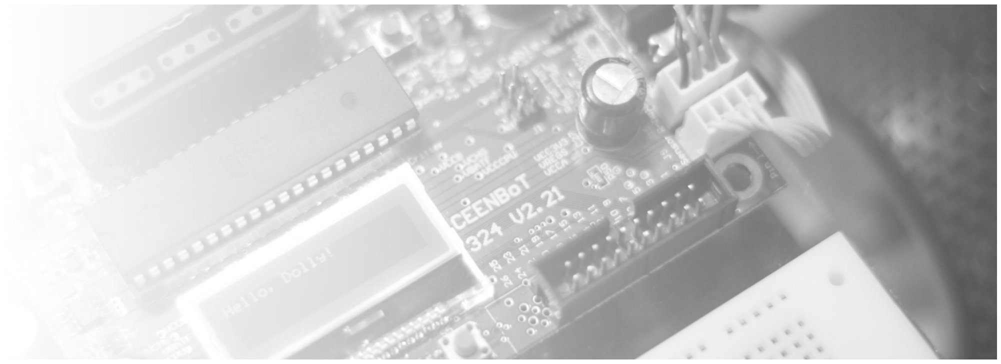
Mobile Robotics I: Lab 5

Obstacle Avoidance with Ultrasonic Sensing

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



Alisa N Gilmore, P.E., *Instructor, Course & Lab Developer*

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

Rev 1.01

Purpose

In this lab, you will add another exteroceptive sensor to your robot, an ultrasonic ranging sensor, mounted on a DC servo motor. You will learn to trigger and gather distance data, explore the limitations of the sensing technology, and use the ultrasonic sensor to create an obstacle avoidance routine for the CEENBoT. You will incorporate the SONAR_AVOID behavior into your existing behavior-based control program structure and demonstrate the operation of all behaviors created using bbc_skeleton.c from Lab 4, Part 2.

Lab Objectives

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

- Experiment with the integration and operation of ultrasonic sensing technology by incorporating a ultrasonic sensor onto the CEENBoT. Perform field tests to discover the range and limitations of this sensor technology.
- Develop modular C code to trigger and condition the data produced by the ultrasonic sensor that will allow you to detect obstacles at some distance in front of the CEENBoT. Compare and contrast the use of the ultrasonic sensor to the IR sensor.
- Create a SONAR_AVOID behavior and add it to your behavior-based control structure in which you will have a total of 4 behaviors with multitasking and arbitration.
- Demonstrate the operation of all 4 behaviors created thus far, including: IR_AVOID, SONAR_AVOID, LIGHT_FOLLOW, and CRUISE, working in unions with a given set of arbitration priorities.
- Get to know the STOPWATCH, and USONIC subsystem modules of the CEENBoT API along with how to implement 'pin-change' interrupts using API functions.

Requirements

Preliminary Readings

- Information about ultrasonic sensors can be reviewed in the Mataric and Jones texts.
- Familiarize yourself with the operation of the *Parallax PING)))* ™ *Ultrasonic Sensor* by reviewing the datasheet.
- Read the following chapters in the *CEENBoT-API: Programmer's Reference* manual (Rev 1.05) for relevant information about how to utilize API functions to perform the tasks required in this laboratory exercise:

  Chapter 14: *The USONIC subsystem module* – to learn how to use the Parallax ™ Ultrasonic Sensor the 'easy' way.

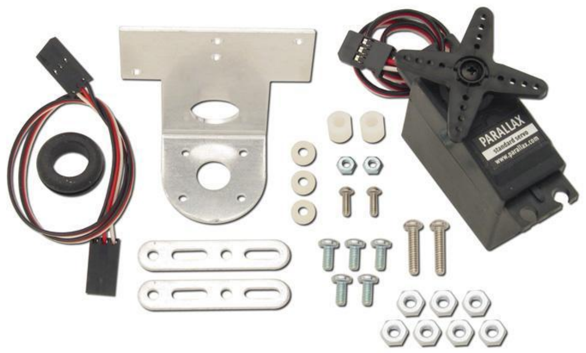
Note: The *Programmer's Reference* is now up to Rev 1.05 – please, make sure that you have downloaded the *latest* PDF.

