Princess Sumaya University for Technology
King Abdullah II Faculty of Engineering
Electrical Engineering Department

Embedded Systems
Autonomous Mobile Robot Design

Authors:

Baraa Ishtaiwe	20210141
Baraa Abedalhadi	20220148
Harith Alfazza	20220807

Supervisor:
Dr. Esam Qaralleh

JANUARY 14, 2026

Abstract

This report details the design and implementation of an autonomous mobile robot capable of navigating a multi-stage track. The system is built around the PIC16F877A microcontroller. The robot is designed to perform line following, light intensity detection (tunnel navigation), line following, obstacle avoidance, and precision parking. The project emphasizes low-level embedded programming by avoiding built-in compiler libraries for delays and bit manipulation, relying instead on direct register access and hardware timers.

Introduction and Background

In this project, we took on the "Autonomous Mobile Robots Challenge," which requires designing a vehicle capable of navigating a complex, multi-stage track without human intervention. The course is not a simple line-follower; it tests the robot's ability to adapt to changing environments, moving from a standard track to a dark tunnel, and then to an obstacle avoidance stage where the guide line completely disappears, finishing in a parking challenge.

We built our system around the PIC16F877A microcontroller, but the real technical challenge was the strict software constraint: we were prohibited from using standard compiler libraries for delays or hardware interfacing. This meant we could not simply use pre-made commands; instead, we had to write low-level code to manipulate the system registers directly for every timer, motor pulse, and sensor reading. This report details how we integrated these raw software concepts with a differential-drive mechanical design to create a robot that can detect obstacles, sense light levels, and park itself with precision.

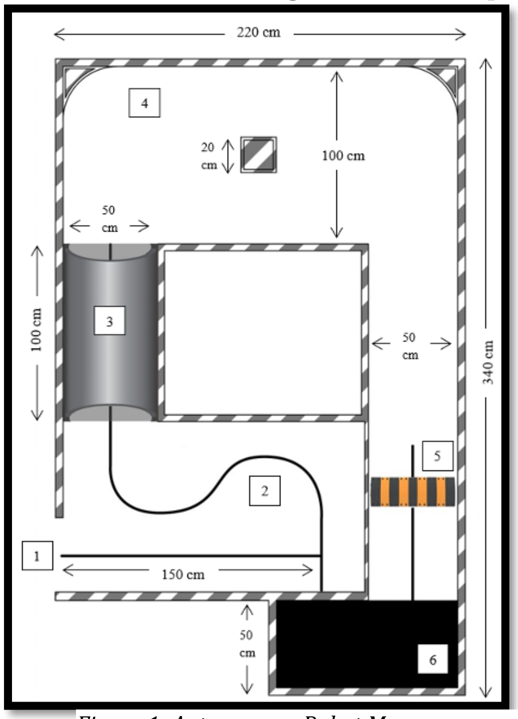

Figure 1: Autonomous Robot Map

System Design

Mechanical Design:

The physical robot was built to be compact, strictly adhering to the competition's dimensions to ensure it could fit through the tunnel and parking zones. We used a differential drive chassis, which controls direction by varying the speed of two independent DC motors. The sensor layout was critical to our success; as we used color sensor for the line following stages, and mounted the ultrasonic and IR sensors at the very front for the earliest possible obstacle and angle detection, while the servo motor responsible for raising the "Success Flag".

Electrical Design

Electrically, the system relies on the PIC16F877A as the central brain. We used a H-bridge to link the gap between the microcontroller's logic signals and the high-current needs of the motors, controlling their speed via Pulse Width Modulation (PWM) signals generated by the PIC's internal CCP modules. The sensor array includes three color sensors for tracking the floor lines, side-mounted IR sensors for following walls, and a Light Dependent Resistor (LDR) linked to the ADC module to detect when the robot enters the dark tunnel.

