Toronto Metropolitan University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Microprocessor Systems
Course Number:	COE538
Semester/Year (e.g.F2016)	F2023

Instructor:	Samaneh Yazdani Pour

Assignment/Lab Number:	Lab Project
Assignment/Lab Title:	Robot Guidance Challenge

Section No.	9
Submission Date	12/4/23
Due Date	12/4/23

Student Name	Student ID	Signature*
Lathika Soori	500904706	L.S
Hamza Hashmi	501093570	H.H.
Ian Naunheimer	501102587	I.N.

**By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab repor 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:* www.ryerson.ca/senate/current/pol60.pdf.

Table of Contents

Summary

The COE538 term final project ties together all of the content learned in previous labs for the Robot Guidance Challenge, where teams were tasked with programming the eebot robot [1] in the assembly language with a basic artificial intelligence to explore and solve a maze. The robot is required to stay on top of the black electrical tape using it's sensor subsystem, and make decisions at each intersection, noting correct paths and managing the associated dead end of an incorrect by reversing and taking the alternate path instead. The foremost goal of the project is to showcase the robot's capability to reach the destination point at the end of a maze with a random design created by facilitators on the demo day.

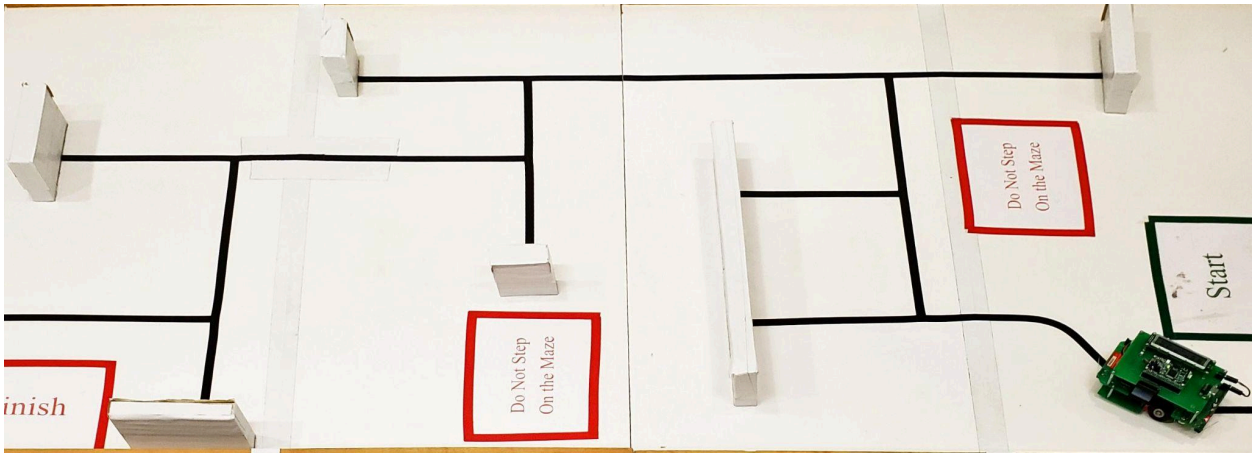

Figure 1: Maze on Demo Day

Objective

The COE538 Project [2] focuses on developing an efficient navigation system tailored for the mobile robot eebot, specifically designed for maze traversal. The robot is tasked with adeptly navigating the maze, which includes the following obstacles: S-turns, intersections, and dead ends.

Description of Programmed Behaviour

The robot moves forward with the use of two motors on its port and starboard sides. With each incremental movement forward, it scans the ground for the guide ensuring it is centred over the line.

The robot may at any time encounter an “s-turn”, which is a curving path requiring the robot to have differing displacements on the left and right motors to stay on track. The robot accomplishes this by incrementally moving forward while simultaneously reading its sensors, making small adjustments in order to stay on top of the black guide path.

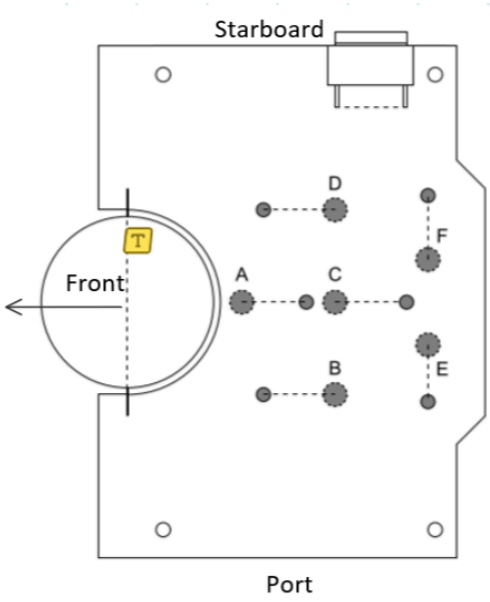

Figure 2: Diagram of eebot’s sensors and direction of Movement. Adapted from [3, Figure 1]

