intersections,or irregularities in the line.

4) **Wireless Communication and Control**: Remote control, real-time monitoring, or even autonomous robot coordination with other robots can all be made possible by integrating wireless communication capabilities, such as Bluetooth or WiFi. This can increase the robot's adaptability and broaden the range of applications for it.

5) **Energy Efficiency and Battery Management**: The robot's operational time can be increased and its overall performance improved by optimizing the robot's energy consumption and battery management system. Techniques like power-saving modes, energy-efficient parts, or intelligent charging systems may be used in this.

6) **Obstacle Detection and Avoidance**: The robot's safety and navigation can be enhanced by improving its capacity to identify obstacles in its path. In order to detect and avoid objects or obstacles while following the line, proximity sensors, ultrasonic sensors, or LIDAR (light detection and ranging) technology can be used. These sensors can be integrated into line-following robots and linked to their control systems. The sensor data is continuously monitored and analyzed by the control system to identify obstacles. The navigation algorithms of the robot can be programmed to take appropriate actions, such as stopping, changing course, or navigating around the obstacle while staying on the line, if an obstacle is detected.

XV. CONCLUSION

The project successfully showed that it is possible to follow a line using inexpensive IR modules, despite the accuracy limitations. we use a less expensive method of creating line-following robots. since its a cheap IR module the result were e less accurate also. These modules still allow for the most fundamental line tracking functionality, even though they might not offer the same level of precision as more

expensive options. Because of this, the project is appropriate for learning-related endeavors, hobbies, or applications where a high degree of precision is not strictly necessary. Using less precise IR modules, the line-following robot project has shown the importance of accessibility and affordability in robotics. The project provides a starting point for people or projects with limited resources to investigate the fundamentals of line following, even though the accuracy might be compromised. The project can act as a starting point for further developments in low-cost line-following robotics by incorporating additional sensors and future advancements in control algorithms.

ACKNOWLEDGMENTS

We want to thank Prof. Dr. Josef Schmid for your valuable support in resolving doubts throughout the case study, as well as providing guidance. Additionally, we extend our appreciation to Lab Engineer Isabell Herer for facilitating the necessary material purchases.

REFERENCES

[1] L. K. Amifia, M. I. Riansyah, and P. D. Putra, "Design of logistic transporter robot system," *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika*, vol. 6, no. 1, p. 19, 2020.

[2] A. Hassan, H. M. Abdullah, U. Farooq, A. Shahzad, R. M. Asif, F. Haider, and A. U. Rehman, "A wirelessly controlled robot-based smart irrigation system by exploiting arduino," *Journal of Robotics and Control (JRC)*, vol. 2, no. 1, pp. 29–34, 2021.

[3] S. Gurav, P. Khot, D. Potadar, S. Shelke, and B. Chougula, "Remote controlled waiter robot for restaurant automation," *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, vol. 6, no. 5, pp. 156–160, 2017.

[4] S. Arunkumar, V. Kannan, and A. Palanivel, "Proposal for a cost effective automation technique in developing industries using a pid controlled line follower," 2013.

[5] O. Gumus, M. Topaloglu, and D. Ozcelik, "The use of computer controlled line follower robots in public transport," *Procedia Computer Science*, vol. 102, pp. 202–208, 2016.

[6] M. A. Kader, M. Z. Islam, J. Al Rafi, M. R. Islam, and F. S. Hossain, "Line following autonomous office assistant robot with pid algorithm," in *2018 International Conference on Innovations in Science, Engineering and Technology (ICISET)*. IEEE, 2018, pp. 109–114.

[7] J. Thirumurugan, M. Vinoth, G. Kartheeswaran, and M. Vishwanathan, "Line following robot for library inventory management system," in *INTERACT-2010*. IEEE, 2010, pp. 1–3.

[8] S. Bhat and M. Meenakshi, "Embedded system based waiter and military robot path planning," in *2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. IEEE, 2015, pp. 507–510.

[9] M. Engin and D. Engin, "Path planning of line follower robot," in *2012 5th European DSP Education and Research Conference (EDERC)*. IEEE, 2012, pp. 1–5.

[10] M. Pakdaman and M. M. Sanaatiyan, "Design and implementation of line follower robot," in *2009 second international conference on computer and electrical engineering*, vol. 2. IEEE, 2009, pp. 585–590.

