COE538 Microprocessor Project Report: eebot Mobile Robot Navigation System

Jason Nguyen - 501087930
Thanh Khoa Nguyen - 501209517
Lab Section: 01

11th August 2024

Project Description

This project involved programming the eebot mobile robot to navigate through a maze, reverse, and find its way back out. The main objectives were:

- Implement line tracking for navigating S-turns
- Develop decision-making at junctions
- Implement error detection and correction for dead ends
- Create a learning system to remember correct paths
- Enable reverse traversal of the maze

Insights and Reflections

Project Approach

In our sensor-based navigation project, we collect data on the robot's surroundings using a variety of sensors (A through F). The algorithm constantly refreshes and interprets these readings in order to make navigation decisions. The robot's behavior is controlled by a state machine, which includes START, FORWARD, REVERSE, RIGHT TURN, LEFT TURN, and ALL STOP. This method provides for unambiguous transitions between distinct actions in response to sensor inputs and other situations.

The code is designed for real-time processing, thus it constantly updates and responds to sensor data, using interrupt handlers for timing-critical processes. We separated the code into subroutines for different purposes, such as motor control, sensor reading, and display update. This modular approach makes the code easier to maintain.

An LCD display provides user feedback by displaying the current status, sensor readings, and other important information, with a focus on debugging and interactivity. Furthermore, we directly alter hardware registers for activities like motor control and sensor reading, providing us with fine-grained control over the robot's behavior.

Effective Strategies

The robot's decision to turn left, right, or reverse is primarily based on sensor readings and the current state. THe table below captures the logic of bot's movement:

Decision Logic Table

Condition	Action	Location	Explanation
FWD BUMP	REVERSE TURN	FWD_ST	When the forward bumper is hit, the bot enters the reverse turn state, initializes reverse movement, waits, and then initiates a right turn.
REAR BUMP	ALL STOP	FWD_ST	If the rear bumper is hit, the bot enters the ALL_STOP state and stops all movement.
PORT > THRESHOLD	LEFT TURN	FWD_ST	If the port sensor reading exceeds the threshold, the bot initiates a partial left turn.
STBD > THRESHOLD	RIGHT TURN	FWD_ST	If the starboard sensor reading exceeds the threshold, the bot initiates a partial right turn.
LINE < THRESHOLD	LEFT ALIGN	FWD_ST	If the line sensor reading is below the threshold, the bot checks for left alignment.
LINE > THRESHOLD	RIGHT ALIGN	FWD_ST	If the line sensor reading is above the threshold, the bot checks for right alignment.

Condition	Action	Location	Explanation
BOW > THRESHOLD (LEFT)	FORWARD	LEFT	During a left turn, if the bow sensor reading exceeds the threshold, the bot finishes alignment and resumes forward movement.
BOW > THRESHOLD (RIGHT)	FORWARD	RIGHT	During a right turn, if the bow sensor reading exceeds the threshold, the bot finishes alignment and resumes forward movement.
BOW > THRESHOLD (REVERSE)	LEFT TURN, FORWARD	REV_TRN_ST	During a reverse turn, if the bow sensor reading exceeds the threshold, the bot initializes a left turn, then resumes forward movement.
START BUTTON	FORWARD	START_ST	When the start button is pressed, the bot initializes forward movement and enters the FWD state.

1 Challenges and Implementation

1.1 Challenges

The primary challenge in our project is accurately interpreting sensor data for effective navigation. We're addressing several key issues:

- **Sensor Calibration**: Choosing suitable threshold values is essential. Incorrect thresholds might cause misreading of the surroundings.
- **Environmental Variability**: Lighting and surface texture can influence sensor performance. Our bot must adjust to these changes.
- **Sensor Fusion**: Combining data from various sensors is complicated yet necessary for precise navigation.
- **Real-Time Processing**: Quick data processing and decision-making are necessary. Delays can lead to poor navigation or crashes.
- **Decision Logic**: Creating an algorithm that can handle several circumstances, including unexpected barriers, necessitates thorough design and testing.

During maze testing, we encountered an issue where our bot consistently missed turns and moved in the wrong direction. This indicates potential problems with our decision-making logic or sensor data interpretation.

1.2 Current Implementation

Our approach to addressing these challenges includes:

- **Threshold Detection**: We compare sensor readings to preset thresholds to identify significant changes. For example:

```
CHECK_A  LDAA  SENSOR_BOW
         CMPA  #PTH_A_INT
         BLO   CHECK_B
         INC   A_DETN
```

This code snippet checks if a sensor reading exceeds a certain threshold. If so, it's marked as detected.

- **Multiple Sensors**: We utilize sensors A through F to obtain a comprehensive view of the bot's surroundings. Integrating data from all these sources aims to facilitate more informed decision-making.
- **State-Based Decisions**: We've implemented a state machine with various states such as START, FWD, and REV. This enables our bot to make decisions based on both its current state and sensor inputs, enhancing its responsiveness to changing conditions.

By implementing these strategies, we aim to improve our bot's navigation capabilities and resolve the issues observed during testing. While it remains a work in progress, we're encouraged by the improvements we're seeing.

Conclusion

The challenge of sensor-based decision making is central to autonomous robotics. Our current implementation is solid, but there is room for improvement to create a more robust and adaptable system. Balancing algorithm complexity with real-time performance is key.