

# Design and Implementation of an Autonomous Line Following Robot with Obstacle Avoidance

Muhammad Mujhtaba Qaiser, Victor Odadou,  
Ikramul Hassan Sazid, Muhammad Umer Hayat  
Systems Engineering and Prototyping, BSc Electronic Engineering  
Hochschule Hamm-Lippstadt, Lippstadt, Germany

{Muhammad-Mujhtaba.Qaiser, Victor.Odadou,  
ikramul-hassan.sazid, Muhammad-Umer.Hayat}@stud.hshl.de

## I. INTRODUCTION

The prototyping of autonomous robots is an essential skill for embedded systems engineers, bridging hardware and software through practical experimentation. Robotics is an excellent platform for students to learn system integration, embedded programming, and control theory in a practical setting [1]. This project focuses on designing and assembling a robot able to follow a black line using IR sensors, control DC motors via an L298N module, and (optionally) avoid obstacles with an ultrasonic sensor. Our implementation aims to maximize reliability and modularity by using widely available hardware and open-source programming.

## II. SYSTEM DESIGN AND ENGINEERING

### A. Requirements

The robot must:

- Reliably follow a black line on a white surface
- Stop or pause when both sensors detect loss of line
- Detect object in its path.
- Avoid object and return to follow the line
- Use off-the-shelf components (Arduino Uno, L298N, DC motors, IR sensors)
- Modular design for easy expansion (e.g., adding obstacle avoidance)

A careful requirements engineering process is essential to ensure the final product meets the desired functionalities, especially in educational robotics where system modularity is key [2].

### B. Components

Key components:

- **IR Sensors:** Used for line detection, typically reflective type modules (e.g., TCRT5000 or KY-033).
- **Arduino Uno:** Main controller, reads sensors, and drives motors.
- **L298N Motor Driver:** Controls the speed / direction of two DC motors via PWM.
- **DC Motors:** Provide movement (left/right wheels).
- **(Optional) Ultrasonic Sensor:** for detection and avoidance of obstacles.
- **Chassis:** Wooden base to mount components.

## III. ACTIVITY DIAGRAM

The major diagram used to describe an activity is called an activity diagram. It describes and defines the actions in an activity along with the flow of input/output and control between them[11]. Furthermore, activity diagrams are a great communication tool between the developers and the user, this is due to activity diagrams being able to present complex processes in a visual format. For our project we found that an activity diagram was needed as it was important to show how the flow and functions of the car. The diagram starts with

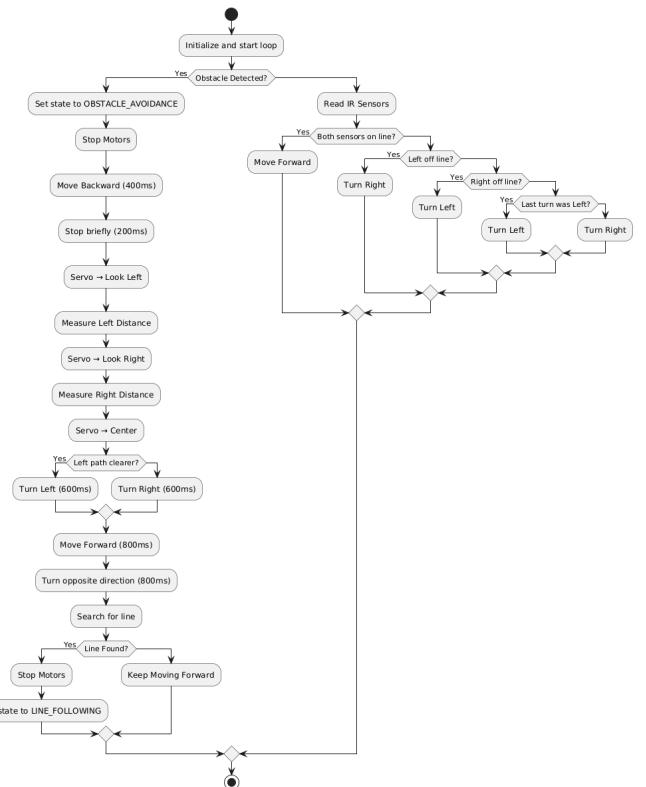


Fig. 1. Activity Diagram

the robot performing a continuous sensor scan. If a black line is detected, it then checks whether an obstacle is also present ahead. If there's no obstacle, the robot simply follows the

black line. If an obstacle is detected, it measures the distance to that object. If the object is closer than 20 cm, the robot stops and waits or signals before deciding what to do next. It uses ultrasonic sensor to detect which way it has free space to go, avoids the object and comes back to the black line. If the obstacle is farther than 20 cm, it continues following the line without interruption. Whenever the line is lost (no black line detected), the robot switches to a “search for line” mode, looking around until it spots the line again. All these decisions loop back into the main sensor-scan step, allowing the robot to react in real time to both line and obstacle inputs.