[11] M. A. Kader, M. Z. Islam, J. Al Rafi, M. R. Islam, and F. S. Hossain, "Line following autonomous office assistant robot with pid algorithm," in *2018 International Conference on Innovations in Science, Engineering and Technology (ICISET)*. IEEE, 2018, pp. 109–114.

[12] G. Benet, F. Blanes, J. E. Simó, and P. Pérez, "Using infrared sensors for distance measurement in mobile robots," *Robotics and autonomous systems*, vol. 40, no. 4, pp. 255–266, 2002.

[13] F. Kaiser, S. Islam, W. Imran, K. Khan, and K. Islam, "Line follower robot: Fabrication and accuracy measurement by data acquisition," in *2014 International Conference on Electrical Engineering and Information & Communication Technology*. IEEE, 2014, pp. 1–6.

[14] E. Artistry, "Demystifying microcontroller gpio settings," https://embeddedartistry.com/blog/2018/06/04/demystifying-microcontroller-gpio-settings/, 2018, accessed: July 2, 2023.

[15] B. Wagner, pkulikov, and Thraka, "Title of the web article," Web Article, May 2023, accessed: Date. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum

[16] R. F. Fun, "Ir sensor tutorial," http://www.robotsforfun.com/webpages/irsensor.html, 2018, accessed: 05.07.2023. [Online]. Available: http://www.robotsforfun.com/webpages/irsensor.html

