Toronto Metropolitan University

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

Faculty of Engineering, Architecture and Science
Program: Computer Engineering

Course Number	**COE538**
Course Title	**Microprocessor Systems**
Semester/Year	**F2024**
Instructor	**Dr. Sattar Hussain**
TA Name	**Samaneh Yazdani Pour**

COE538 Final Project

Report Title	**Robot Guidance Challenge**

Section No.	**10, 16**
Submission Date	**Nov 29, 2024**
Due Date	**Nov 29, 2024**

Name	Student ID	Signature*
Sayeed Ahmed	*500985882*	
Ashwin Sundaresan	*501159998*	
Aronno Das	*501170036*	

**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.ryerson.ca/senate/policies/pol60.pdf.*

Objective

This project aims to develop a program that works in tandem with previous subroutines created in this course that enables the EEbot mobile robot to navigate a maze using its sensors and front bumper. The EEbot has 6 sensors: A (Front Sensor), B (Port Sensor), C (Middle Sensor), D (Starboard Sensor) and E, F (Line Sensors). The EEbot uses data from the front bumper and sensors shining their light on the maze to determine which path it should take through a trial-and-error approach. By continuously processing the sensor inputs the EEbot can detect obstacles, changes in the pathway and any alignments to make to stay centered on the maze line.

Description

Dispatcher: The dispatcher routine simply implements a finite state machine. The FSM tells the EEbot what to do by checking current state values stored in register A. Each of the states in the FSM is directly related to a subroutine that the dispatcher calls to control the movements and actions of the EEbot on the maze when certain aspects have been met.

Movements: This section of code contains all the movement and action subroutines the EEbot uses. It enables the EEbot to realign itself to the path by centring its sensors, along with controlling its forward, reverse and turning movements. Each subroutine here is called in the dispatcher once certain aspects are met.

Timer Overflow: The EEbot's timer system operates in the background continuously by utilizing a counter. When the counter overflows, an interrupt increments the clock counter. A Prescaler is then used to extend the overflow duration, allowing for longer delays. The timer is implemented without interrupting the program because it compares the current clock counter to the sum of the clock counter and a specified delay. This in return allows for the EEbot to perform its actions for a set duration without turning off.

Sensors: The sensor section of the code is responsible for collecting data using the multiple analog sensors on the bottom of the EEbot, converting those analog values to digital values and then storing them for future processing. The 'READ_SENSORS' subroutine initializes the sensors to begin their reading process. Next, the 'SELECT_SENSOR' subroutine is used to select specific sensors. Next, an Analog-to-Digital conversion process begins using the ATD module. Once the conversion is finished the 8-bit digital output is read from the ADC data register and is stored in an array corresponding to the selected sensor. This process then repeats for al other sensors until all readings are collected.

List of Issues Encountered During the Development and Testing of EEBOT Maze Runner

1. Line Readers Not Fully Functional

The first issue encountered was that the robot was moving forward, and the back bumper was functioning correctly, but the line readers were not properly detecting the path. This issue arose because not all five line readers were functioning simultaneously. Upon investigation, it was discovered that the READ_SENSORS subroutine did not correctly cycle through all the sensors. Specifically, the loop controlling the sensor readings prematurely exited when the pointer reached SENSOR_STBD. The condition in RS_MAIN_LOOP:

```
CPX   #SENSOR_STBD
BEQ   RS_EXIT
```

caused the loop to terminate without reading the SENSOR_LINE. This prevented the robot from correctly aligning itself to the maze line, leading to erratic movements. After fixing this by extending the loop to include all five sensors, the robot could effectively detect and follow the line.

2. Failure to Turn at Intersections

The second issue was that the robot would not turn at intersections, instead continuing straight. This was traced to a logical error in the DISPATCHER routine that failed to prioritize turning states over the forward movement state (FWD). The condition to switch to turning states in VERIFY_FORWARD:

```
BRSET PORTAD0, $04, NO_FWD_BUMP
MOVB  #REV_TRN, CRNT_STATE
```

checked for bumps but did not include a check for sensor readings indicating an intersection. As a result, the robot missed the logic necessary to detect intersections. This was addressed by adding sensor checks in the forward state to detect intersections and properly transition to LEFT_TRN or RIGHT_TRN as needed.

3. Back Bumper Stopped Functioning After Turn Implementation

After implementing the logic for turning at intersections, a new issue emerged where the back bumper stopped working. This was due to an oversight in the INIT_REV subroutine:

```
BSET PORTA, %00000011 ; Set REV direction for both motors
BSET PTT, %00110000   ; Turn on the drive motors
```

While enabling reverse movement, the code failed to reset the back bumper sensor logic, which was inadvertently disabled during the intersection handling logic in FWD_ST. Specifically, the interaction between BRSET checks and PORTAD0 conditions interfered with the back bumper's functionality. This was deprioritized to focus on enabling turns, as completing the maze relied on successful intersection navigation. Future iterations of the code will need to balance both functionalities.

4. Inconsistent Maze Completion

Lastly, there were occasions where the same code resulted in the EEBOT failing to complete the maze. This inconsistency could be attributed to hardware limitations or environmental factors. For instance, prolonged testing might have caused sensor fatigue or thermal issues in the microcontroller. Another possibility is slight variations in sensor calibration or surface conditions leading to unpredictable behaviour. The lack of a robust error-handling mechanism in the code may have further exacerbated the problem. These issues highlight the importance of periodic maintenance and ensuring hardware reliability for consistent performance.

Insights on Managing a Major Software Project: EEBOT Maze Runner

Managing the EEBOT project taught me the value of planning, testing, and adaptability. In the future, I'd start with modular testing for each feature and use an iterative approach with clear milestones to catch issues early. Better documentation and version control, like clear commit messages, would make collaboration and debugging easier. The FSM design worked well, making the robot's behaviour easy to manage and debug. Hardware abstraction through routines like READ_SENSORS helped simplify updates, and team problem-solving was effective in fixing tricky issues like sensor logic errors. Some things didn't go as planned. Delaying integration testing led to compounded issues, like the back bumper failure after adding turning logic. Calibration was also a challenge, as sensor variability from lighting and surface changes made the robot inconsistent at times. Debugging sensor logic and integrating intersection turns with the back bumper were the hardest parts. The sensor-reading loop needed trial and error to fix, and new dependencies between states made debugging tricky. Random failures, possibly from hardware wear or environmental factors, added to the frustration. Next time, I'd use automated tests, test in smaller steps, and focus on better calibration. Adding error handling and fallback states would also improve reliability. These changes would make future projects more efficient and reliable.