

University of Pennsylvania
ESE 112: Introduction to Electrical & Systems Engineering

Lab 8: Maze Navigation

Objectives

- To gain further experience with Object-Oriented Programming (OOP)
- To introduce the Ping sensor
- To emulate a Bot that can move through a maze composed of walls

Introduction

You have already programmed your Boe-Bot that could navigate through a maze whose dimensions were given. But at that time, you had to keep account of how long the Boe-Bot should keep performing a certain move in order to navigate through the maze. For example, if the maze was composed of a 1-inch straight line, a curve to the right and then a 2-inches straight line, you had to tell the Boe-Bot how long it should keep moving forward before turning, and then for how long it should keep moving forward after the turn. As a consequence, your program was highly dependent on the maze that you were working with, and for each maze that you had to navigate the entire code had to be changed.

For this week's lab you will be using the Ping sensor to navigate through a maze with walls, and you will be constructing a program that will allow you to navigate through any maze, independently of the path that the Boe-Bot must follow.

Background

The PING)))™ Ultrasonic Distance Sensor

The Parallax PING))) ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It provides its users with great accuracy when detecting an object's distance. The PING sensor works by transmitting an ultrasonic (well above human hearing range) burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. It sends a brief chirp with its ultrasonic speaker and measures the echo's return time to its ultrasonic microphone. By measuring the echo pulse width, the distance to the target can easily be calculated.

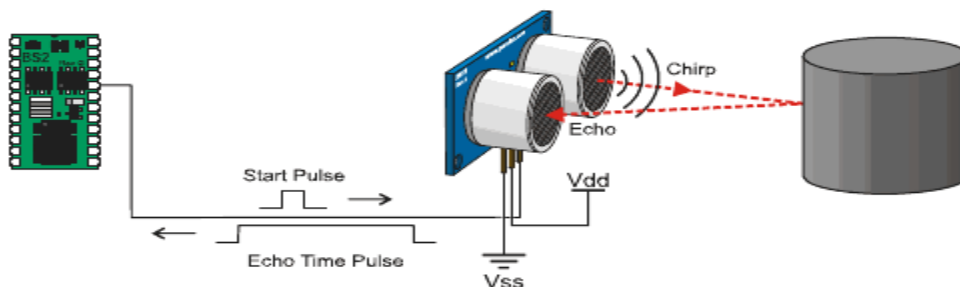


Figure 1: Ping Sensor Connection

In order to use the Ping sensor you will be working with the Ping class. The Ping class allows us to control the sensor by using object oriented program. As with the servos for the PWM class, the Ping class allows us to create a new sensor object and then call methods on it.

To use the Ping class you will have to import it just like you imported the CPU class. To do this you will have to put the following at the top of your program:

```
import stamp.peripheral.sensor.Ping;
```

You will also have to save the Ping.java program that we provide to your current folder. To create a new Ping object, use the following syntax:

```
Ping instance_name = new Ping(CPU.pinX);
```

Replace “instance_name” and “pinX” with what you need. Now, that we know how to create a ping object, how do we measure distances with the sensor? The Ping class provides us with all the methods that we need ot measure distances. Some of them are:

- getIn() - Returns PING))) distance values in inches
- getCm() - Return PING))) distance values in centimeters

For more information on the other methods to get the distance, please refer to the Ping.java file.

If the BoeBot is supposed to be able to go though a maze, it is necessary for it to have the capability of detecting objects located at its front or at its side. How do we control the sensor to make it turn to the left or to the right depending on what we need? We use the PWM class. Your Ping sensor came accompanied with a Bracket Kit. This kit has a servo that can be connected to the Ping sensor. We can then create a new PWM object to control the sensor's servo. This is what we are going to be doing.

In terms of accuracy and overall usefulness, ultrasonic distance detection is really good, especially compared to other low-cost distance detection systems. That doesn't mean that the Ping))) sensor is capable of measuring "everything". Figure 2 shows a few situations that the Ping))) is not designed to measure: (a) distances over 3 meters, (b) shallow angles, and (c) objects that are too small. Keep this in mind when controlling your sensor's servo.



Figure 2 – Situations for which the sensor is not designated

Materials

- Boe-Bot unit with Javelin Stamp
- PING)))™ Bracket Kit
- PING)))™ Ultrasonic Distance Sensor
- 3-pin male/male reader
- Jumper wires

Pre-Lab Questions

1. You will have to attach your Bracket kit and your Ping sensor to your BoeBot. In order to do this, follow the instructions that came with your PING)))™ Bracket Kit. You should skip steps 7 and 8. Instead of following these steps, you should follow the Breadboard Connection section on page 3. Note that the 3-pin male/male reader is the same one that you used for the Whisker navigation program.
2. After following these instructions connect the sensor servo's extension cable into port 14 (the one with three connections to the top of the breadboard of your BoeBot).