APPENDIX

Appendix I

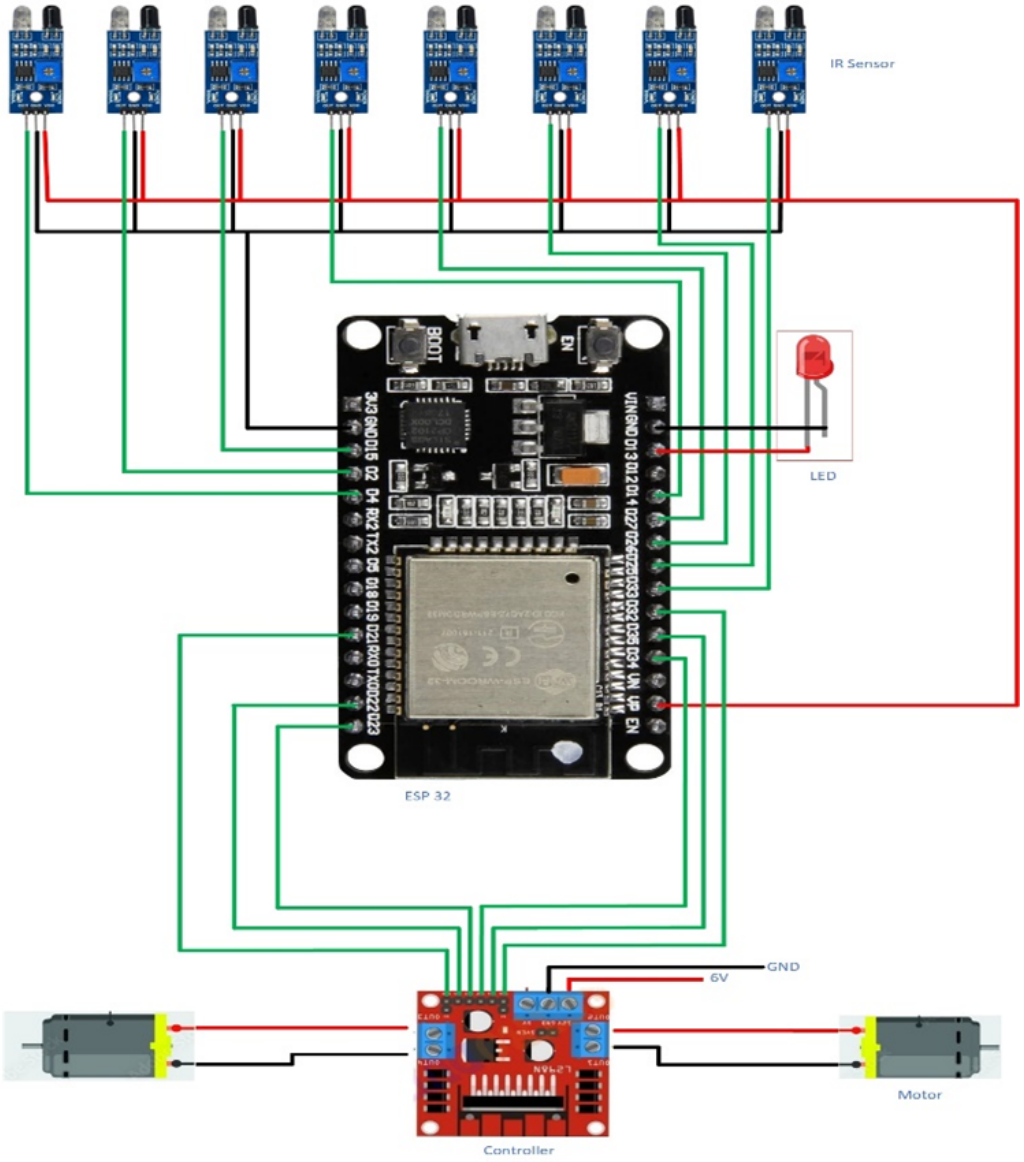

Fig. 21. Circuit Pictorial Diagram

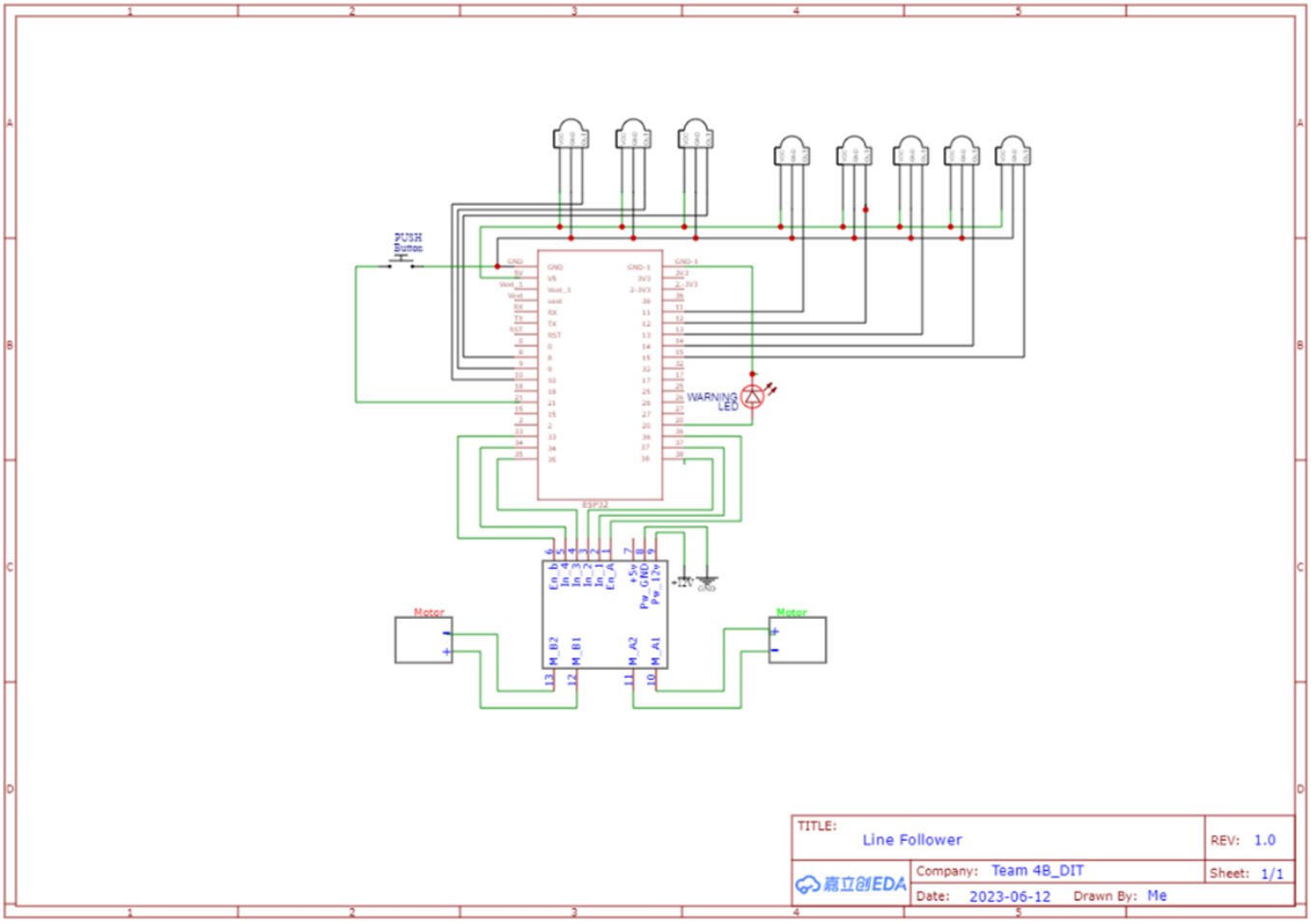

Fig. 22. Schematic Power Distribution Diagram

Appendix II

```
// Code is for Line following robot with obstacle sensing
// Author : Sai Bharadhwaj Matha, for queries contact bharadhwajsaimatha@gmail.com

//IR sensors for line following
#define line_sensor_1             11
#define line_sensor_2             12
#define line_sensor_3             13
#define line_sensor_4             14
#define line_sensor_5             15

//IR sensor for obstacle avoidance
#define obstacle_sensor_1         8
#define obstacle_sensor_2         9
#define obstacle_sensor_3         10

// Motor pins
// Left motor
#define left_motor_pwm            01     //ENB
#define left_motor_dir_1          02     //IN3
#define left_motor_dir_2          03     //IN4
//right motor
#define right_motor_pwm           04     //ENA
#define right_motor_dir_1         05     //IN1
#define right_motor_dir_2         06     //IN2

//LED pins
#define warning_led               20

//Safety switch
#define switch_button             21


//defined variables
uint8_t switch_val;
int steer_action;
float robo_speed = 120;

//state and drive enums
enum state
{
  DRIVE_FWD,
  GOAL_STOP,
  NO_LINE,
  OBSTACLE,
  FAIL_SAFE
};
enum state robo_state;
enum drive
{
  SHARP_LEFT = -4,
  HEFTY_LEFT,
  LEFT,
  LOW_LEFT,
  STRAIGHT,
  LOW_RIGHT,
  RIGHT,
```

```
  HEFTY_RIGHT,
  SHARP_RIGHT,
};
enum drive drive_state;
enum obstacle
{
  NO_OBSTACLE,
  EXTREME_LEFT,
  MID_LEFT,
  MIDDLE,
  MID_RIGHT,
  EXTREME_RIGHT,
  SOMEWHERE
};
enum obstacle obstacle_state;

//control variables
float Kp = 10;
float Ki = 5;
float Kd = 0.05;
float prev_error = 0;
float error = 0;
float error_integral = 0;
float error_derivative = 0;
float control_signal = 0;

//defined functions

float PID_control(float error_val)
{
  error_integral = prev_error + error_val;
  error_derivative = error_val - prev_error;
  control_signal = (Kp*error_val) + (Ki*error_integral) + (Kd*error_derivative);
  prev_error = error_val;
  return control_signal;
}
void set_pwm(float ctrl_sig)
{
  digitalWrite(left_motor_dir_1, LOW);
  digitalWrite(left_motor_dir_2, HIGH);
  digitalWrite(right_motor_dir_1, HIGH);
  digitalWrite(right_motor_dir_2, LOW);
  analogWrite(left_motor_pwm, robo_speed+ctrl_sig);
