Project Milestone 05:Final Report

Dhanani School of Science and Engineering

Microcontrollers & Interfacing

EE 375L-T1/T2

Course Name: Microcontrollers and Interfacing

E-375L-T1/T2

Instructor's Name: Miss Abeera Farooq Alam

Group Members (Names & IDs):

Bushra Sadaf - bs09267

Saira Talha - st09211

Javeria Nadeem - jn09271

Abeeha Hasan - ah09187

Submission Date:23rd April, 2025

Table of Contents

1. Introduction:

This report presents the final outcomes of our autonomous wall-following robotic car project, marking the successful completion of all design, implementation, and testing milestones. The primary objective of this project was to develop a self-navigating robot capable of traversing a predefined arena while executing three precise 90-degree turns without colliding with the walls. Through systematic hardware-software integration, modular code development, and effective use of version control tools, we achieved a fully functional robotic system that meets all the performance criteria. Key features of this robot include:

- Bluetooth-based start and stop commands, allowing remote operation
- Straight line motion
- 3 90(degrees) turn
- Autonomous Sensor Detection

This report outlines the design rationale, control strategies, subsystem integration, and the testing phases that led to the final operational robot.

2. System Overview:

Our robot is a wall following car that can drive on its own and follow walls without bumping into them. It was made using both hardware (like sensors and motors) and software.

The car moves by itself based on what the sensors tell it, and we can also start or stop it using a Bluetooth remote from our phone. We used a microcontroller Tiva C LaunchPad to control everything, and we programmed it using Energia.

At first, we tried a complex control method (PID), but it didn't work well for our setup. So we used a simple method based on "if-else" rules—for example, if something is too close on the left, turn right.

There are three ultrasonic sensors (one in front, one on the left, and one on the right) that check for nearby walls or obstacles. Based on these, the robot decides how to move.

We built the system in separate parts—sensors, motors, Bluetooth, and LED—so each part can be tested and fixed easily without affecting the others. This helped us build the robot step by step and made fixing problems easier.

3. Robot Design and Architecture:

The mechanical structure of the robot is a compact and lightweight chassis, designed to hold all essential components securely. The chassis includes compartments and mountings for:

- Tiva C LaunchPad
- L298N driver module
- Three ultrasonic sensors (front-facing, left-facing, and right-facing)
- Two geared DC motors for differential drive
- Bluetooth module (HC-05) for remote communication
- Battery pack (providing regulated power to motors and sensors)

Electrical Design:

- Motor Driver (L298N):

  Controls the direction and speed of the left and right motors independently. It receives PWM signals from the microcontroller and translates them into appropriate motor actions.

- Ultrasonic Sensors (HC-SR04):

  Positioned at the front, left, and right to provide obstacle distance data.

- Bluetooth Module (HC-05):

  Enables wireless command reception. 'S' command starts the car and 'N' command stops it

Motion System:

The robot employs a differential drive system using two DC motors connected to the L298N motor driver. The drive system enables:

- Straight line motion by activating both motors simultaneously.
- In-place turns (90°) by rotating one motor while halting the other.

Sensor Placement:

- Front Sensor: Directly faces forward to detect oncoming obstacles.
- Left and Right Sensors: Positioned at left and right to monitor distances from side walls, enabling accurate wall-following behavior.

All the parts were placed in a way that keeps the robot balanced, reduces shaking, and lets the sensors work properly. We tested the setup many times to make sure the robot could move well and detect obstacles easily.

4. Software Implementation:

1. Core Components

Main Loop

The program runs in a continuous loop, constantly checking for Bluetooth commands and sensor data. When a Bluetooth command is received via the Serial.read() function, it's processed in the processBluetooth() function. This controls the robot's movement based on commands ('S' for start and 'N' for stop).

Sensor Interface (getUltrasonicDistance)

The robot uses three ultrasonic sensors (front, left, right) to detect obstacles. The getUltrasonicDistance() function triggers each sensor, measures the pulse duration, and converts it to a distance. These distances are stored in cmFront, cmLeft, and cmRight for navigation decisions.

Motion Control (moveForward, stopMotors, turnLeft, turnRight)

The robot's movement is controlled via an L298N motor driver:

1)moveForward(): Moves the robot forward by activating both motors.

2)stopMotors(): Stops the motors, halting the robot.