The robot detects intersections using its sensors D, A and B for the right, forward and left paths respectively. The sensors can be seen in Figure 2 above. Note that the challenge rules indicate a maximum of 3 way-intersections, hence there only being two outgoing paths- one correct and one incorrect. The order in which the robot explores each path is right, forward and finally left.

In the event the robot encounters a dead end, the front bumper will be activated, initiating the robot’s 180-degree turn, making note that the path it had taken was incorrect. Once back at the intersection, it proceeds to explore the remaining route, marking that remaining path as the correct one. If the robot does not encounter a dead end, the robot will recognize at the next intersection that it has taken the correct path and stores that information.

Note that the challenge rules indicate that for any maze there will be a maximum of 7 intersections [2, p. 3] with 4 possible directions, so the robot stores this correct path information in a fixed 7 by 4 data structure.

The ultimate objective is for the robot to reach the maze's endpoint, and when it has done so, the operator taps the back bumper, halting the movement of the eebot. An optional bonus feature that could have been implemented would begin at this point, in which the robot uses the stored correct path information to traverse the maze backwards, however to focus on the base functionality, this implementation had omitted this extra feature.

Difficulties and Obstacles

Numerous challenges and issues were encountered during the COE538 Project lab. In particular, the development of the eebot's navigation system for maze traversal proved to be a complex endeavour.

Decision-Making Algorithm Complexity:

Designing an effective decision-making algorithm for the robot at junctions presented difficulties. Striking a balance between enabling the robot to make optimal decisions and minimising computational complexity proved to be a challenging task.

Sensor Scanner Subsystem Computational Cost:

Initially, leveraging the light sensors to keep the eebot on track posed a significant hurdle. The high computational expense of the sensor subsystem introduced a higher than desirable delay whenever the sensor was taking readings. During this period the robot's previous state would be prolonged and this proved to be an issue when navigating S-turns as the robot is required to make very slight adjustments and therefore needs to be free to quickly and repeatedly update its motor state. Solving this challenge first, if possible, would likely have made the task of completing S-turns far more reliable, specifically when a high battery voltage causes slight path adjustments to be more difficult to execute, as discussed in the next paragraph.

Overshooting S-turns & the effect of the Battery Voltage:

The behaviour of the robot in the S curve consists of oscillating between slight left and right turns. This caused a problem during demonstration which had not been encountered in the numerous successful tests prior to demo week, when the batteries of the eebots were not fully charged. The increased voltage of the batteries on demo day caused the distance in which the eebot was deflected for each intended minor correction to be far greater. The overshooting led to increased compensation, culminating with the eebot being deflected off the track entirely. As a result, the eebot unexpectedly ran an intersection subroutine believing that the invalid line configuration it was reading as a result of being off the path was an intersection.

Difficulty in Dead-End Detection and Handling:

Accurately identifying dead ends and establishing a robust mechanism for the robot to execute a 180-degree turn proved to be challenging. Ensuring the robot could consistently retrace its steps and select an alternative path posed its own set of difficulties.

Testing and Validation:

Comprehensive testing and validation of the navigation system across various labyrinth conditions were deemed necessary but time-consuming. Rigorous testing was essential to confirm that the robot consistently adhered to the correct course, made sound decisions, and effectively managed dead ends.

Code Description

Equates Segment and Working Storage Area

Lines 9-16 encompass variables and definitions, specifying thresholds for sensors to detect the black line, state values, and binary indicators reflecting whether the sensors identify the black line (1 indicating detection, 0 indicating no detection).