Required Equipment

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

Background

The *Parallax PING))) Ultrasonic Distance Sensor*



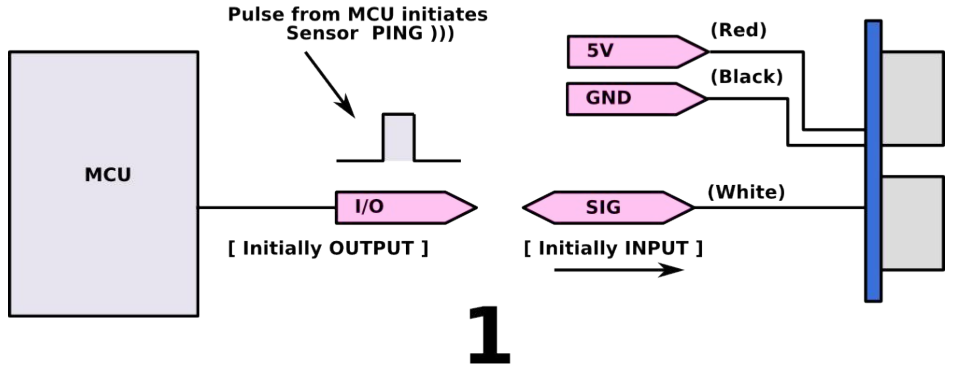
For this laboratory exercise you will be using the *Parallax PING))) Ultrasonic Distance Sensor* (#28015 – pictured above on the left) and will assemble and attach it using the mounting bracket and screws onto the RC servo (pictured above on the right) in preparation for future upcoming labs. For *this* laboratory exercise, however, you will focus only on assembling the unit, and using the ultrasonic sensor alone, mounted on the CEENBoT so that it faces forward.

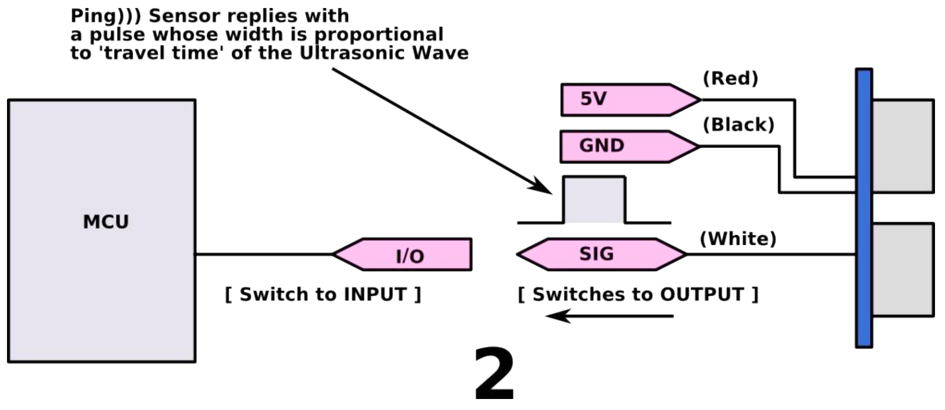
Compared to the IR sensor you have been working with thus far, the ultrasonic sensor possess notable differences. First, the detection range of the ultrasonic sensor is larger – it will detect objects at a distance anywhere between 2cm (~0.8in) up to 3 meters (3.3 yards). Second, unlike the IR sensors which are basic 'ON/OFF' sensor devices (object is either blocking or not blocking), the ultrasonic provides gives information that can be used to calculate the *distance* to target – opening up all sorts of creative possibilities. However, unlike the IR sensor, the ultrasonic device is also subject to specular reflection and will not accurately detect objects at angles of 45-degrees or less from the forward direction of the robot. Additionally, unlike the line-of-sight IR sensors, the ultrasonic ping emitted by the sensor is a 'conical' beam shape with a width between 50 to 60-degrees. (*What other differences can you observe between the sensors?*)

Theory of Operation

The operation of the *Parallax PING))) Ultrasonic Sensor* is outlined in its datasheet. Consult this document for operating details of the unit.

The Parallax ultrasonic sensor is a 3-pin unit with connections for *power, ground* and a *digital* I/O pin used to both 'trigger' the device and indicate echo return. Upon power-up of the unit, the digital I/O pin is configured as an INPUT. In the INPUT stage, the sensor waits for a 'triggering' signal from a controlling device (such as an MCU digital I/O pin) to tell it when to emit an 'ultrasonic *ping*'.




Ping))) Sensor replies with a pulse whose width is proportional to 'travel time' of the Ultrasonic Wave
MCU
I/O
[Switch to INPUT]
5V
(Red)
GND
(Black)
SIG
(White)
[Switches to OUTPUT]
2

As soon as the sensor detects a triggering signal, it sends an ultrasonic ping, and immediately switches the I/O pin from an INPUT to an OUTPUT. This pin then goes HIGH to indicate the sensor has *fired the ultrasonic ping*. This pulse will remain HIGH until the sensor detects the ping 'echo' back to the device. At the point of echo reception, the signal will return to a LOW, and switch to INPUT mode, waiting for the next 'trigger'. The scenarios of the ultrasonic sensor's operation are depicted on the next page.