#### IV. STATE MACHINE DIAGRAM

A state machine diagram is the primary tool for modeling the various modes or “states” of a system and the events that trigger transitions between them. It clearly defines each distinct behavior—such as moving forward, turning left, turning right, or stopping—and shows how the robot moves from one state to another based on sensor inputs. By visualizing conditions and guards on each transition, a state machine diagram gives a precise, step-by-step account of the control logic. The robot

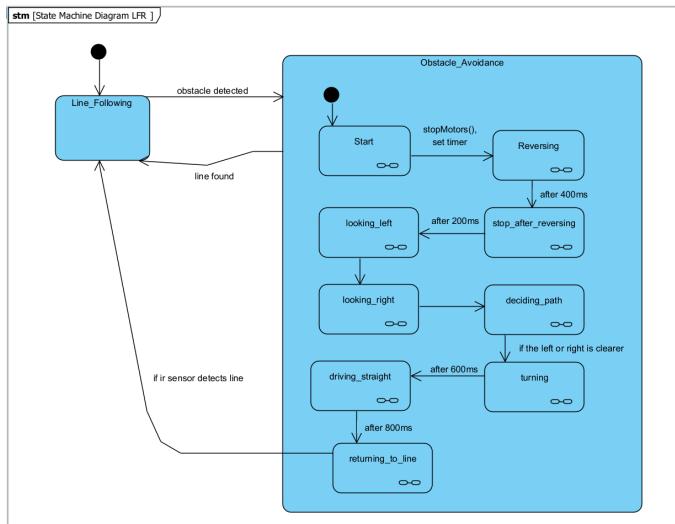


Fig. 2. State Machine Diagram

begins in an Idle state, sitting still until its IR sensors detect the black line. Once a line is spotted, it enters LineFollowing, driving straight ahead as long as both sensors stay on the line. If the ultrasonic sensor flags an obstacle while following, it transitions to ObstacleDetected, where it stops or carries out an avoidance maneuver. After the obstacle is cleared and the line is found again, it returns to LineFollowing. When either IR sensor re-detects the line from that state, the robot shifts back into LineFollowing and resumes its course.

#### V. HARDWARE COMPONENTS

##### A. IR Sensor

Infrared sensors are used to detect the black line against the white background. These modules consist of an emitter and a receiver; the IR LED emits infrared light, and the

photodiode detects reflected IR, which changes based on the color/reflectivity of the surface beneath. The sensor outputs a digital high or low depending on the amount of reflected IR. In the project, two KY033LT IR sensors are used.

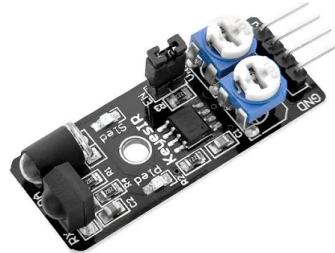


Fig. 3. KY033LT Infrared Sensor

*“Infrared sensors provide a reliable method for detecting line boundaries on contrasting surfaces and are commonly used in line-following robots due to their simplicity and low cost.” [3]*

##### B. Ultrasonic Sensor

The ultrasonic sensor measures the distance to the obstacles by emitting sound pulses and measuring their echo time. This allows the robot to detect nearby objects and perform basic avoidance maneuvers.



Fig. 4. Ultrasonic distance sensor (e.g., HC-SR04)

*“Ultrasonic sensors enable non-contact measurement of distances, providing real-time data for obstacle detection and avoidance.” [4]*

##### C. L298N Motor Driver

The L298N is a dual H-bridge driver that allows bidirectional control of two DC motors, supporting both direction and speed (via PWM). It features four input pins (for direction) and two enable pins (for speed control).

*“The L298N dual H-bridge driver module is widely used in robotics for efficient control of DC motors, allowing bidirectional operation and PWM-based speed regulation.” [5]*

##### D. DC Motors

DC motors provide the mechanical power to move the robot. By adjusting the direction and voltage applied via the L298N, we can move the robot forward, backward, or turn.

