

Program:	Electrical/Computer Engineering
----------	--

Course Number	COE 538
Course Title	Microprocessor Systems
Semester/Year	Fall 2025
Instructor	Dr. Vadim Geurkov

Report Title	Final Project - Robot Guidance Challenge
---------------------	--

Submission Date	December 1, 2025
Due Date	December 1, 2025

Name	Student ID	Signature*
Sweety Philip	501212883	STP
Sukhmanjot Aulakh	501161279	<i>S. Singh</i>
Gayarube Kiritharan	501244655	<i>G.K</i>

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: <http://www.torontomu.ca/senate/policies/pol60.pdf>.*

1. Project Description

This project implements an autonomous line-following and maze-running robot using the Freescale MC9S12 microcontroller. The robot uses five IR photo resistor sensors and two bumper switches to follow a black line on a white floor, navigate junctions and 90-degree corners, and recover from collisions by reversing and turning away from obstacles. All behavior is coordinated through a finite state machine that selects between forward motion, turning, alignment, and stop states based on live sensor readings.

• Initialization Routines

These routines configure the microcontroller peripherals at startup, including setting the data direction registers for motor and LCD pins, enabling and configuring the ADC module, initializing the LCD in 4-bit mode, clearing the LCD text buffers, and enabling the timer overflow interrupt that drives the TOF counter.

• Line-Following Logic

This block continuously reads the IR sensors and compares them against calibrated baseline and variance values to determine whether the robot is centered on the line, drifting left or right, or encountering a side branch. It uses the BOW, MID, and LINE sensors together to decide when to perform small corrective alignment pivots versus when to continue straight.

• State Machine

The state machine is implemented around the CRNT_STATE variable and a dispatcher subroutine. Each state (START, FWD, LEFT_TURN, RIGHT_TURN, REV_TURN, LEFT_ALIGN, RIGHT_ALIGN, ALL_STOP) has its own handler that encapsulates the behavior for that mode, and the dispatcher jumps to the appropriate handler each loop based on the current state value.

• Motor Control Subroutines

These routines abstract the low-level control of the drive motors. They set motor direction bits on PORTA for forward, reverse, left pivot, and right pivot, and control the enable bits on PORTT for on, off, and pulsed motion. The SLOW_FWD_STEP routine implements a timed duty cycle that produces smoother, slower movement for both forward motion and turning.

• Obstacle-Avoidance Behaviour

This block uses the front and rear bumper switches together with the line sensors to recover from collisions and dead ends. When the front bumper is triggered, the code updates the state to REV_TURN, reverses the robot for a fixed time, then commands a right pivot to search for the line again. The ALL_STOP state is entered when the rear bumper is pressed, shutting off the motors until the user reactivates the robot, providing a safe stop mechanism.

2. What the Robot Successfully Accomplished

- It was able to follow a taped line reliably using real-time ADC sensor values.
- It corrected drifts to both left and right using multi-level steering corrections.
- It reacted properly to obstacles by stopping and executing a reverse-and-turn manoeuvre.
- It reacquired the line after losing it by using a scanning/seek behavior.

Even though tuning required significant adjustment, the final version demonstrated stable behaviour and reliable transitions between states.

3. Main Problems Encountered & How They Were Solved

• Sensor Variability and Inconsistent Thresholds

When we first deployed the code on the eebots, we underestimated how sensitive the system was to the exact threshold values for the IR sensors. Poorly tuned thresholds led to sporadic motion, missed lines, and incorrect turns, which made the robot's behavior feel unpredictable and hard to debug. We addressed this by systematically logging sensor readings on both the white floor and black line, then selecting baseline and variance values that created a clear, stable separation between the two. This experience reinforced the importance of structured calibration and data-driven parameter tuning, which we would plan for earlier in future projects instead of treating it as a last-minute adjustment.

• Overturning During Collision Recovery

Our initial reverse and turn recovery sequence used delays that were too long, so after a collision, the robot would often overshoot and nearly spin in circles before it could reacquire the line. This made the behavior look chaotic and wasted a lot of time in recovery. We fixed this by iteratively shortening the reverse and turn delays and testing after each change, until the robot performed a controlled reverse, a partial turn, and then smoothly reengaged the line. From a project management perspective, this highlighted how important it is to isolate time-sensitive behaviors into clearly tunable constants so they can be adjusted without rewriting logic.

• State Machine Getting Stuck

At higher speeds, the robot occasionally detected a turn only for a brief instant, then continued forward, lost the branch, and effectively became stuck in the forward state. This revealed a mismatch between how fast the robot moved and how often the sensors were sampled. To resolve this, we slowed the robot's effective speed using pulsed motor control and increased the sensor sampling responsiveness, which reduced the chance of missing a junction. This issue showed us that state machine design cannot be separated from timing and sampling considerations, and that in future projects we should treat speed, sensor rate, and state transitions as a coupled design problem rather than tuning them independently.

4. Insights Gained & What Should Be Done Differently Next Time

This project really highlighted how much our software depends on the hardware it runs on. Small changes in battery level made the eebot behave unpredictably, to the point where code that previously worked on the same robot suddenly failed, which forced us to think in terms of hardware constraints rather than assuming ideal conditions. Next time, we would build battery monitoring and re-calibration checks into our workflow, and test behaviours at different battery levels instead of only when fully charged. We also saw how a clean, well-structured state machine and centralized parameters (thresholds, delays, speeds) made it much easier to debug and retune the system as the hardware drifted. In future projects, we would start with that structure from day one and design our tests specifically to stress the edge cases created by imperfect hardware.

Conclusion

Overall, the project successfully demonstrated a functioning autonomous robot capable of following a line, avoiding obstacles, and recovering after losing the path. The process required heavy debugging, calibration, and careful state-based logic design, but it provided valuable experience in low-level programming, real-time control, and structured project organization.