Lab 2
Random Wander, Obstacle Avoidance

Reading: Ch. 3 of text

(Demonstration due in class on Thursday)
(Code and Memo due in Moodle drop box by midnight on Sunday at midnight)

Read this entire lab procedure before coming to lab.

************************************************************

Purpose: An essential characteristic of an autonomous robot is the ability to navigate in an environment safely. The purpose of this lab is to develop random wander and obstacle avoidance behaviors for the CEENBoT. The design of your program should use subsumption architecture. Layer 0 of your control architecture will be the collide and run away behaviors to keep the robot from hitting obstacles. Layer 1 will be the random wander behavior which moves the robot a random distance and/or heading every n seconds.

Objectives: At the conclusion of this lab, the student should be able to:
- Acquire and use data from all of the robot’s range sensors
- Write random wander and obstacle avoidance behaviors on the CEENBoT
- Use modular programming to implement subsumption architecture on the CEENBoT
- Move the robot safely in an environment with obstacles

Equipment: CEENBoT platform, ‘324 v2.21.
AVR In-System Programmer (ISP)

Theory:
The CEENBoT is equipped with left and right non-contact bump sensors in the front wired to the TINY microprocessor. The robot also has 3 IR sensors wired directly into the ATmega 324 processor on analog inputs: PA3 (left), PA4 (right) and PA5 (front), PA7 (back). These connections are shown on Figure 1.

---

Figure 1: Range Sensors
The left and right contact switches are digital inputs and the state is transferred to the master MCU (ATMega 324) via the SPI serial interface. SPI is synchronous peripheral interface that requires four lines of communication. The master MCU requests the state of these sensors from the Tiny MCU and the Tiny sends the data back as a byte of data (8-bits). The function to get the data from the contact switch is ATTINY_get_IR_state(). The function to set the servo is ATTINY_set_RC_servos(). You will also need to use equation (1) to convert the byte sample to a voltage. Refer to the CEENBoT-API Programmer’s Reference Guide for detailed information regarding these functions.

\[\text{voltage} = \text{sample} \times \frac{5}{1024}\]  

(1)

LAB PROCEDURE

Part 1 – Range Sensors

Download the RangeSensors project from the Moodle course folder and use it to test the sensors. You may need to modify the code to suit your purposes such as changing the library directories or printing different values to the LCD. By pressing the first button, the robot will beep once and LCD will display the contact sensor and light sensor data. If you press the second button, the robot will beep twice and the LCD will display the infrared sensor data. If you press the third button, the robot will beep three times and the LCD will display the temperature data.

The Sharp GP2Y0D810Z0F contact sensor is a reflectance sensor that emits and detects infrared light. When an object is detected, the corresponding bit will be high or return a one. When an object is not detected, the bit will be low or zero. The contact sensor detects an object within 4 inches. The specifications state that the Sharp GPD120 sensors measure between 1.5” and 12”. The IR will actually return an analog value that is proportional to the distance to an object. As with all IR sensors; the ambient light, color, texture, material, and angle of incidence determine how much energy is returned and this as well as specular reflection may affect accurate measurements.

In order to determine the specific characteristics of the range sensors on your particular robot, you should take several measurements on each sensor to correlate the measured distance with the actual distance to an object. You will need to use this information in order to have consistency and reliability between the different measurement devices and acceptable robot performance. You should write your code to account for any discrepancies. The best way to display this information would be to use a data table and perform an error analysis (see Table 1). You should include this data table in the appendix of your lab submission and provide a summary of the results in the text of the memo.
Table 1: Sensor Calibration Data

<table>
<thead>
<tr>
<th>Distance (“)</th>
<th>Contact (left)</th>
<th>Contact (right)</th>
<th>IR (front)</th>
<th>IR (left)</th>
<th>IR (right)</th>
<th>IR (back)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Part 2 – Layer 0 (Obstacle Avoidance)

This is the first time you will begin to add modularity in your programs by incorporating behavior-based techniques. Behavior-based programming uses primitive behaviors as modules for control. Primitive behaviors are concerned with achieving or maintaining a single, time-extended goal. They take inputs from sensors (or other behaviors) and send outputs to actuators (or other behaviors). You will create your first two primitive behaviors: obstacle avoidance, and random wander. The obstacle avoidance behavior will use either collide or run away behavior. In the collide behavior, the robot will drive forward and stop when an obstacle is detected and continue moving forward when the object is removed. In the run away behavior, the robot will move forward and when an obstacle is detected, move away proportional to where the obstacle is felt (feel force). Your program should be modular and make it clear which behavior is active (collide, run away).

