Lab 03

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* where 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. In addition, the student team will integrate the Avoid-Obstacle, Random Wander, and Go-To-Goal behaviors by blending.

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:

Last week, you created the first two primitive motion behaviors on the robot: *Go-To-Goal* and *Go-To-Angle*. It is important to safely navigate a cluttered environment so this week you will implement the *Avoid-Obstacle* and *Random Wander* behaviors. Each week, you will create more functionality and behaviors for your robot so you should design the system to be as modular as possible in order to re-use algorithms and codes and build on what you have done before. One way to do this is to use the subsumption architecture with layers where the most basic layer is motion control and obstacle avoidance.

The CEENBoT is equipped with left and right non-contact bump sensors in the front wired to the TINY microprocessor. The robot also has 4 IR sensors wired directly into the ATMega processor on analog inputs: PA3 (right), PA4 (left) and PA5 (back), PA7 (front). These connections are shown on Figure 1.

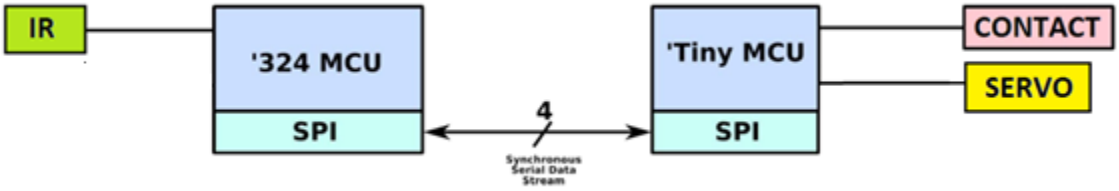


Figure 1:Range Sensors

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(). You will also need to use equation (1) to convert the byte sample for the IR sensors to a voltage. Refer to the *CEENBoT-API Programmer's Reference Guide* for detailed information regarding these functions.

$$voltage = sample * \frac{5}{1024} \quad (1)$$

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.

****

LAB PROCEDURE

****

Part 1 –Range Sensors

Download the *"Sensors.c"* program 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. Note that the right light sensor data will be wrong because it is not connected. It shares a port with the bark IR sensor which is currently connected on the CEENBot. 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. One of these values is the ambient temperature and won't change.

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

Distance (“)	Contact (left)	Contact (right)	IR (front)	IR (left)	IR (right)	IR (back)
1						
…						
15						

Part 2 – Avoid-Obstacle Behavior (Layer 0)

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 two more primitive behaviors: *avoid-obstacle*, 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). The robot's motion will be based upon the potential fields concept.

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). The robot should turn by some angle proportional to the repulsive vector and continue moving. 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 or the robot can start out moving and move away from an obstacle based upon the potential field (see Figure 2).

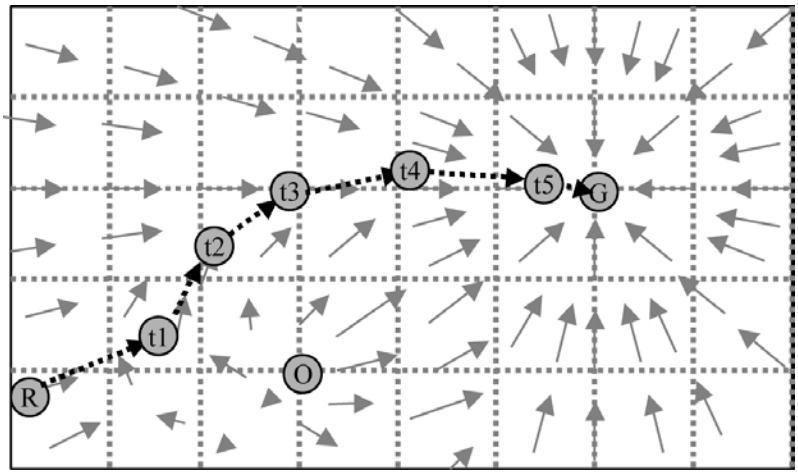


Figure 2: Potential Fields Method for Obstacle Avoidance

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. Figure 3 shows an example of the Avoid-Obstacle behaviors which will be layer 0 of the subsumption architecture.