3)turnLeft() and **turnRight()**: Turn the robot left or right by controlling individual motors. They are used for sharp 90-degree turns:

- turnLeft() moves the left motor forward while stopping the right motor.
- turnRight() moves the right motor forward while stopping the left motor.

Each function uses a fixed delay (typically ~550 ms) to execute a clean turn in place.

4)turnLeftAdjust() and turnRightAdjust():

These are **minor alignment corrections** triggered when the robot veers too close to a wall or edge:

- turnLeftAdjust() turns the left motor on briefly to nudge right.
- turnRightAdjust() turns the right motor on briefly to nudge left.

They use short, timed pulses (~100 ms) for subtle adjustments without affecting the overall navigation path.

2. State Management

State Variable (carRunning)

The carRunning variable is used as a flag to control whether the robot is actively running or stopped. This flag is set via Bluetooth commands and determines whether the robot should engage in autonomous navigation or remain stationary.

Non-blocking Design

To ensure that the robot remains responsive to Bluetooth commands even when navigating autonomously, the software avoids long delay() calls in the main loop. The only delays in the loop are within the turning functions, where the robot must pause momentarily to complete a turn.

This approach ensures that the robot continuously polls for Bluetooth commands while it is running, so it can start or stop immediately upon receiving input.

3. Timing & Performance

The sensor polling occurs every loop iteration, typically at a frequency of 5 to 10 milliseconds. This allows the robot to react quickly to changes in the environment, such as obstacles or sudden changes in direction.

The motors are controlled using PWM signals, with duty cycles adjusted for different speeds. For example, the left motor runs at 191 and the right motor at 200 to maintain a balance in speed. These values can be easily adjusted to fine-tune the robot's performance in accordance with the voltage the battery provides.

4. Error Handling & Safety

The robot ignores invalid sensor readings (e.g., zero or out-of-range distances) and defaults to stopping the motors if it encounters an error as you can see here:

```
    if (cmFront <= 9 && cmFront > 0) {
       if (cmLeft > cmRight) {
          turnLeft();
       } else {
          turnRight();
       }
     } else if (cmLeft <= 4 && cmLeft > 0) {
       turnRightAdjust();
     } else if (cmRight <= 4 && cmRight > 0) {
       turnLeftAdjust();
     } else {
       moveForward();
     }
   } else {
     stopMotors();
   }
}
```

This ensures safe operation, preventing the robot from running into obstacles or performing erratic movements.

5. Autonomous Navigation and Turn execution:

Initially, we implemented a **PID control-based wall-following system** that aimed to maintain a constant distance (10 cm ± 2 cm) from the both the walls of the arena. The goal was to enable smooth and dynamic correction of the robot's path using real-time feedback from ultrasonic sensors.

However, despite the theoretical benefits of PID (such as smoother motion and better precision), it **didn't perform as expected in practice**. The reasons included:

- Inconsistent ultrasonic readings due to environmental noise or surface reflectivity.
- PID tuning (Kp, Ki, Kd) was highly sensitive and minor fluctuations in sensor readings led to frequent overcorrections leading to wobbly car movement.
- The robot sometimes oscillated or behaved unpredictably in narrow or complex environments.

```
// PID Constants (tunable)
float Kp = 10.0;
float Ki = 0.5;
float Kd = 2.0;

float previousError = 0;
float integral = 0;

const int rightWallSetpoint = 10; // desired right distance (cm)
const int errorMargin = 2;        // acceptable error range
const int frontMinDistance = 9;   // stop if too close in front
```

```
unsigned long lastTime = 0;

// Motor Pins
int enA = 40; int in1 = 12; int in2 = 10;
int enB = 37; int in3 = 13; int in4 = 27;

// Ultrasonic Sensor Pins
const int frontTrig = 2, frontEcho = 24;
const int leftTrig = 30, leftEcho = 4;
const int rightTrig = 26, rightEcho = 19;

long cmFront, cmLeft, cmRight;

void setup() {
   Serial.begin(9600);
   pinMode(enA, OUTPUT); pinMode(in1, OUTPUT); pinMode(in2, OUTPUT);
   pinMode(enB, OUTPUT); pinMode(in3, OUTPUT); pinMode(in4, OUTPUT);
   pinMode(frontTrig, OUTPUT); pinMode(frontEcho, INPUT);
   pinMode(leftTrig, OUTPUT); pinMode(leftEcho, INPUT);
   pinMode(rightTrig, OUTPUT); pinMode(rightEcho, INPUT);
}

void loop() {
   cmFront = getUltrasonicDistance(frontTrig, frontEcho);
   cmLeft = getUltrasonicDistance(leftTrig, leftEcho);
   cmRight = getUltrasonicDistance(rightTrig, rightEcho);

   Serial.print("Front: "); Serial.print(cmFront); Serial.print(" cm, ");
   Serial.print("Left: "); Serial.print(cmLeft); Serial.print(" cm, ");
   Serial.print("Right: "); Serial.print(cmRight); Serial.println(" cm");

   if (cmFront > 0 && cmFront <= frontMinDistance) {
      stopMotors();
   } else {
      wallFollowWithPID();
   }
}

void wallFollowWithPID() {
   unsigned long currentTime = millis();
   float deltaTime = (currentTime - lastTime) / 1000.0;
   lastTime = currentTime;
```

```
    float error = rightWallSetpoint - cmRight;

    // Only correct if error exceeds margin
    if (abs(error) < errorMargin) error = 0;

    integral += error * deltaTime;
    float derivative = (error - previousError) / deltaTime;
    float correction = Kp * error + Ki * integral + Kd * derivative;
    previousError = error;

    int baseSpeed = 180;
    int leftSpeed = constrain(baseSpeed + correction, 100, 255);
    int rightSpeed = constrain(baseSpeed - correction, 100, 255);

    // Drive motors with adjusted speeds
    digitalWrite(in1, HIGH); digitalWrite(in2, LOW); analogWrite(enA, leftSpeed);
    digitalWrite(in3, HIGH); digitalWrite(in4, LOW); analogWrite(enB, rightSpeed);

    Serial.print("Correction: "); Serial.println(correction);
}

long getUltrasonicDistance(int trigPin, int echoPin) {
    digitalWrite(trigPin, LOW); delayMicroseconds(2);
    digitalWrite(trigPin, HIGH); delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    return (pulseIn(echoPin, HIGH) * 0.034) / 2;
}

void stopMotors() {
    Serial.println("Stopping Motors");
    digitalWrite(in1, LOW); digitalWrite(in2, LOW); analogWrite(enA, 0);
    digitalWrite(in3, LOW); digitalWrite(in4, LOW); analogWrite(enB, 0);
}
```

So, in order to improve reliability and control, we shifted to a **simplified rule-based navigation approach** using threshold values and discrete decisions:

- If an obstacle is detected in front (within 9 cm), the robot chooses to turn **left or right** based on which side is more open.
- If too close to the left wall (≤ 4 cm), it nudges right (by keeping a smaller delay in the turning time).
- If too close to the right wall (≤ 4 cm), it nudges left.
- Otherwise, it moves forward at fixed motor speeds.

This method proved to be:

- More stable and predictable in performance.
- Easier to debug and tune, with straightforward logic paths.
- Less sensitive to sensor noise, thanks to fixed thresholds and discrete actions.