  analogWrite(right_motor_pwm, robo_speed - ctrl_sig);
//  Serial.println(robo_speed+ctrl_sig);
}

void rotate_robot()
{
  digitalWrite(left_motor_dir_1, HIGH);
  digitalWrite(left_motor_dir_2, LOW);
  digitalWrite(right_motor_dir_1, HIGH);
  digitalWrite(right_motor_dir_2, LOW);
  analogWrite(left_motor_pwm, robo_speed - 20);
  analogWrite(right_motor_pwm, robo_speed + 20);
}

void steering_action(int steer_arr[5])
```

```
{
//  Serial.println("started in steering action\n");
  if((steer_arr[0] == 0)&&(steer_arr[1] == 0)&&(steer_arr[2] == 0)&
  (steer_arr[3] == 0)&&(steer_arr[4] == 0))
  {
    robo_state = NO_LINE;
  }
  else if((steer_arr[0] == 0)&&(steer_arr[1] == 0)&&(steer_arr[2] == 1)&&
  (steer_arr[3] == 0)&&(steer_arr[4] == 0))
  {
    robo_state = DRIVE_FWD;
    drive_state = STRAIGHT;
  }
  else if((steer_arr[0] == 0)&&(steer_arr[1] == 1)&&(steer_arr[2] == 1)&&
  (steer_arr[3] == 0)&&(steer_arr[4] == 0))
  {
    robo_state = DRIVE_FWD;
    drive_state = LOW_LEFT;
  }
  else if((steer_arr[0] == 0)&&(steer_arr[1] == 1)&&(steer_arr[2] == 0)&&
  (steer_arr[3] == 0)&&(steer_arr[4] == 0))
  {
    robo_state = DRIVE_FWD;
    drive_state = LEFT;
  }
  else if((steer_arr[0] == 1)&&(steer_arr[1] == 1)&&(steer_arr[2] == 0)&&
  (steer_arr[3] == 0)&&(steer_arr[4] == 0))
  {
    robo_state = DRIVE_FWD;
    drive_state = HEFTY_LEFT;
  }
  else if((steer_arr[0] == 1)&&(steer_arr[1] == 0)&&(steer_arr[2] == 0)&&
  (steer_arr[3] == 0)&&(steer_arr[4] == 0))
  {
    robo_state = DRIVE_FWD;
    drive_state = SHARP_LEFT;
  }
  else if((steer_arr[0] == 0)&&(steer_arr[1] == 0)&&(steer_arr[2] == 1)&&
  (steer_arr[3] == 1)&&(steer_arr[4] == 0))
  {
    robo_state = DRIVE_FWD;
    drive_state = LOW_RIGHT;
  }
  else if((steer_arr[0] == 0)&&(steer_arr[1] == 0)&&(steer_arr[2] == 0)&&
  (steer_arr[3] == 1)&&(steer_arr[4] == 0))
  {
    robo_state = DRIVE_FWD;
    drive_state = RIGHT;
  }
  else if((steer_arr[0] == 0)&&(steer_arr[1] == 0)&&(steer_arr[2] == 0)&&
  (steer_arr[3] == 1)&&(steer_arr[4] == 1))
  {
    robo_state = DRIVE_FWD;
    drive_state = HEFTY_RIGHT;
  }
  else if((steer_arr[0] == 0)&&(steer_arr[1] == 0)&&(steer_arr[2] == 0)&&
  (steer_arr[3] == 0)&&(steer_arr[4] == 1))
  {
```

```
    robo_state = DRIVE_FWD;
    drive_state = SHARP_RIGHT;
  }
  else if((steer_arr[0] == 1)&&(steer_arr[1] == 1)&&(steer_arr[2] == 1)&&