Now that you are familiar with the range sensors and how to move the robot, create an obstacle avoidance behavior. The obstacle avoidance abstract behavior includes a collide and run away primitive behaviors. For the collide behavior, robot would drive forward and if an object is detected within 3 - 6 inches of the range sensors, the robot should halt the forward drive motor. If the object is removed, the robot should continue to move forward. You can use the free running ability of the stepper motor to do this. Refer to the CEENBoT-API Programmer’s Reference Guide for detailed information regarding the stepper motor.

For the run away behavior, create a plot of the sensor readings and use the sum to create a repulsive vector to turn the robot away (i.e. feel force). Finally, the robot should turn by some angle proportional to the repulsive vector and continue moving (see Figure 2). There are two ways to illustrate the run away behavior, one is to have the robot sit in the middle of the floor and move away when an object gets within 3 to 6 inches and then stop. Alternately, the robot can move forward and move when an object gets within 3 to 6 inches and then continue moving forward.
To demonstrate the *collide* behavior, the robot should drive forward until it encounters an obstacle and stop without hitting it. Think of the collide behavior as the *aggressive kid* who comes close but then stops short from touching the object. To demonstrate the *run away* behavior, the robot should sit in the middle of the floor until an object gets close and then move the opposite direction to get away. Think of this behavior as the *shy kid* who does not want any object to get too close to him. It is possible to also show run away for an *aggressive kid* who moves forward and then moves away when an object gets close. It should be clear during the demonstration what type of behavior the robot is executing.

**Part 3 – Layer 1 (Random Wander)**

In random wander, the robot will move in a random pattern when no obstacles are present.

Create a random wander routine that the robot uses to explore the room. This can be done by generating a random number that represents the robot’s heading, steps, distance, or motor speed every n seconds. You have the flexibility of using any combination of these values to make the robot explore the environment. You may also need to include `<stdio.h>` and `<stdlib.h>` libraries and use the `rand() % n` command to generate a random number between 0 and n.

**Part 4 – Subsumption Architecture – Avoid Behavior**

In this section the overall the subsumption architecture to create smart wander will be implemented. This architecture is shown in Figure 3. The perceptual schemas are *feel force* and *collide*. The motor schemas are *run away* and *collide*. The primitive behaviors are run away and collide and the two together make the abstract behavior, *obstacle avoidance*. The second layer of the architecture is the *wander* module created in part 3. The wander module passes the heading the *avoid* module which combines the feel force and wander heading to determine the direction the robot should turn to move away from obstacle. Note that the power of subsumption
architecture is that output of the higher level subsumes the output from the lower level. The avoid module suppresses the output from runaway and replaces it to make the robot turn.

Now improve the random wander routine by integrating obstacle avoidance (collide and run away). The robot should wander randomly until an obstacle is encountered. The robot should RunAway from the obstacle and continue to wander. The robot’s heading from the Wander behavior should be modified based upon the force from the range sensors and then turn and move from the obstacle. The Avoid module in Layer 1 combines the FeelForce vector with the Wander vector. The Avoid module then subsumes the heading from the Run Away module and replaces it with the modified heading as input to the Turn module.