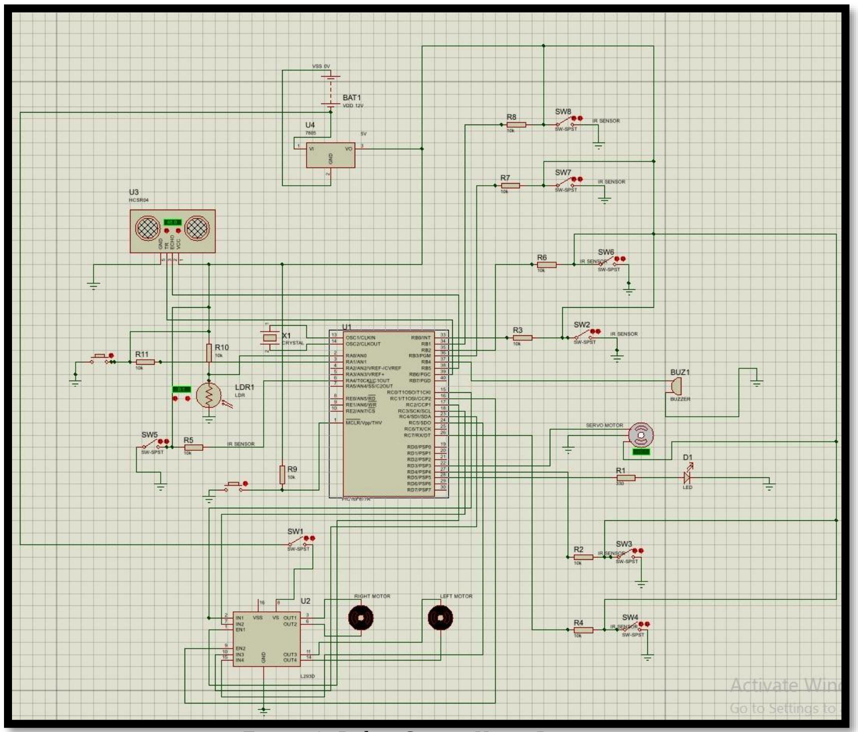

Figure 2: Robot Circuit Using Proteus

Software Design

The software is the core of the autonomous behavior, it was written in Embedded C using mikroC PRO for PIC. A strict design constraint was followed: no built-in libraries (like Delay_ms) were used. Instead, all delays were created using Timer0 interrupts and register calculations. The core architecture uses a State Machine approach implemented via sequential **while** loops in the **main()** function.

The code uses Timer Interrupts & Multitasking Instead of using blocking delays for everything, we implemented a Timer0 Interrupt Service Routine (ISR).

- Timer0 is configured to overflow every 1ms.
- Inside the ISR, a global variable **ms** is incremented.

- This counter allows us to create a "non-blocking" beeping effect during the tunnel stage (Buzzer is toggled as long as the LDR sees black), and parking stage (toggling the buzzer every 250ms) while the robot continues to drive and check sensors.

Control Logic Stages:

1. Initialization:

The robot holds for 3 seconds using **my_delay (3000)** before starting.

2. Line Following:

we have two cases for the line following stage, the first one is a straight line where the robot should follow the line at full speed, when both sensors read black it turns left indicating that it has passed the T-turn and a curved line is coming ahead, so lower speed will be used. The robot well keep the same speed until the ADC value exceeds 500 (darkness detected), it transitions to Tunnel Mode.

3. Tunnel Mode:

The robot maintains line following logic with higher speed to pass the tunnel in less than 3 seconds, it activates the Buzzer to signal entry and exits this mode when the LDR stops reading darkness.

4. Wall Following:

The robot switches to using the side IR sensors and Ultrasonic sensor. In this stage the robot tries to keep hugging the left wall within range, it keeps going left until dist_front < 30cm, the robot executes a hard right turn to avoid the corner. If it happens and the obstacle is in the robots way, the front ultrasonic sees the obstacle and gives an order of turning right to avoid hitting the obstacle, using this logic will allow us to always get through this stage and never get stuck or hit an obstacle.

5. Line following(with bump)

After completing the previous stage, the robot is asked again to look for the black line but this time a bump is there, so the speed of the robot should be enough to pass above it and reach the final stage.

6. Parking (Final Stage):

The robot enters the black floor area and detects it using three color sensors , the third sensor is used to distinct the final stage from the T-turn that the robot faces at first, once it reaches the black floor the robot slows down and the buzzer starts beeping, it keeps going forward and looks for the wall, when the wall is detected (dist_front< 15cm), the robot performs a short "Right Turn" adjustment, waits 250ms and moves forward again to ensure the robot does not stop at an angle, once the final distance is confirmed, the motors stop, and the servo pulses are generated to raise the flag.