As soon as the ultrasonic sensor is triggered using a digital I/O pin (on the MCU-side), you must immediately switch or 'reconfigure' this same digital I/O pin from OUTPUT to INPUT in order to catch the beginning (LOW-to-HIGH transition) of the pulse and the end (HIGH-to-LOW) emitted from the PING))) device.

Note: The preceding discussion is not meant to be taken as a comprehensive detailed operation of the Parallax PING))) Ultrasonic Sensor – PLEASE refer to the datasheet for pulse and timing specifications for details.

There are several ways in which you can accomplish this task – some 'pointers' are given over the next page or so. However, the key is that the 'pulse-width' (measured in units of time) is measured to determine the time between when the 'ping' ultrasonic wave was sent and echoed back in order to compute the distance-to-target.

Computing the Distance-to-Target

To measure the distance-to-target you'll use the following generalized formula:

$$d = \frac{1}{2} \cdot v \cdot t$$

where *d* is the distance, *v* is the rate or speed, and *t* is the time in seconds. For our purposes, we will use the speed of sound in *air*, which is 344.8 m/s (or 34480 cm/s – since we want distances in units of 'cm'). Thus, we have the following:

$$d = \frac{1}{2} \cdot (34480) \cdot t \;=\; 17240 \cdot t$$

however, *t* is still in units of *seconds* which is much too large, since the pulse emitted by the sensor spans in units of *micro-seconds*, so to specify *t* in units of micro-seconds we multiply the above by **`1x10`**$^{-6}$, and thus, arrive at the following:

$$d_{\text{cm}} \;=\; 0.01724 \cdot t_{\mu\text{s}}$$

Keep in mind that the reason we multiply by ½ is because we want the distance from 'sensor-to-target' – we don't want the distance from 'sensor-to-target and back'.

CEENBoT-API Features You May Find Useful

You can achieve what is needed for this exercise by letting the CEENBoT-API handle the Ultrasonic sensor for you by invoking the **`USONIC`** subsystem module which was specifically written to make use of the *Parallax PING))) Ultrasonic Sensor*. Here you would only need to start the **`STOPWATCH`** subsystem module, but you wouldn't have to interact with it, since **`USONIC`** will take care of the rest. If you choose this route, the SIG pin on the Ultrasonic Sensor *must* be attached to I/O pin **`PA4`**, which is **`pin 2`** on **`Header J3`** (there is a table further down in this document).

Using the `USONIC` Subsystem Module

The **`USONIC`** subsystem module in the CEENBoT-API was specifically written to control the *Parallax PING))) Ultrasonic Sensor*. This module uses the **`STOPWATCH`** subsystem module to measure the pulse-width of the sensor in microseconds, and also sets up the pin-change interrupts to let the MCU know when the 'SIG' pin on the sensor has changed. It completely facilitates using this sensor as everything is handled for you. The following 'snippet' of code shows the typical usage:

```
#include "capi324v221.h"

void CBOT_main( void )
{

    // Local variables that we'll be using:

    unsigned long int usonic_time_us;       // Holds 'trip' time in micro-seconds.
    SWTIME            usonic_time_ticks;    // Holds 'ticks' from STOPWATCH.
    float             distance_cm;          // Holds the distance-to-target in 'cm'.

    // Open the needed modules:

    STOPWATCH_open();        // Must be opened FIRST (needed by USONIC).
    USONIC_open();           // Open the Ultrasonic module.
    LCD_open();              // Open the LCD subsystem module.

    // NOTE:  Here we're going to measure ONCE -- but you would place this in
    //        a loop of some kind to measure multiple times.

    // Ping ONCE and store 'ticks' elapsed from STOPWATCH in variable.
    usonic_time_ticks = USONIC_ping();

    // Convert from 'ticks' to 'us'.  ** Note the TYPE-CASTING ***
    usonic_time_us = 10 * (( unsigned long int )( usonic_time_ticks ));

    // Convert to 'cm' using d = 0.01724*t_us.
    distance_cm = 0.01724 * usonic_time_us;

    // Print the result to 3 decimal places for 'show'.
    LCD_printf( "Dist = %.3f\n", distance_cm );

} // end CBOT_main()
```

So as you can see, only *three* principal functions are required – the first two only happen once, per say. You open the **STOPWATCH** first, since the ultrasonic *needs* to use it internally to measure the pulse-width from the ultrasonic sensor in units of microseconds. The STOPWATCH, however, returns time in units of *ticks*, where each *tick* is worth 10us. This variable is stored in a variable of type **SWTIME** (for STOPWATCH time), which essentially defaults to an `unsigned short int` (16-bits), in case you're curious.

