

Lab 4: Introduction to Boe-Bot Navigation

Objective

- To introduce Boe-Bot Navigation
- To implement your own methods
- Introduce API and Javadoc
- To practice with more Java Language Syntax features

Background

Java Programming Language:

We will further explore java language syntax and practice writing methods to solve parts of the problem. For details related Java syntax, please refer to the class notes.

Boe-Bot Overview:

Last week basic Boe-Bot hardware and software were introduced, along with learning basic Java language programming syntax. This week you will be introduced to navigating Boe-Bot with servo motors. Recall the Boe-Bot components from Figure 1 in Lab 3. Note that front of the Boe-Bot is the breadboard end.

Servos:

The servos, or the wheels, are what allow the Boe-Bot to move. The CRS (Continuous Rotation Servos) on the Boe-bot use an analog signal to encode the rotation rate. Each servo is controlled by three input wires: the red wire is usually connected to the power supply (Vdd), the black wire is usually connected to the ground (Vss) and the white wire is usually connected to the controlling signal (in this case pins 12 and 13).

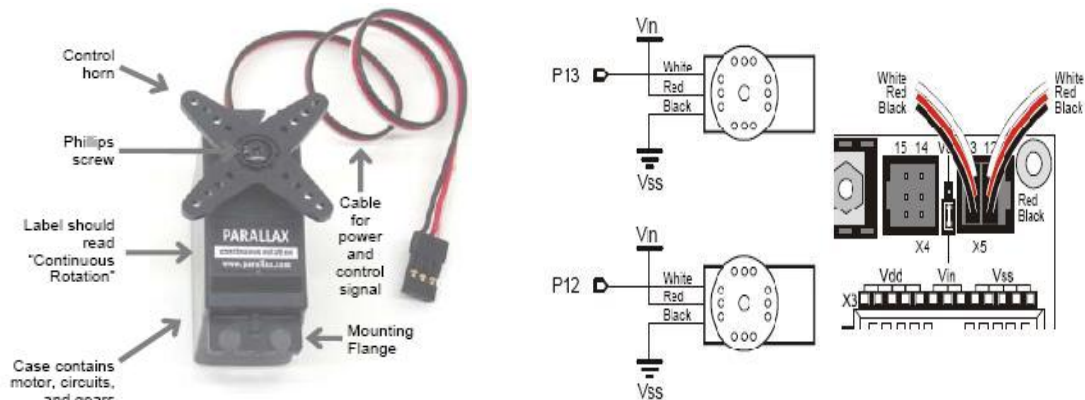


Figure 1: Continuous Rotation Servos (left) and Connection (right)

The pulse is usually a square wave function (see Figure 2). For the servo motor that we are using, the power supply is 6 volts (i.e. the input voltage from the power source). The Javelin stamp is capable of generating a square wave by sending a command every 20ms. For the servo that we are using, the neutral point (the pulse width at which the servo stays

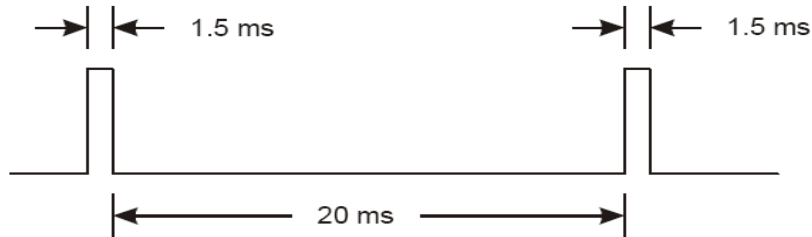


Figure 2: Square Servo Pulse

at 90 degrees) is about 1.5 ms. You will control the servos (wheels) using the `pulseOut` method the CPU class (see the CPU class documentation in the API and Javadoc section). This method accepts two arguments, pin number and pulse width. The pulse width is the duration of the pulse, measured in units of $8.68 \mu\text{s}$. The width of 173 is equivalent to “centering” the servo (i.e. $173 \times 8.68 \mu\text{s} = 1.5 \text{ ms}$), which will make the Boe-Bot stop. A pulse width of 220 will make the wheels turn counter-clockwise, while a pulse width of 130 will make the wheel turn clockwise.

If the servos are not completely centered and do not remain stationary when sending a `pulseOut` of 173 they must be calibrated. You will calibrate your servos in class.