  (steer_arr[3] == 1)&&(steer_arr[4] == 1))
  {
    robo_state = GOAL_STOP;
  }
  else
  {
    robo_state = DRIVE_FWD;
    drive_state = STRAIGHT;
  }

}

void read_line_sensors(int* line_arr)
{
  line_arr[0] = digitalRead(line_sensor_1);
  line_arr[1] = digitalRead(line_sensor_2);
  line_arr[2] = digitalRead(line_sensor_3);
  line_arr[3] = digitalRead(line_sensor_4);
  line_arr[4] = digitalRead(line_sensor_5);
}
void obstacle_action(int obst_arr[3])
{
  if((obst_arr[0] == 0)&&(obst_arr[1] == 0)&&(obst_arr[2] == 0))
  {
    obstacle_state = NO_OBSTACLE;
  }
  else if((obst_arr[0] == 1)&&(obst_arr[1] == 0)&&(obst_arr[2] == 0))
  {
    obstacle_state = EXTREME_LEFT;
  }
  else if((obst_arr[0] == 1)&&(obst_arr[1] == 1)&&(obst_arr[2] == 0))
  {
    obstacle_state = MID_LEFT;
  }
  else if((obst_arr[0] == 0)&&(obst_arr[1] == 1)&&(obst_arr[2] == 0))
  {
    obstacle_state = MIDDLE;
  }
  else if((obst_arr[0] == 0)&&(obst_arr[1] == 1)&&(obst_arr[2] == 1))
  {
    obstacle_state = MID_RIGHT;
  }
  else if((obst_arr[0] == 0)&&(obst_arr[1] == 0)&&(obst_arr[2] == 1))
  {
    obstacle_state = EXTREME_RIGHT;
  }
  else
  {
    obstacle_state = SOMEWHERE;
  }
// return obstacle_state;
}
```

```
//void obstacle_action(int obst_arr[3])
//{
//  if((obst_arr[0] == 1)&&(obst_arr[1] == 1)&&(obst_arr[2] == 1))
//  {
//    obstacle_state = NO_OBSTACLE;
////    Serial.println("this is updated\n");
////    Serial.println(obstacle_state);
//  }
//  else if((obst_arr[0] == 0)&&(obst_arr[1] == 1)&&(obst_arr[2] == 1))
//  {
//    obstacle_state = EXTREME_LEFT;
//  }
//  else if((obst_arr[0] == 0)&&(obst_arr[1] == 0)&&(obst_arr[2] == 1))
//  {
//    obstacle_state = MID_LEFT;
//  }
//  else if((obst_arr[0] == 1)&&(obst_arr[1] == 0)&&(obst_arr[2] == 1))
//  {
//    obstacle_state = MIDDLE;
//  }
//  else if((obst_arr[0] == 1)&&(obst_arr[1] == 0)&&(obst_arr[2] == 0))
//  {
//    obstacle_state = MID_RIGHT;
//  }
//  else if((obst_arr[0] == 1)&&(obst_arr[1] == 1)&&(obst_arr[2] == 0))
//  {
//    obstacle_state = EXTREME_RIGHT;
//  }
//  else
//  {
//    obstacle_state = SOMEWHERE;
//  }
//// return obstacle_state;
//}


void read_obstacle_sensors(int* obstacle_arr)
{
  obstacle_arr[0] = digitalRead(obstacle_sensor_1);
  obstacle_arr[1] = digitalRead(obstacle_sensor_2);
  obstacle_arr[2] = digitalRead(obstacle_sensor_3);
}

void halt_robot()