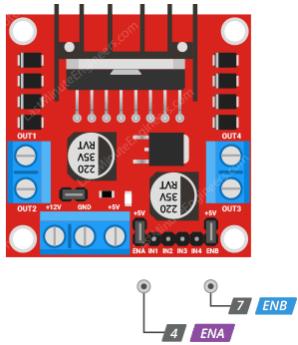


Fig. 5. L298N H-Bridge motor driver module



Fig. 6. DC Motor

#### E. Color sensor

The TCS3200 color sensor uses an array of red, green, and blue photodiodes with filters to convert light intensity into a square-wave output frequency. By selecting which color filter to activate, the chip outputs a frequency proportional to the intensity of that color component in the ambient light. Four scaling pins let you adjust the output frequency range to suit your microcontroller's sampling speed. A fifth pin provides the actual frequency output, which you measure with a digital input and translate into color values. Because it directly outputs a frequency, it's easy to interface with most Arduino boards without needing an analog-to-digital converter. This makes the TCS3200 a popular, low-cost choice for simple color-detection tasks in robotics and automation.



Fig. 7. Color Sensor TCS3200

#### F. Arduino Uno

The Arduino Uno acts as the robot's "brain." It reads sensor values, makes decisions, and sends control signals to the motor

driver. Its simplicity, open-source software, and large user community make it perfect for prototyping and learning.

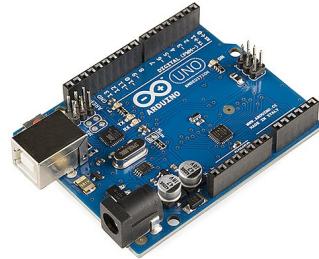


Fig. 8. Arduino Uno microcontroller board

*"Arduino Uno's open-source nature and extensive community support make it an ideal platform for educational robotics and prototyping."* [6]

#### VI. PROTOTYPING AND ASSEMBLY

All components were mounted on a plywood (or acrylic) chassis. IR sensors were placed at the front, spaced to match the line width for optimal detection. If necessary, holes were reconfigured to reposition the sensors closer. The L298N and Arduino Uno were fixed at the front of the chassis, with the Arduino on the left and the L298N adjacent to it. The servo motor and ultrasonic sensor were placed in between, while the breadboard and battery were mounted at the central back. All connections were soldered or placed using jumper wires.

#### VII. CIRCUIT DESIGN

Pin assignments:

- D3, D10: Motor driver enables (PWM)
- D5, D6: Left motor (IN1, IN2)
- D7, D8: Right motor (IN3, IN4)
- D4: Left IR sensor OUT
- D2: Right IR sensor OUT
- (Optional) A0, A1: Analog IR or other sensors
- (Optional) D11, D12: Ultrasonic trig/echo

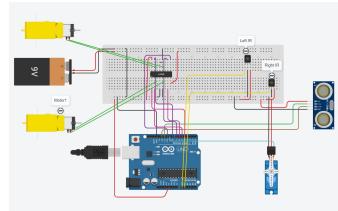


Fig. 9. Complete circuit schematic of the robot

#### VIII. MAIN DESIGN

In shaping our vehicle, we drew on a variety of tools and techniques that we'll dive into shortly. At the heart of our workflow was SolidWorks—the CAD platform we used to model, refine, and bring every part of our design to life.

### A. Main Board

The plywood chassis acts as both the structural backbone and mounting platform for our robot. Drilled holes laid out to match the footprints of the Arduino, motor driver, battery pack, and sensor modules. Each component sits on nylon standoffs or spacers and is fastened with machine screws and lock nuts, ensuring a snug fit that resists vibration while still allowing quick removal for maintenance. Cable routing channels were carved into the board so that wiring stays neat and clear of moving parts, and any future add-ons (IR sensors, Ultrasonic sensors) can simply be aligned to existing holes or tapped where needed.

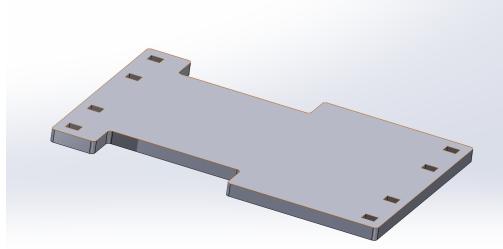


Fig. 10. Main chassis

### B. Motor Holder

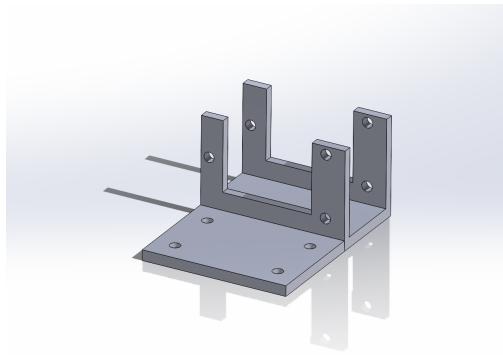


Fig. 11. Motor Holder

We crafted custom motor mounts in SolidWorks that perfectly cradle both DC motors within our chassis. Each holder features a contoured pocket and precise mounting flanges, ensuring the motors lock in place without any unwanted play. This design not only keeps the wheel axles aligned for smooth rotation but also maintains enough clearance around the gears so the wheels spin freely and the assembly stays rock-steady during operation.

