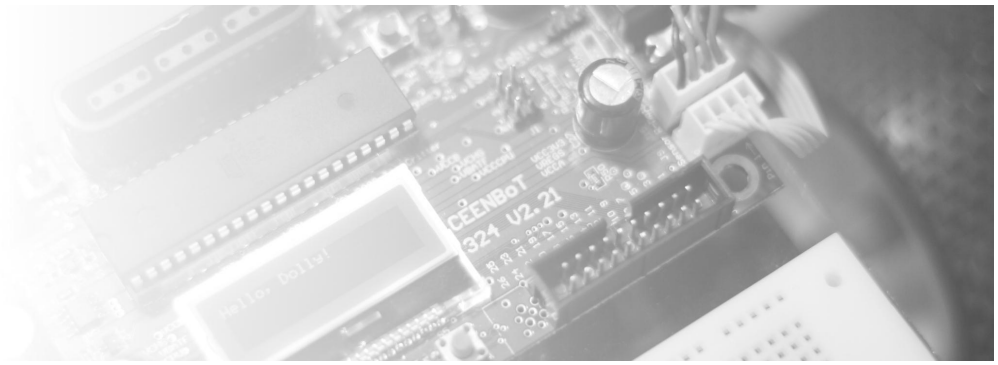
Mobile Robotics I: Lab 3

Obstacle Avoidance with IR Sensors

CEENBoT™ Mobile Robotics Platform Laboratory Series
CEENBoT v2.21 – '324 Platform



The Peter Kiewit Institute of Information Science & Technology
Department of Computer & Electronics Engineering
University of Nebraska-Lincoln (Omaha Campus)

Rev 1.01

(Blank)

Purpose

In this laboratory exercise you will incorporate *exteroceptive* sensor data for the first time in guiding the locomotion of your CEENBoT. Specifically, you will take advantage of the on-board infrared (IR) proximity sensors mounted on the front of the CEENBoT to provide input about the presence (or absence) of obstacles to the front left and front right of the robot. You will create and implement two behaviors for your robot in the C programming language: CRUISE and IR-AVOID. These will provide a starting point for behavior-based control and enable your robot to avoid detected obstacles while moving around in the world.

Lab Objectives

By following the directions in this lab, you are expected to achieve the following:

- Learn how to read the CEENBoT's on-board Left and Right IR proximity sensor inputs over the SPI bus using the appropriate CEENBoT API functions.
- Experiment with various ways to implement an avoidance behavior (stop, back up, turn, move forward). Create a finite state machine description of an avoidance behavior and name it IR-AVOID.
- Implement the IR-AVOID obstacle avoidance behavior in C using the CEENBoT API functions.
- Implement a CRUISE behavior that will cause the robot to roam around freely in some manner while no obstacles are not detected.
- Experiment with the IR-AVOID behavior and observe sensor response with various obstacles in front of the robot (light, dark, at an angle, shiny, soft, rough) to characterize the range, effectiveness and limitations of IR reflectance sensing technology.

Requirements

Preliminary Readings

- You must have successfully completed and have read through the *CEENBoT-API – Getting Started* guide. This document guides through the procedure of writing CEENBoT programs that take advantage of the CEENBoT-API – the document discusses program structure, compiling, linking, and flashing of your CEENBoT.
- For information about Sensors: *The Robotics Primer* by M. Mataric Chapters 7-8 and *Robot Programming: A Practical Guide to Behavior-based Programming* by J. Jones Chapter 9.
- For information on Behavior-based Programming: Mataric Chapter 14 and Jones Chapter 3.

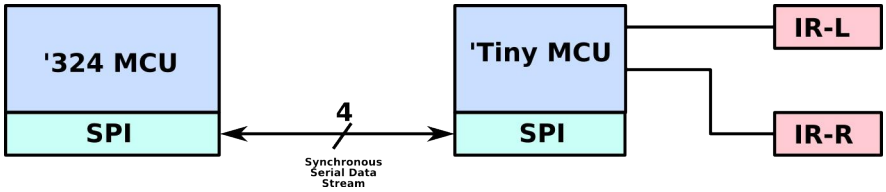
Required Equipment

- CEENBoT, platform '324 v2.21.
- Means to program your CEENBoT. (i.e., USB or Serial ISP programmer).