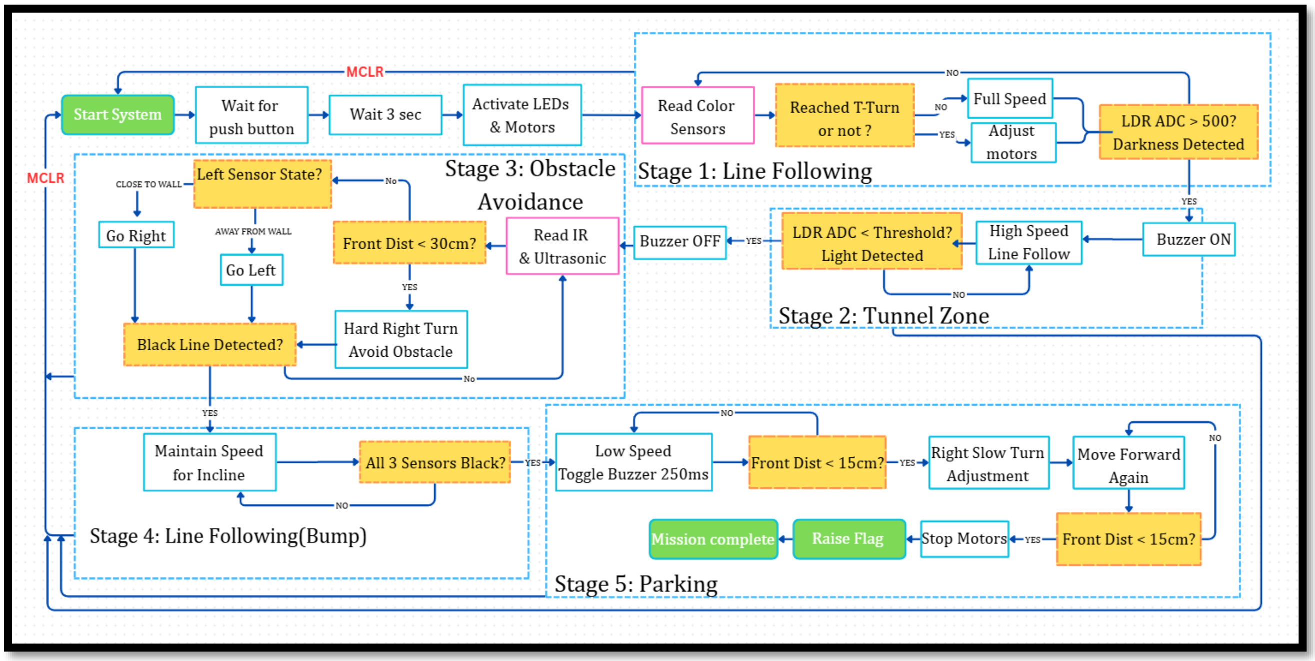

Figure 3: Logical Flow Chart

Problems and Recommendations

At the time we built the robot and wrote the code for each stage we faced multiple challenges, from small ones like a loose jumper to complicated problems like building the robot and

Problem 1: Parking Alignment

Issue: During testing, the robot often approached the parking wall at an angle, causing one sensor to trigger early while the robot was still crooked.

Solution: We added a specific logic block in the final code: if (dist_front < 15){ right_slow(); ... }. This forces a small re-alignment turn before the final stop, ensuring the robot parks straight.

Problem 2: Buzzer Timing

Issue: Using a standard delay for the beeping sound caused the motors to "stutter" or stop because the processor was stuck waiting for the delay to finish.

Solution: We moved the timing logic to the Timer0 interrupt. The main loop checks the ms counter to toggle the buzzer bit (PORTB |= 0x10) without stopping the motor control loop.

Recommendation: Currently, the steering uses "bang-bang" control (Hard Left/Hard Right). Implementing a PID controller for the motors would allow for smoother line following and wall tracking, reducing the oscillation seen on straight paths.

Conclusion

This project successfully demonstrated how low-level embedded systems concepts translate into a functional physical design. By bypassing standard libraries and interfacing directly with the PIC16F877A's registers for ADC, PWM, and Interrupts, we gained a much deeper understanding of how microcontrollers function at the hardware level. The final robot is capable of autonomously recognizing and reacting to its environment, successfully transitioning between line following, tunnel navigation, and obstacle avoidance to complete the challenge objectives.

GitHub Link: https://github.com/baraaabuzayed/Embedded-Systems-Project.git