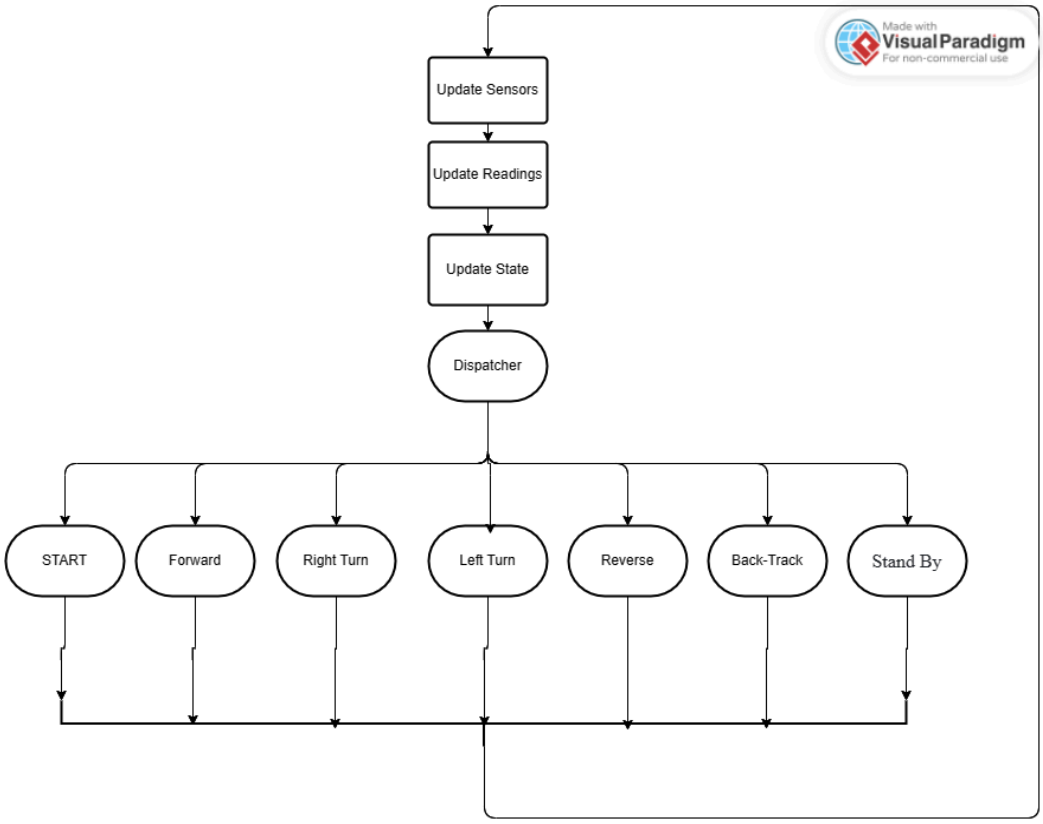

Figure 4: Visual Diagram of main loop which uses dispatcher method.

Initialization and Main Loop

For the implementation, the dispatcher approach was used rather than a simple state machine. A diagram of the dispatcher method can be seen in Figure 4 above, which is a diagram of the main loop: lines 138 to 143.

Subroutines Section

The section from lines 185 to 227 constitutes the dispatcher subroutine itself, enabling the processor to transition through states based on its current state and the conditions leading to the next state.

In the range of lines 232-494, various states of the robot are detailed, along with instructions for each state and the conditions for transitioning to the subsequent state. For instance, in the forward state, it checks for a front bumper press and transitions to the reverse state if detected. Otherwise, it branches to the no forward bump state, which further checks for a back bumper press. In the absence of a back bumper press, it proceeds to the no back bump state, inspecting black line detection on various sensors and adjusting its position accordingly. At intersections, if a right turn is detected, it prioritises that direction; otherwise, it proceeds straight. If the forward bumper is pressed, a 180-degree turn is executed, followed by forward movement until the next intersection, repeating the decision-making process.

Lines 496-536 initiate different states by controlling motor states (on/off, reverse/forward).

Lines 539-622 leverage the provided guider API to assess sensor readings and update corresponding variables.

Lines 626-803 house useful subroutines, including putcLCD and putsLCD, facilitating the output of individual characters and strings to the LCD.

Lines 806-885 manage the LCD updates on the eebot, incorporating values from sensors, battery voltage, and the robot's state.

Lines 887-989 encompass additional utility subroutines, such as timer systems, port initialization, LCD initialization, LCD clearing, and LCD cursor positioning.

Lines 998 to 1020 are for the Timer systems subroutines which employ hardware counters and interrupts.

Lines 1021 to 1031 is the interrupt vector section which notably includes the vectors for the COUNT1 and COUNT2 integers used for the first and second interrupts respectively.

References

[1] P. Hiscocks, V. Geurkov, “eebot Technical Description”, TMU, Toronto, ON
[2] P. Hiscocks, V. Geurkov, “COE538 Microprocessor Systems Lab Project Robot Guidance Challenge”, TMU, Toronto, ON
[3] P. Hiscocks, V. Geurkov, “COE538 Microprocessor Systems The *eebot* Guider”, TMU, Toronto, ON