## IX. SOFTWARE IMPLEMENTATION

### A. Main Loop

This code segment implements a finite state machine for autonomous robot navigation, wherein the system transitions from a line-following state to an obstacle-avoidance state upon

detecting an obstruction, subsequently executing behavior appropriate to the active state.

```

void loop() {

    if (currentState == LINE_FOLLOWING &&
        isObstacleDetected()) {
        currentState = OBSTACLE_AVOIDANCE;
        currentAvoidanceStep = START;
    }

    // Main state machine
    switch (currentState) {
        case LINE_FOLLOWING:
            followLine();
            break;
        case OBSTACLE_AVOIDANCE:
            avoidObstacle();
            break;
    }
}

```

### B. Line Following Logic

This function implements a reactive line-following algorithm using binary infrared sensor input. The robot interprets sensor readings to determine line presence: simultaneous high signals (black surface) from both left and right sensors result in forward motion; asymmetrical readings induce directional turning (left or right) to correct course. If both sensors detect absence of line (white surface), the robot initiates a recovery behavior by turning in the last known direction of deviation, by the stored ‘lastTurn’ value, enhancing line reacquisition.

```

void followLine() {
    int leftSensor = digitalRead(irLeft);
    int rightSensor = digitalRead(irRight);

    // Both sensors on black: move forward
    if (leftSensor == 1 && rightSensor == 1) {
        moveForward();
        lastTurn = 0;
    }
    // Left on white, right on black: turn right
    else if (leftSensor == 0 && rightSensor == 1) {
        turnRight();
        lastTurn = 2;
    }
    // Right on white, left on black: turn left
    else if (leftSensor == 1 && rightSensor == 0) {
        turnLeft();
        lastTurn = 1;
    }
    // Both on white: search for the line
    else {
        if (lastTurn == 1) {
            turnLeft();
        } else {

```

```

        turnRight();
    }
}
}

```

### C. void objectAvoid()

The avoidObstacle() function defines a structured, multi-phase obstacle avoidance routine based on a finite state machine. Each case within the switch statement represents a distinct behavioral step: initiating avoidance (START), reversing from the obstacle , pausing , environmental scanning, LOOKINGRIGHT, decision-making , executing a turn (TURNING), advancing past the obstacle , and re-aligning with the original path . This modular design enables adaptive navigation around obstacles while preserving the robot's overall trajectory fidelity. Keep the report brief, the inner working of the each cases are abstracted here.

```

// === Main Behaviors ===

void avoidObstacle() {
    unsigned long currentTime = millis();

    switch (currentAvoidanceStep) {
        case START:
        case REVERSING:
        case STOP_AFTER_REVERSE:
        case LOOKING_LEFT:
        case LOOKING_RIGHT:
        case DECIDING_PATH:
        case TURNING:
        case DRIVING_STRAIGHT:
        case RETURNING_TO_LINE:
        case SEARCHING_FOR_LINE:
    }
}

```

## X. TESTING AND RESULTS

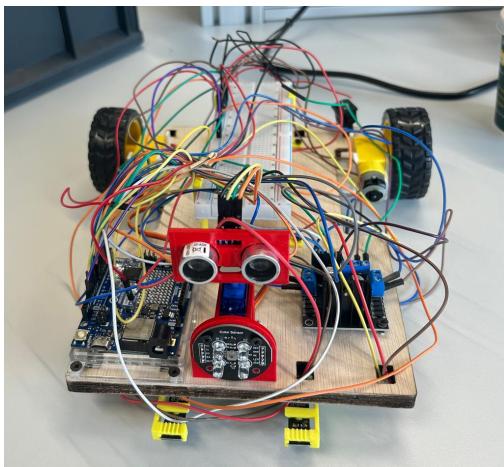


Fig. 12. Final robot car.

The robot was tested on various tracks with straight and curved lines. Reliability was highest when the IR sensors were closely spaced (1–2cm apart). Motors were tuned using PWM to ensure no overshooting. Performance was limited by surface reflectivity and ambient light.

The car is able to detect obstacle in its path and is able to avoid it and go around it. after avoiding it, we faced difficulty to find the line again.

