Obstacle Avoidance & Line Following Robot

Miguel Cid Echevarria Mohanad Kandil Naman Kumar
Owen Allah Naamneh Prashant Kumar Mishra Serdar Gülbahar
Urko Cornejo Merodio Vlad Dumitrescu

1 September 2025

Code Documentation

This project describes the design and implementation of a simple autonomous robot that combines line following with ultrasonic obstacle avoidance. The hardware setup consists of two IR reflective sensors used for detecting the line, one ultrasonic sensor (HC-SR04) mounted on a servo motor to scan the surroundings, and four DC motors connected to the Adafruit Motor Shield V1 (L293D). The system is programmed in Arduino C++, using the libraries `AFMotor.h`, `Servo.h`, and `NewPing.h`.

The robot's core behavior is divided into two main tasks: line following and obstacle avoidance. The line following module reads signals from the left and right IR sensors. Depending on the readings, the robot moves forward when both sensors detect the background, steers left or right if only one detects the line, or stops when both sensors see the line simultaneously. This ensures the robot maintains alignment with the path.

Obstacle avoidance is triggered when the ultrasonic sensor detects an object within a set threshold (default 15 cm). The robot stops, sweeps its servo left and right to measure available space, and selects the direction with the larger distance. It then performs a fixed sequence: back up, pivot toward the chosen direction, drive forward to bypass the obstacle, and finally pivot into the opposite direction to re-approach the line. The system keeps the ultrasonic sensor re-centered after each scan to maintain consistent orientation.

Tuning constants allow flexible calibration. These include the servo forward angle, obstacle distance, and timing values for back up, pivoting, and re-acquisition. Additionally, motor and IR polarity can be inverted via software flags, which simplifies hardware adaptation. Calibration involves aligning the servo to face forward, testing IR polarity on black and white surfaces, and adjusting avoidance timings according to the robot's speed and surface conditions.

In summary, this project demonstrates a reliable yet simple approach to autonomous navigation. By combining deterministic avoidance maneuvers with flexible line-following control, the robot can effectively follow paths and recover its route even after encountering obstacles.

Key Code Snippets

The full project code is extensive, but the following snippets highlight the core logic for the robot's behavior.

Global Configuration and Tunables

This section of the code defines the pins, motor configurations, and crucial tuning constants. These values are a critical part of the calibration process mentioned in the documentation.

Listing 1: Tunable Constants

```
// ======================== Sensors & Pins ========================
#define TRIGGER_PIN A0
#define ECHO_PIN A1
#define max_distance 50

// IR sensors
#define irLeft A3
#define irRight A2

// ======================== Avoidance Tunables ====================
const int SERVO_FORWARD = 53; // <--- TUNE THIS
const int OBSTACLE_CM = 15; // trigger distance
const int AVOID_BACK_MS = 400; // back off time
const int AVOID_TURN_MS = 500; // pivot time
const int AVOID_FORWARD_MS = 1300; // drive forward
const int REACQ_TURN_MS = 1100; // pivot to re-approach line
```

The Main `loop()` Function

The `loop()` function contains the primary decision-making logic. The robot first checks for obstacles using the `objectAvoid()` function, which will take control if an obstacle is detected. If the path is clear, it follows the line based on the readings from the IR sensors.

Listing 2: The main loop() function

```
void loop() {
  if (readLeftIR() == 0 && readRightIR() == 0) {
    objectAvoid();
  }
  else if (readLeftIR() == 0 && readRightIR() == 1) {
    objectAvoid();
    moveLeft();
  }
  else if (readLeftIR() == 1 && readRightIR() == 0) {
    objectAvoid();
    moveRight();
  }
  else if (readLeftIR() == 1 && readRightIR() == 1) {
    Stop();
    lookForward();
  }
}
```

The `objectAvoid()` Function

This function is the core of the obstacle avoidance routine. It checks the distance to an object, and if it's within the threshold, it stops the robot, scans the area, and executes the pre-programmed avoidance maneuver.

Listing 3: The objectAvoid() function

```
void objectAvoid() {
  lookForward(); // start with the sensor forward

  int d = getDistance();
  if (d <= OBSTACLE_CM) {
    Stop();
    int L = lookLeft();
    int R = lookRight();
    delay(100);

    bool preferLeft = (L >= R); // choose the freer side
    avoidManeuver(preferLeft);

    lookForward();
  } else {
    // Clear path => go forward and keep sensor forward
    lookForward();
    moveForward();
  }
}
```