Lab Instructions

Part I - Getting acquainted with the Ping Class

1. Write a PingTest class that measures the distance of a target object from the sensor to the BoeBot. Place the object about 5 cm away from the BoeBot and then get its distance in inches. Repeat these measurements for the object 10, 15 and 20 cm away.
2. Build an LED circuit in your breadboard. Now, implement your PingTest class to make the LED turn on when there is object in range.

Part II - Moving the Ping sensor using a servo

So far, you have been measuring objects to the front of the BoeBot. Now you want to be able to move your ping sensor so it can take measurements of distances to the left and to the right of your BoeBot. To do this, you are going to add some more functionality to your BoeBotControl class.

1. Add to your BoeBotControl class, an instance variable of type PWM called servoLook to control the sensor's servo. Now you need to change your constructors to also initialize servoLook when a BoeBotControl object is created. If no inputs are provided to the constructor, servoLook should receive a new PWM object created at pin 14. If inputs are provided to the constructor, this constructor should accept an extra PWM variable and assign its value to servoLook.
2. Using servoLook object, we can make the Ping sensor face right, left or the center, you are going to use the PWM class. When you call the update method, you provide two inputs the highTime and the lowTime (refer to Javadocs for more information). The lowTime is going to be the one that you have been using so far. For each different

highTime value that you provide, the servo will turn to a certain direction (left or right) and stop at a specific position. Each position has a specific highTime associated to it. No matter which position the sensor was facing before, when you call the update method and provide the highTime of a specific position, the sensor's servo will go to the position relative to the highTime assigned to it.

Add to your BoeBotControl class the methods lookRight, lookLeft and centerLook. All three methods should accept the integer time as an input and don't need to return anything. This variable will control how long the sensor has to keep facing right or left. Use these three methods that you have created to find the highTime values that make the Ping sensor face right and left at 90 degrees, and the one that makes it face the center. Record these values for your lab report.

In the BoeBotControlTest, create an instance of BoeBotControl and test your lookRight, lookLeft and centerLook methods.

Part III – MazeBot

Create a new class called MazeBot. A MazeBot object should have: an object of type Ping and an object of type BoeBotControl. When a new MazeBot object is created, if no inputs are provided to the constructor, the constructor should create a new Ping sensor object at pin 15 and a new BoeBotControl object. If inputs are provided to the constructor, the values for the Ping and BoeBotControl objects provided to the constructor should be assigned to the MazeBot instance variables. Your MazeBot class should also have the checkWall method. This method should receive as an input an integer and return a boolean value.

Part IV - Maze Navigation

Now that your BoeBot is able to look right, look left or face the center, you are going to make it go through a maze by writing the MazeNavigation class. Remember that this maze has walls, so you can utilize the Ping sensor to decide what to do. Your code should be independent of the distances of the maze and, because of that, these distances are not provided.

The way to implement this code is to follow the right hand rule. Any maze fan knows that the easiest way to solve mazes is by following the right-hand wall. You might head down some dead ends you'd otherwise miss, but you'll almost always eventually reach the end. When the BoeBot follows the maze, it should navigate it by following the right-hand wall. Depending on the situation, this might simply make your BoeBot take a step forward, or it might have to turn first.

This may sound a bit tricky, so before you start writing code, think and record what should happen — under what conditions can the BoeBot just move forward? If it has to turn, which way should it turn? What should your BoeBot do if there's a wall in front of it?

Then, write the MazeNavigation class that will make the BoeBot navigate through the maze. Your MazeNavigation class should have a main method that creates a MazeBot object. You are not allowed to have new objects of type PWM, BoeBotControl or Ping in your MazeNavigation class. [Hint: Think about the instance variables of MazeBot, how you can access them and that they are objects with their own behaviors.]

Post-Lab Questions

1. How does the Ping))) sensor work? (Don't copy from the lab write up. Explain with your own words)
2. What is the range of distances that the Ping sensor can detect? What value does it return when the object is out of this range?
3. What three sensor-object orientation scenarios could cause the Ping))) sensor to return an incorrect distance measurement? Why do you think, under those conditions, it doesn't return the right measurement?
4. Your MazeBot class has two different constructors. How does the program know which constructor should be called what when a new MazeBot object is created?
5. What does the call `CPU.delay()` do in your BoeBotControl's `wheelForward` and `wheelBackward` methods? How did you account for delay values greater than 32767.
6. Provide pseudo-code for `lookRight`, `lookLeft` and `centerLook` methods in BoeBotControl class and `checkWall` method in MazeBot class.
7. Describe your MazeNavigation program/algorithm using a flowchart.
8. What were some problems you encountered while you were programming the BoeBot, and how did you resolve them?