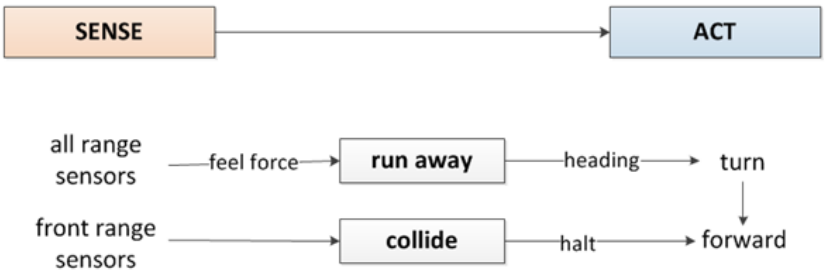


Figure 3: Level 0 – Obstacle Avoidance

Figure 4 provides an example of the robot's sample motion for the Avoid-Obstacle behavior.

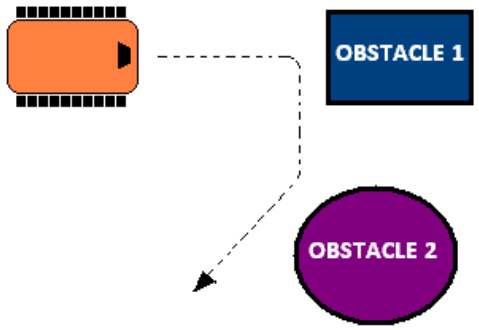


Figure 4: Subsumption Architecture – Sample Robot Motion

Part 3 – Random Wander (Layer 1)

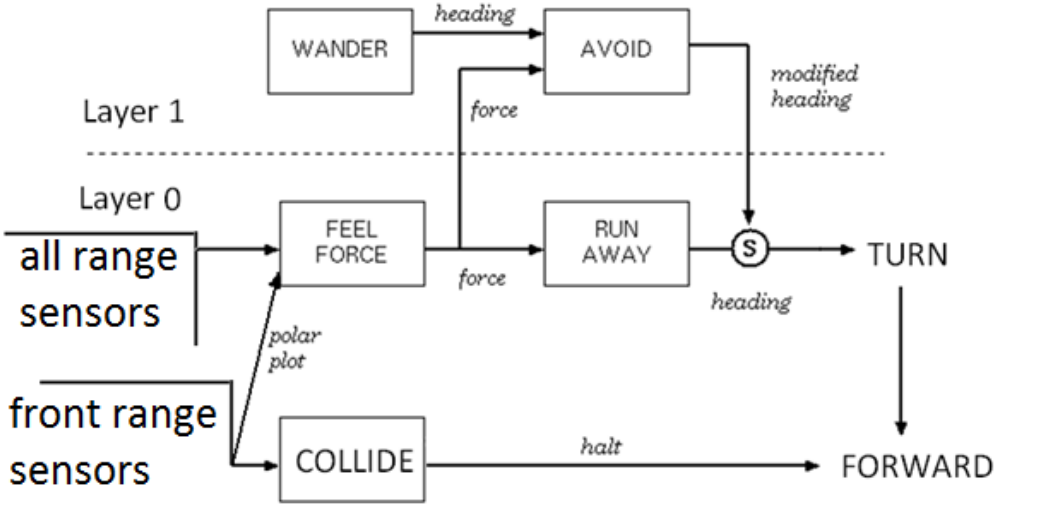
In a random wander behavior, the robot will move in a random pattern when no obstacles are present. Create a random wander routine that the robot uses to explore the room. This can be done by generating a random number that represents the robot's heading, steps, distance, or motor speed every n seconds. 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. To demonstrate this behavior, set the robot on the floor and execute the random motion.

Part 4 – Subsumption Architecture – Smart Wander Behavior (Layer 1)

In this section, you will use the subsumption architecture to create a *smart wander* behavior. This architecture is shown in Figure 5. The perceptual schemas are *feel force* and *collide* and 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 *run aways* 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. The power in this type of architecture is the flexibility the execution based upon the use of inhibition and suppression.



s

Figure 5: Smart Wander 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.