Background

IR SENSORS INTERFACE

The CEENBoT comes equipped with a Left and Right non-contact bump switch circuitry in the form of Infrared (IR) proximity sensors that are wired and integrated in the following microprocessor scheme as shown by the following simplified diagram.



As can be seen the Left and Right 'bump switches' are digital inputs that are under the control of the **ATtiny48**'s supporting MCU. The information regarding the state of the IR sensors is transferred to the primary MCU (the **ATmega324**) via the SPI serial interface. SPI is a *synchronous* serial peripheral interface that requires four lines for communication: MOSI (Master-Out, Serial-In), MISO (Master-In, Serial-Out), a SS (Slave Select) pin to select the target device to communicate with and the CLK (Clock) signal which synchronizes the receiver to the master.

To request the 'state' of the IR sensors, the master MCU has to request this information from the 'Tiny, and the 'Tiny responds by sending back a byte of data (8-bits) with bits 0 and 1 corresponding to the state of the IR sensors. You request IR state information by using the CEENBoT-API function `ATTINY_get_sensors()` – the *very* same one you used to obtain switch state information, so it is important that you refer to the *CEENBoT-API Programmer's Reference Guid*e for detailed information regarding usage of this function.

A Sharp **GP2Y0D810Z0F** sensor is the central component of the CEENBoT bump switch circuit. The unit is a reflectance sensor that emits and detects infrared light reflected from any surface within the detecting distance of the unit. The unit incorporates an infrared emitting diode, a detector and a signal processing circuitry to detect an object at a distance of 4 inches. When an object is detected, the corresponding *bit* returned by `ATTINY_get_sensors()` will be high (1), and when no object is detected, the bit will be low (0). Ordinarily, the color, texture, angle, materials, of objects affect how much infrared light is reflected or absorbed by the IR sensor, and thus may impact how accurately an object can be detected by an infrared sensor.

A triangulation method is employed in the unit's design to overcome some of these effects. In *some* CEENBoT models, power to the IR sensor is being switched ON and OFF at 10Hz or so to reduce power consumption. The Green LED on the IR sensor board will flash at this frequency when the unit is receiving power. In the *newer* units, you may only observe the flashing while 'obstruction' is taking place.

ROBOT PROGRAMMING

This is the first time you will begin to add modularity in your program 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 for CRUISE and IR-AVOID. CRUISE takes care of moving the robot forward when no obstacles are present, and IR-AVOID takes care of moving the robot away from detected obstacles. More formalism will be introduced in how you are to code the behavior-based structure in future labs. For now, just focus on how to implement each behavior using the API functions.

Directions

Part I – Obstacle Avoidance with the IR-Sensors

1. Experiment with various ways to implement an avoidance behavior (stop, back up or turn, move forward) using the IR sensors. Remember they have a range of 4 inches. Experiment with how long it takes the robot to stop at various speeds to determine your max speed for CRUISE to keep the robot from running into objects it detects. Will you need to use the brake function? Consider which bump switch is activated (LEFT only, RIGHT only, or BOTH) in determining how you will move the robot to turn away from an obstacle.

2. Create a finite state machine description of your chosen avoidance behavior and name it IR-AVOID. Be sure to show what happens when LEFT, RIGHT and BOTH IR sensors are triggered.

3. Create a CRUISE behavior that will move the robot forward in some manner. Take advantage of the CEENBoT API functions.

4. Write a C program that implements the IR-AVOID behavior and the CRUISE behavior. (When working together the two behaviors will cause the robot to move forward, STOP before hitting an obstacle; TURN in some manner away from the obstacle; and continue moving forward until another obstacle is detected.) The IR-AVOID behavior should be coded so that it has precedence in sending commands to the motors whenever an obstacle is sensed over the CRUISE behavior.