Communication Protocol:
We implemented an autonomous operation logic so that our robot starts moving on 'S' command via Bluetooth and continues autonomous navigation using sensor data, and stops on ‘N’ command. Additionally, the robot uses an RGB LED to indicate its status: green for active and red for stopped. This setup enables smooth Bluetooth control, allowing the robot to start and stop with ease, while autonomously navigating and avoiding obstacles using sensor data for safe operation.
Here is the compiled **final code**:

```

// RGB LED setup
const int servoPin = 26; // PD3
const int red = 30, blue = 40, green = 39; // RGB LED Pins
char received = ' '; // Character received via UART
bool carRunning = false; // Flag to control car movement
```

```
// Motor A (Left Motor) - TivaC and L298N connections
int enA = 40, in1 = 12, in2 = 10;
// Motor B (Right Motor) - TivaC and L298N connections
int enB = 37, in3 = 13, in4 = 27;

// Ultrasonic Sensor Pins
const int frontTrig = 2, frontEcho = 24;
const int leftTrig = 30, leftEcho = 4;
const int rightTrig = 26, rightEcho = 19;
long cmFront, cmLeft, cmRight;

void setup() {
   Serial.begin(9600);
   pinMode(red, OUTPUT); pinMode(green, OUTPUT); pinMode(blue,
OUTPUT);
   analogWrite(red, 0); analogWrite(green, 0); analogWrite(blue, 0);
   myservo.attach(servoPin);
   pinMode(enA, OUTPUT); pinMode(in1, OUTPUT); pinMode(in2,
OUTPUT);
   pinMode(enB, OUTPUT); pinMode(in3, OUTPUT); pinMode(in4, OUTPUT);

   pinMode(frontTrig, OUTPUT); pinMode(frontEcho, INPUT);
   pinMode(leftTrig, OUTPUT); pinMode(leftEcho, INPUT);
   pinMode(rightTrig, OUTPUT); pinMode(rightEcho, INPUT);
}

void loop() {
   if (Serial.available() > 0) {
      received = Serial.read();
      Serial.println(received);
      processBluetooth(received);
   }

   if (carRunning) {
      cmFront = getUltrasonicDistance(frontTrig, frontEcho);
      cmLeft = getUltrasonicDistance(leftTrig, leftEcho);
      cmRight = getUltrasonicDistance(rightTrig, rightEcho);
```

```
        Serial.print("Front: "); Serial.print(cmFront); Serial.print(" cm, ");
        Serial.print("Left: "); Serial.print(cmLeft); Serial.print(" cm, ");
        Serial.print("Right: "); Serial.print(cmRight); Serial.println(" cm");

        if (cmFront <= 9 && cmFront > 0) {
            if (cmLeft > cmRight) {
                turnLeft();
            } else {
                turnRight();
            }
        } else if (cmLeft <= 4 && cmLeft > 0) {
            turnRightAdjust();
        } else if (cmRight <= 4 && cmRight > 0) {
            turnLeftAdjust();
        } else {
            moveForward();
        }
    } else {
        stopMotors();
    }
}
void processBluetooth(char sign) {
    switch (sign) {
        case 'S': // Start car
            carRunning = true;
            analogWrite(red, 0); analogWrite(green, 255); analogWrite(blue, 0);
            break;
        case 'N': // Stop car
            carRunning = false;
            analogWrite(red, 255); analogWrite(green, 0); analogWrite(blue, 0);
            break;
        default:
            break;
    }
}

long getUltrasonicDistance(int trigPin, int echoPin) {
    digitalWrite(trigPin, LOW); delayMicroseconds(2);
    digitalWrite(trigPin, HIGH); delayMicroseconds(10);
```

```
  digitalWrite(trigPin, LOW);
  return (pulseIn(echoPin, HIGH) * 0.034) / 2;
}
void moveForward() {
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW);
  analogWrite(enA, 191);
  digitalWrite(in3, HIGH); digitalWrite(in4, LOW);
  analogWrite(enB, 200);
}
void stopMotors() {
  digitalWrite(in1, LOW); digitalWrite(in2, LOW);
  analogWrite(enA, 0);
  digitalWrite(in3, LOW); digitalWrite(in4, LOW);
  analogWrite(enB, 0);
}
void turnRight() {
  Serial.println("Turning right 90 Degrees");
  digitalWrite(in1, LOW); digitalWrite(in2, LOW); analogWrite(enA, 0);
  digitalWrite(in3, HIGH); digitalWrite(in4, LOW); analogWrite(enB, 200);
  delay(550);
}
void turnRightAdjust() {
  Serial.println("Turning right slightly");
  digitalWrite(in1, LOW); digitalWrite(in2, LOW); analogWrite(enA, 0);
  digitalWrite(in3, HIGH); digitalWrite(in4, LOW); analogWrite(enB, 200);
  delay(100);
}
void turnLeft() {
  Serial.println("Turning left 90 Degrees");
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW); analogWrite(enA, 191);
  digitalWrite(in3, LOW); digitalWrite(in4, LOW); analogWrite(enB, 0);
  delay(550);
}
void turnLeftAdjust() {
  Serial.println("Turning left slightly");
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW); analogWrite(enA, 191);
  digitalWrite(in3, LOW); digitalWrite(in4, LOW); analogWrite(enB, 0);
  delay(100);
}
```

6. Sensor Integration and Testing

To enable autonomous navigation and obstacle avoidance, our robot utilizes three ultrasonic distance sensors positioned at the front, left, and right sides of the chassis. These sensors continuously monitor the surroundings, providing real-time distance feedback to the microcontroller.