Part 5 – Integrate Avoid-Obstacle and Go-To-Goal Behaviors (Layer 2)

An alternate environment exploration technique would be to send the robot to a specific goal point or via points. The final step in the procedure will be to integrate the Avoid-Obstacle and Go-To-Goal Behaviors as shown in Figure 6. The robot should attempt to drive to a goal position and stop within a given error. If an obstacle is encountered, the robot should navigate around the obstacle until there is a direct line of sight to the goal position and then continue moving to the goal position. Note that this behavior would have to subsume the *random-wander* behavior.

It is necessary to use potential fields to solve this problem by finding the sum of the goal attraction vector and the obstacle repulsion vector. It is important to keep track of the robot's state and position as it avoids the obstacle in order to calculate the new vector to the goal and also to identify when to abandon the obstacle and make a straight path to goal. Figure 7 shows an example of the state machine that can be used to switch between avoid-obstacle and go-to-goal.

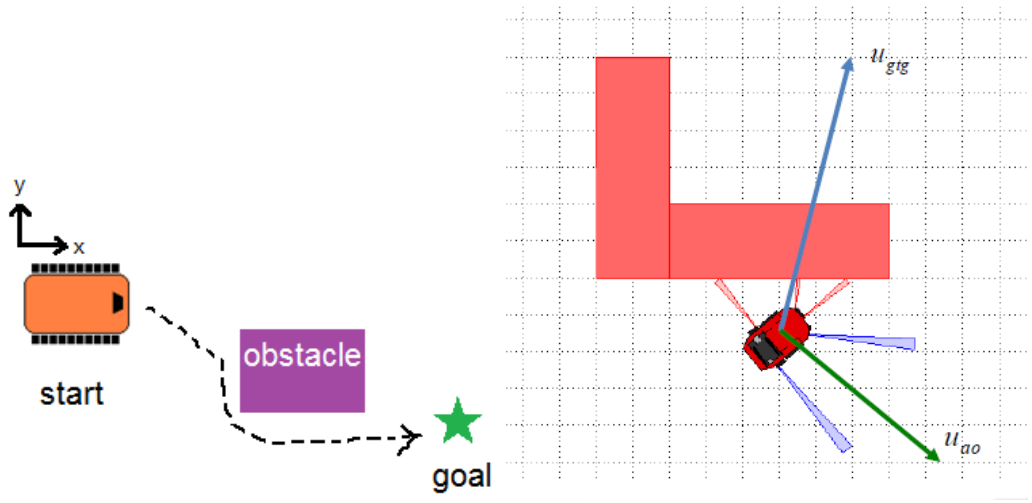


Figure 6: Avoid-Obstacle & Go-To-Goal Behaviors

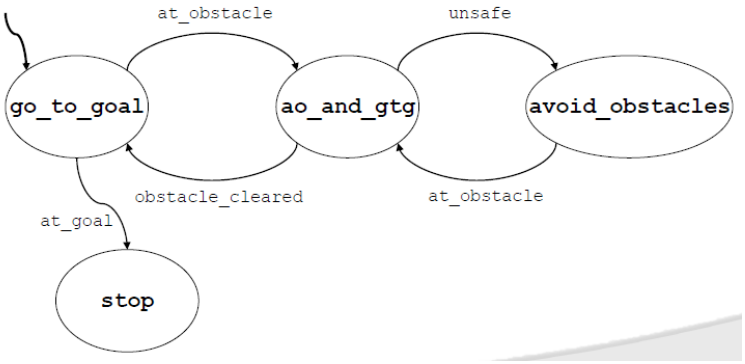


Figure 7: Avoid-Obstacle & Go-To-Goal State Machine

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, separately. Please review the lab procedure if you don't recall what this means.
- For layer 1, to demonstrate random wander, the robot should turn at a random heading and move forward periodically.
- To demonstrate the *Avoid* behavior, the robot should wander in the environment 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 final component of the demonstration involves giving the robot a goal position and the robot should move to this location while also avoiding obstacles. The buzzer, LCD and LEDs can be used to provide feedback on the robot's state. You may use the pushbuttons to start or stop the different layers on the robot and provide a goal position to the robot.

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 plug the robot batter into the wall using the wall charger. Note that this is a fast charger and will not last as long.

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?
11. How did you keep track of the goal position and robot states as it integrated avoid-obstacle and go-to-goal behaviors?
12. What should the subsumption architecture look like for the addition of the go-to-goal and avoid-obstacle behaviors?

Grading Rubric:

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

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

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.