## XI. CONCLUSION

A low-cost autonomous robot was designed and implemented using IR-based line detection and an Arduino control platform. The robot reliably followed black lines on a white floor, demonstrating the effectiveness of simple control logic and modular design for educational robotics. Future improvements could include PID control, speed regulation, and multi-sensor fusion for complex tracks.

## APPENDIX: TASKS AND COMMITMENTS

### A. Muhammad Mujhtaba Qaiser

**IR Sensor Design:** He developed a SolidWorks model for the IR sensor mounts, exploring optimal positioning and housing. While the printed version was not used, the design process supported the team's understanding of spatial constraints and sensor layout. **Repository Organization:** He organized and maintained the project's GitHub repository, establishing a clear folder structure, documentation workflow, and version control practices to streamline collaboration and development. **System Diagrams:** He was responsible for creating several system-level diagrams to guide the software and system architecture:

- State Machine Diagram to define robot behaviors and transitions,
- Block Definition Diagram to illustrate component relationships,
- Activity Diagram to visualize process flow,
- Sequence Diagram to map interactions between system components over time.

**Sensor Mounting and Motor Adjustment:** He assisted with physically mounting the IR sensors onto the chassis, ensuring proper alignment. Additionally, he re-tightened one of the motors to enhance mechanical stability.

**Cable Management:** He organized the robot's internal wiring using zip ties, improving safety, accessibility, and the overall cleanliness of the hardware setup.

### B. Victor Odadou

**Line Following Algorithm:** Developed the main Arduino code that enables the vehicle to follow a black line. This involved integrating the input from the two downward-facing IR sensors and translating their state into specific motor commands. The logic was built to control motor speed with PWM signals, allowing for both forward movement and corrective turns. **Off-Track Recovery Logic:** Designed and implemented the crucial software logic that allows the vehicle to recover after it has lost the line. **Paper Prototyping:** Created the initial paper prototype sketch of the vehicle. This low-fidelity model

was used to validate the overall vehicle design and layout before proceeding with the physical build.

was essential for planning the physical layout of all electronic components. Hardware Assembly: assisted the team with the general assembly of the prototype. This included screwing components onto the chassis, helping to determine the final placement of parts, and ensuring all connections were secure.

#### C. *Ikramul Hassan Sazid*

Physical Modeling: Design of the two motor holders in Solidworks to ensure it can hold two dc motors in the provided chasis and enable the wheels to run freely in stable form. Code Responsibilities: Obstacle Avoiding Feature: Developing code to implement the obstacle avoiding feature using ultrasonic sensor that allows the car to detect any object in the path of following line, and being able to successfully avoid the clash and avoid it. Afterwards the car is designed to find the line and continue following it again. Color Sensor: Developed the color library and logic to differentiate red, green, blue color of the in order for the robot to detect the color of the obstacle and be able to act in a predetermined action depending on the color. Implement the color sensor properly with the rest of the code.

#### D. *Muhammad Umer Hayat*

Contributed in both the mechanical and software aspects of the project. During the assembly phase, responsible for installing and securely fixing key hardware elements—including sensors, motors, and the control board—onto the chassis using the available tools, ensuring precise alignment and mechanical stability. Assisted in the soldering process, helping to connect wires to various electronic components and contributing to the overall reliability of the electrical system.

#### ACKNOWLEDGMENT

We thank our university lab instructors and peers for support and collaborative discussions.

#### REFERENCES

- [1] M. B. Dias, S. Thayer, B. Browning, et al., "Sliding autonomy for remote robotic exploration," *IEEE Robotics & Automation Magazine*, vol. 12, no. 2, pp. 77-87, 2005.
- [2] R. Davis, M. F. Collins, "A model-driven engineering approach to requirements in robotics," *IEEE International Conference on Robotics and Automation*, 2017.
- [3] I. Khemapech, J. Ichalkaranje, "A review of line following robot algorithms," *IEEE Int. Conf. on Industrial Technology*, 2018.
- [4] C. Y. Chong, S. P. Kumar, "Sensor networks: evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247-1256, 2003.
- [5] S. R. Ahmad, M. S. Abdullah, S. Z. Zainal, et al., "Development of a low-cost mobile robot using Arduino microcontroller," *IEEE Control and System Graduate Research Colloquium*, 2016.
- [6] P. Daponte, L. De Vito, F. Picariello, et al., "Educational robotics: An innovative tool for developing future engineers," *IEEE Instrumentation & Measurement Magazine*, vol. 21, no. 6, pp. 28-37, 2018.