Each ultrasonic sensor calculates distance using the time-of-flight principle, with the following function in the code:

```
long getUltrasonicDistance(int trigPin, int echoPin) {
   digitalWrite(trigPin, LOW); delayMicroseconds(2);
   digitalWrite(trigPin, HIGH); delayMicroseconds(10);
   digitalWrite(trigPin, LOW);
   return (pulseIn(echoPin, HIGH) * 0.034) / 2;
}
```

This returns the distance in centimeters by measuring the duration of the echo pulse and applying the speed of sound conversion factor.

Threshold Calibration and Obstacle Avoidance

To define when an obstacle is considered "close," we empirically set a distance threshold of 15 cm:

- If cmFront ≤ 15 ->Obstacle in front
- If cmLeft ≤ 15 -> Obstacle on the left
- If cmRight ≤ 15 -> Obstacle on the right

These thresholds were determined through manual calibration and observation during test runs, ensuring reliable avoidance without false triggers from minor disturbances

```
      // Obstacle avoidance logic
      if (cmFront <= 15 && cmFront > 0) {
         stopMotors();
         if (cmLeft > cmRight) {
            turnLeft();
         } else {
            turnRight();
         }
      } else if (cmLeft <= 15 && cmLeft > 0) {
         turnRight();
      } else if (cmRight <= 15 && cmRight > 0) {
         turnLeft();
      } else {
         moveForward();
      }
   } else {
      stopMotors();
   }
}
```

7. Github Collaboration

https://github.com/Javeria666/mci_project

8. Results

The autonomous robotic car successfully met all design and performance objectives by the end of the project. The following outcomes were observed during testing and demonstration:

- **Accurate 90-Degree Turns**: The robot was able to complete three precise 90-degree turns without colliding with either wall, consistently navigating corners in the test arena.

- **Reliable Obstacle Avoidance**: The **ultrasonic sensor array** enabled the robot to maintain a safe distance from both walls at all times. The sensing mechanism proved responsive to dynamic changes in the environment.

- **Fully Integrated Subsystems**: All hardware components—including the motor drivers, sensors, RGB LED, and Bluetooth module—were successfully integrated with the software system. This ensured seamless operation and real-time decision-making.

- **Autonomous Operation**: Once activated via Bluetooth, the robot performed the entire navigation routine autonomously without any manual intervention. It responded to sensor inputs and adjusted its trajectory in real time.

- **Modular and Scalable Codebase**: The code was written in a modular fashion, making it easier to debug, modify, and scale. Each subsystem (e.g., motor control, Bluetooth handling, sensor logic) was developed in independent modules.

- **Version Control Usage**: GitHub was effectively used throughout development to manage changes, collaborate as a team, and maintain code history, which greatly helped in tracking progress and resolving merge conflicts.

9. Task Division

Abeeha Hasan – Sensor Module testing, Mechanical design, chassis design, component placement,Bluetooth configuration, Arena Volunteer, bluetooth testing, report formatting, Sensor module implementation, serial communication implementation , report formatting

Javeria Nadeem –Programming approach, firmware development and initialization, car motion algorithms, Sensor implementation, Turning logic, power management and battery testing, testing and calibration, report formatting

Bushra Sadaf – Programming approach, firmware architecture, car motion debugging, Sensor implementation, device drivers(motor drivers configuration), Mobility and motor control, power management and battery testing, report formatting

Saira Talha –Sensor module implementation, Sensor configuration, device drivers(sensors drivers configuration), Bluetooth implementation, Sensor selection, placement, tuning, hardware components, serial communication implementation , report formatting

10. Conclusion

This final milestone marks the successful completion of our autonomous wall -following robot project. The robot effectively integrates ultrasonic sensors for real-time obstacle avoidance, Bluetooth communication for wireless control, and an RGB LED for visual feedback. Through precise calibration and systematic development, the robot consistently demonstrated smooth navigation and reliable performance. This project not only met its functional goals but also significantly enhanced our skills in sensor interfacing, and autonomous robotics. The experience lays a strong foundation for us for future innovations in robotic systems.