![Diagram of robot behavior](image)

**Figure 3: Avoid Behavior**

Your program should provide a method to get the robot ‘unstuck’ if it approaches any local minima points (i.e. oscillates between two obstacles). Your program should be as modular as possible with multiple subroutines and behaviors that will be integrated in subsequent programs. Devise a method to test and confirm that your program works correctly and present the results in the laboratory memo. Figure 4 provides sample motion for the fully integrated robot behaviors.
Demonstration:

During the demonstration, you will show each layer of the architecture separately. For layer 0, the robot should demonstrate shy kid and aggressive kid. Please review the lab procedure if you don’t recall what this means. For layer 1, the robot should turn at a random heading and move forward periodically. Finally, to demonstrate the complete architecture, the robot should wander and halt when it “collides” with an obstacle, or modify the heading when it encounters and obstacle and then run away. The robot should give some type of audible and/or visual signal when an obstacle is encountered. The buzzer, LCD and LEDs can be used to provide feedback on the robot’s state. You may use the three pushbuttons to start or stop the different layers on the robot to speed up the demonstration.

*Bring your robot fully charged to class on Thursday for the demonstration. Note that you always must re-flash the factory firmware and plug in the AC adapter in order for the robot to charge. Alternately, you can put the robot battery in the RC car battery charger. Note that this is a fast charger and will not last as long as the outlet charge.*

Program:

In subsequent weeks you will reuse this code thus your code should follow proper programming techniques such as detailed commenting and be as modular as possible where behaviors and reactive rules are separate functions.

Memo Guidelines:

Please use the following checklist to insure that your memo meets the basic guidelines.

> Format

- Begins with Date, To, From, Subject
- Font no larger than 12 point font
Spacing no larger than double space

Includes handwritten initials of both partners at the top of the memo next to the names

Written as a paragraph not bulleted list

No longer than three pages of text

Writing

Memo is organized in a logical order

Writing is direct, concise and to the point

Written in first person from lab partners

Correct grammar, no spelling errors

Content

Starts with a statement of purpose

Discusses the strategy or pseudocode for implementing the robot paths (may include a flow chart)

Discusses the tests and methods performed

States the results including error analysis

Shows data tables with error analysis and required plots or graphs

Answers all questions posed in the lab procedure

Clear statement of conclusions

Questions to Answer in the Memo:

1. What was the general plan you used to implement the random wander and obstacle avoidance behaviors?

2. How did you create a modular program and integrate the two layers into the overall program?

3. Did you use the contact and IR sensors to create redundant sensing on the robot’s front half.

4. How could you create a smart wander routine to entirely cover a room?

5. What kind of errors did you encounter with the obstacle avoidance behavior?

6. How could you improve the obstacle avoidance behavior?
7. Were there any obstacles that the robot could not detect?

8. Were there any situations when the range sensors did not give you reliable data?

9. How did you keep track of the robot’s states in the program?

10. Did the robot encounter any “stuck” situations? How did you account for those?

Grading Rubric:

The lab is worth a total of 30 points and is graded by the following rubric.

<table>
<thead>
<tr>
<th>Points</th>
<th>Demonstration</th>
<th>Code</th>
<th>Memo</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Excellent work, the robot performs exactly as required</td>
<td>Properly commented with a header and function comments, easy to follow with modular components</td>
<td>Follows all guidelines and answers all questions posed</td>
</tr>
<tr>
<td>7.5</td>
<td>Performs most of the functionality with minor failures</td>
<td>Partial comments and/or not modular with objects</td>
<td>Does not answer some questions and/or has spelling, grammatical, content errors</td>
</tr>
<tr>
<td>5</td>
<td>Performs some of the functionality but with major failures or parts missing</td>
<td>No comments, not modular, not easy to follow</td>
<td>Multiple grammatical, format, content, spelling errors, questions not answered</td>
</tr>
<tr>
<td>0</td>
<td>Meets none of the design specifications or not submitted</td>
<td>Not submitted</td>
<td>Not submitted</td>
</tr>
</tbody>
</table>

Submission Requirements:

You must submit you properly commented code as a zipped folder of the C file and the lab memo in a zipped folder by **11:59 pm on Sunday** to the Moodle Course Drop box. Your code should be modular with functions and classes in order to make it more readable. You should use the push buttons, buzzers and LCD to command the robot and indicate the robot state during program execution.