Since each 'tick' is worth 10us, the next line down multiplies this value by 10 to obtain the time in units of microseconds. However, since multiplying by 10 might result in overflow (beyond 16-bits), we have to *typecast* to a LARGER storage type. Note that the variable `usonic_time_us` is of type `unsigned long int` also to accommodate for this potentially larger result.

It shouldn't be too difficult for you to take the above code 'snippet' and expand on it so that you can *continuously* monitor distances in real-time by implementing the above in some kind of *loop*.

Further information on the **USONIC** subsystem module (and also the STOPWATCH) can be found in the *CEENBoT-API: Programmer's Reference* manual – make sure you read on that to understand what is happening on the 'inside' – in particular the **STOPWATCH**, which could be useful for to you for other tasks.

ROBOT PROGRAMMING

In this lab you will append your modified bbc_skeleton.c program so that at the end it will include a total of 4 behaviors. You will add to your existing IR-AVOID, CRUISE and LIGHT_FOLLOW behaviors, a SONAR_AVOID behavior. You will also add ultrasonic sensor-data-gathering functionality to the program using a modular function call format.

When you complete this lab, you will have a random wanderer robot that seamlessly avoids obstacles at a distance and backs up from obstacles close up. When a high enough intensity light source appears, the robot will follow it. You will demonstrate this functionality and have your program structured in a behavior-based control structure in C with cooperative multitasking among behaviors and fixed priority arbitration. (The arbitration priorities are given in the Directions section.)

Directions

Part I

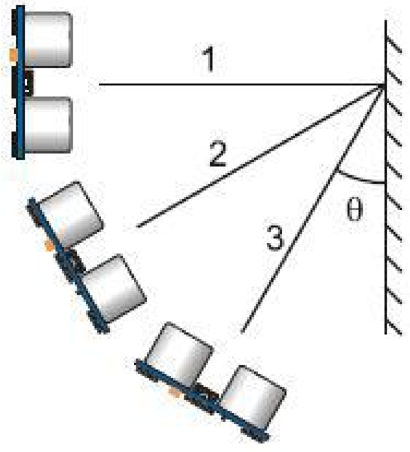
1. Assemble the servo mounting bracket and the ultrasonic sensor, and install both on the front of the CEENBoT. Next, wire the ultrasonic sensor as follows. The sensor has one I/O pin in addition to power and ground wires. Connect these wires to the CEENBoT via Header `J3`. Pin connections are outlined below for your reference:

MCU Pin	Pin on J3
5V	19
GND	20
PA4 (for connecting to SIG)	2

Note: Please be mindful of the pin numbers and where you connect your wires!

Write code to *trigger* the ultrasonic pulse, wait, and listen for the return pulse (echo) and calculate a distance based on the width of the return pulse. Recall that it will be necessary to reconfigure PA4 (SIG above) from OUTPUT to INPUT between 'triggering events'. Decide on how often you will trigger the sensor and create intelligence to manage your readings. A recommended way to do this is to create a C function to gather ultrasonic sensor data and return readings to your program. This ensures modularity.

2. Test your sensor by positioning the sensor at 90-degrees from a wall at a *known* distance. Run your ultrasonic program, and display your detected distance on the LCD. Check to see that your sensor reading generated by your program is accurate. Do this for several distances: a *small* distance (~2in or 5cm), a *medium* distance (~12in or 30cm), and a *larger* distance (~24in or 60cm). Compare your actual distances. For the 2nd to 3rd trials, verify that your time actually doubled. If your distances are off, check your control code for correct operation. You may also need to tweak your distance calculation until distance measured is accurate – keep in mind that the *speed of sound* in air is *temperature dependent!*



3. Next, position your CEENBoT 12 inches from the wall at 90-degrees as shown in the previous figure. Record the distance detected. Then, move the CEENBoT at smaller angles (as shown (2) and in (3) in the previous figure) from the wall. Each time, record the distance detected. Do the values change? At what angle does the sensor cease to detect the wall?