{
  analogWrite(left_motor_pwm, 0);
  analogWrite(right_motor_pwm, 0);
  digitalWrite(left_motor_dir_1, LOW);
  digitalWrite(left_motor_dir_2, LOW);
  digitalWrite(right_motor_dir_1, LOW);
  digitalWrite(right_motor_dir_2, LOW);
  while(true)
  {
    digitalWrite(warning_led, HIGH);
    delay(500);
    digitalWrite(warning_led, LOW);
    delay(500);
  }
```

```
}

void temporary_halt()
{
//  Serial.println("Robot is temporarily haulted\n");
  analogWrite(left_motor_pwm, 0);
  analogWrite(right_motor_pwm, 0);
  digitalWrite(left_motor_dir_1, LOW);
  digitalWrite(left_motor_dir_2, LOW);
  digitalWrite(right_motor_dir_1, LOW);
  digitalWrite(right_motor_dir_2, LOW);
}

void setup()
{
  Serial.begin(115200);

  //Line Follow sensors
  pinMode(line_sensor_1, INPUT);
  pinMode(line_sensor_2, INPUT);
  pinMode(line_sensor_3, INPUT);
  pinMode(line_sensor_4, INPUT);
  pinMode(line_sensor_5, INPUT);

  //Obstacle IR sensors
  pinMode(obstacle_sensor_1, INPUT);
  pinMode(obstacle_sensor_2, INPUT);
  pinMode(obstacle_sensor_3, INPUT);

  //Motor pins
  pinMode(left_motor_pwm, OUTPUT);
  pinMode(left_motor_dir_1, OUTPUT);
  pinMode(left_motor_dir_2, OUTPUT);

  pinMode(right_motor_pwm, OUTPUT);
  pinMode(right_motor_dir_1, OUTPUT);
  pinMode(right_motor_dir_2, OUTPUT);

  //Safety_switch and LED
  pinMode(switch_button, INPUT_PULLUP);
  pinMode(warning_led, OUTPUT);

  //Initialisation
  digitalWrite(left_motor_dir_1, LOW);
  digitalWrite(left_motor_dir_2, LOW);
  digitalWrite(right_motor_dir_1, LOW);
  digitalWrite(right_motor_dir_2, LOW);
  analogWrite(left_motor_pwm, 0);
  analogWrite(right_motor_pwm, 0);
}

void loop()
{

  //Ready safety switch state
  if(digitalRead(switch_button) == LOW)
  {
    robo_state = FAIL_SAFE;
```

```
  }
  int line_arr[5];
  int obstacle_arr[3];
  read_line_sensors(line_arr);
  read_obstacle_sensors(obstacle_arr);
  obstacle_action(obstacle_arr);
  steering_action(line_arr);
  if(obstacle_state != NO_OBSTACLE)
  {
  switch(robo_state)
  {
    case OBSTACLE:
      temporary_halt();
      break;

    case GOAL_STOP:
      temporary_halt();
      break;

    case FAIL_SAFE:
      halt_robot();
      break;

    case NO_LINE:
      rotate_robot();
      break;

    case DRIVE_FWD:
      set_pwm(PID_control(drive_state));
      break;
  }
 delay(10);
}
```

Appendix III

Fig. 23. Final Testing Path