5. Experiment with the IR-AVOID behavior and observe sensor response with various obstacles in front of the robot (light, dark, at an angle, shiny, soft, rough). Observe and record both the range and effectiveness of IR reflectance sensing on at least 4 different types of surfaces.

6. **BONUS:** Can you improve upon the standard *bump-bot* mode that comes loaded on the factory-installed CEENBoT? For instance, the current bump-bot program causes the robot to stop when an obstacle is detected, turn 90 degrees away from the obstacle and keep moving. In some environments, the 90 degree turn creates limitations to get the CEENBoT out of a jam. Consider triggering a random component to the way the robot turns when x obstacles are detected in a short time frame (indicating the robot is stuck in some corner). How would this impact the CEENBoT's performance in a crowded environment with irregular obstacles or small openings versus an environment with primarily right angled walls? Does this addition make the robot look more intelligent when trying to get out of a jam?

Questions to consider in your Report

In your lab report, address your approach to each exercise and your observations of your results. Answer all questions posed in the Directions section. Show the finite state machine description for your IR-AVOID behavior. In your C-code, use comments to show where and how the CRUISE and IR-AVOID behaviors are implemented.

What hurdles did you have to overcome to get it to work? What did you observe about IR sensing technology (reliability or lack thereof..)? How well did the detection work for for an obstacle with a black surface, versus a light surface, and the total of 4 different types of surfaces tested above? Append your C source code to your report as per the directions in the following pages. You will demonstrate your IR-obstacle avoidance routine in *class*.

Deliverables

Demonstrations

You will demonstrate your robot's operation of cruise and IR obstacle avoidance next Thursday at the beginning of lab.

C-code

Include a printout of your CEENBoT program. You may include *snippets* of your code and embed them in your lab report as you discuss how your program works, but a separate attachment of your entire source code must be included as part of your report.

Lab Report

The lab report is due in one week, and should include all of the following:

- **Title Page** – Include *Course Number*, *Course Title*, *Instructor Name, Your Name*, *Lab Name*, & *Due Date*.

- **Overview** Section (a brief paragraph of what the lab was about, and the purpose behind it). Please also make sure you touch upon the following as well:

  1. Resources Used: (human, text, online or otherwise).
  2. Time invested in this project.
  3. A high-level description of your robot and program.
  4. If this was a *team assignment*, state how tasks were delegated and split up among you.
  5. Known problems with your solution (e.g., the robot will not work on the *carpet*, or... it breaks if you make it go further than 5 ft (for whatever reason), etc.)

- **Background** Section (discuss some of the theory addressed in this particular lab).

- **Procedure** – Briefly describe the lab procedures for each lab – what were you asked to do? What did the process consist of? How *did you* approach it?

- **Discuss your Source Code** – Discuss and explain any relevant details behind the motivation and manner in which you have written your source code. You can *embed* [small] portions on of your code that are relevant to the discussion for clarity, but please ensure to keep a complete copy of your source code as a separate attachment (see *Source Code* section below). You may also have the reader *refer* to particular pages of your source code attachment instead.

- **Results –** Use this section to answer *questions* posted throughout your lab exercise. In particular those given out in the '*Directions*' section. In the '*Procedure'* section, feel free to refer the reader to *this* section for additional info – that is, there is no need to repeat information.

- **Conclusion** – Reiterate the objective of the lab, discuss what you have learned and any further comments you might have.

- **Source Code** – Attach your C-program also.

Grading

Laboratory Grading Rubric		Labs will be graded out of 30 points, unless otherwise	
Points	**Lab Report**	**Code**	**Demonstration**
10	Follows lab format, all sections are complete, thorough, concise. Answers all questions posed.	Properly commented, easy to follow with modular components.	Excellent work, the robot performs exactly as required.
7.5	Does not answer some of the questions or has spelling, grammatical, content errors.	Partial comments and/or not modular.	Performs most of the functionality with minor failures.
5	Multiple grammatical, format, content, spelling errors, and/or questions not answered.	No comments, not modular, not easy to follow.	Performs some of the functionality but with major failures or parts missing.
0	Not submitted or submitted late.	Not submitted or submitted late.	Meets none of the design specifications or not submitted.