Part II

1. Append your modified bbc_skeleton.c program with the following:

   a) Add the ultrasonic sensor-gathering function(s) you have developed to trigger the ultrasonic sensor and return a distance.

   b) Create a SONAR_AVOID behavior that will move the robot away from obstacles at some distance away. The goal is to detect obstacles and move around them before they are encountered. This SONAR_AVOID should not require the robot to back up, as with the IR_AVOID, but to seamlessly move around obstacles detected in front of the robot. This is because the ultrasonic sensor can detect obstacles farther away (recall the IR sensor has a detection range of 4 inches). This should be designed as a servo behavior, not a ballistic one.

   c) Add in priorities for the BBC arbitration scheme (highest to lowest as: IR_AVOID, SONAR_AVOID, LIGHT_FOLLOW, CRUISE (EXPLORE). You should have the LIGHT_FOLLOW behavior already created from Lab 4, Part 2. This means that the robot will cruise or explore, until it sees a light, which it will follow, until the sonar detects something in the distance, or the IR sensor detects something in close proximity.

2. Be prepared to demonstrate the modified bbc_skeleton.c program with the above 4 behaviors added. It is strongly recommended that you provide some visual or audio indication of the active behavior or current state while the program is running. This will not only help ensure your program works the way you expect it to but also for demonstrating how it works. Options for visual or audio feedback of behavior include:

   a) LED display of the current "state".
   b) Flashing LEDs based on "state".
   c) Playing different speaker frequencies based on "state". (Note, this functionality is not a part of the API, but it is possible to create a method to make this work with the on-board speaker).

Questions to answer in the RESULTS section of your Report

- What hurdles did you have to overcome to get it to work?
- What did you observe about ultrasonic sensing technology? Include in a chart your tests results from Part 1 of the ultrasonic ranging sensor's performance at 3 different distances and 3 angles.
- Give several comparisons and contrasts to your work with IR sensors.
- In Part 1, question 2, did you find it necessary to tweak your ultrasonic distance calculation based on field tests? How so?
- In Part 2, discuss your process to append your modified bbc_skeleton.c program to achieve 4 behaviors working in unison and add the sensor gathering data functionality for the ultrasonic sensor.
  .

Deliverables

Demonstrations

You will demonstrate your robot's execution of these new capabilities during the designated class meeting or lab time.

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 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?

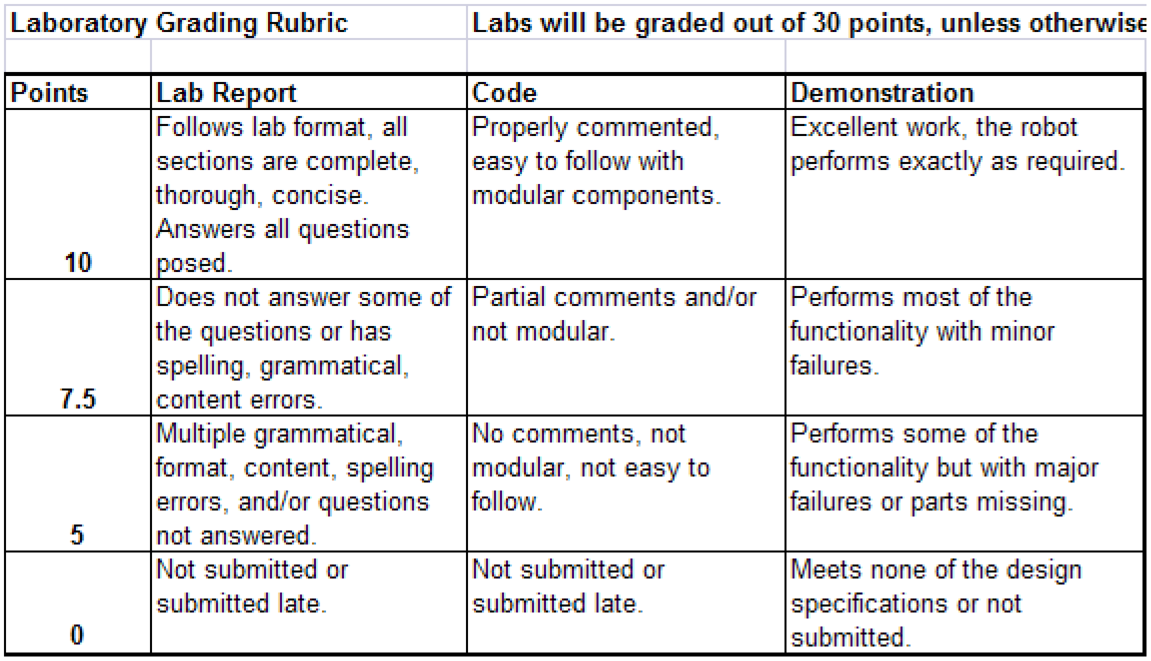
- **Discuss your Source Code** – Discuss and explain any relevant details behind the motivation and manner in which you have written your source code. You can *embed* [small] portions on of your code that are relevant to the discussion for clarity, but please ensure to keep a complete copy of your source 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.