API and Javadoc:

An Application Programming Interface (API) defines the ways by which an application can use libraries (i.e. already implemented source code). Javadoc is a documentation generator (from Sun Microsystems) for generating API documentation in HTML format from Java source code. Here is an example of the API documentation for the CPU class: <http://www.seas.upenn.edu/~ese112/fall09/boebotResources/stamp/core/CPU.html>

When we see any class documentation, we call it a class' public *interface* (i.e. the public face that it shows the world). The documentation will provide four main headings:

- a. Field Summary
 - Has information about variables. The variable's type, identifier, and brief description about its use are provided.
 - For now, variables will be of static nature but later you may see variables that that are not of static nature.
- b. Constructor and Method summaries
 - Describe the public constructors (do not worry about these for now) and methods
 - Method summary just provides the method header and a small description of the method

c. Method Detail

- Provides detail on method inputs (parameter(s)) and output (return type) and some extra details

Note that if method/variable is private, then it will be not a part of its interface.

Materials

- Boe-Bot unit with Javelin Stamp
- 4 AA batteries or AC Adapter
- USB cable
- Javelin Stamp IDE
- Phillips screwdriver
- Ruler (in inches)

Pre-Lab Questions

1. To which pin is the right servo supposed to be connected?
2. Give the method header for a static method named “move” that returns nothing and performs various tasks.
3. What does the return statement do?
4. What problem(s) might you run into if the servos are not centered?
5. What is a Javadoc? What is difference between Method Summary and Method Detail? Provide an example.

Lab Instructions

Part I – Getting acquainted with servos

For this section use [ServoTest.java](#) for writing your code for experimental work.

The servos (wheels) are a crucial part of the Boe-Bot that you will be using in all the labs. After all, what good is a robot if it doesn't move? The following exercise will get you acquainted with the servos and with the programming necessary to make them move. Make sure you have read the paragraph on servos in the Background section.

1. First, check what pins your servos are connected to. Follow one cable from one servo through the Boe-Bot to the port on the top. The number next to the port is the pin number. Since it will be easier if everyone's port numbers are the same, make sure your left servo (as if looking from the back, or the end with the ball wheel) is connected to pin 13 and your right servo is connected to pin 12.
2. You may have to calibrate your servos. To calibrate your servo, send a pulse with a pulse width of 173 (i.e. centering pulse width) continuously by placing your code in a while (true) loop. On the back of each servo by the battery pack there is a small hole giving access to a yellow and blue potentiometer. While running your calibration program, **if you see your wheels turning** then use a Philips screwdriver to the potentiometer until the servo stops turning. Make sure you do not turn them too fast.

There is no direct way of commanding the Boe-Bot to move forward a certain distance or for a certain amount of time. The only measurement tool you have is number of pulses you send using the pulseOut command. Using a counting variable to count the number of pulses, and knowing how far the Boe-Bot moves in a certain number of pulses however, you can easily tell the Boe-Bot to move forward five inches.

3. Write a program that uses a counter to count the number of pulses and that makes the Boe-Bot move forward for 50 pulses and then stop (remember, to stop use a pulse width of 173. You only need to send one pulse at 173 for the Boe-Bot to remain stopped indefinitely.)
4. Place your Boe-Bot (after downloading the program of course) on the ground next to the masking tape line, with the center of the wheels at the starting line. Let the Boe-Bot run its 50 pulses.
5. Measure the distance it traveled and note it down. You will submit this table of values as part of your lab report for this lab.
6. Repeat steps 4 and 5 several times each (at least 3 each) for 100, 150, 200, 300 and 400 pulses.
7. Now, using all your values, calculate an average "distance per 100 pulses" value. This is the conversion factor you will use in future labs. Don't worry if your number is different from another group's value. The Boe-Bots may have slight variations that give it different speeds.

Part II - Navigating Robot (predetermined path)

Now, create a java program called [NavigatePath.java](#) that will make the Boe-Bot move in a path whose dimensions are given in Figure 3. All other aspects are up to you (whether to stop at each turn, which way to go, etc.). Note that we encourage modular programming, meaning that you should breakdown your work into methods. The `main` method should trace out the path as part of its task. In order to get work accomplished you should write your own methods to stop, turn, go forward etc. It's ok if your Boe-Bot is slightly off the path – it doesn't move in a consistent fashion. As long as it traces the path relatively well you have successfully completed your lab.

There are two paths you need to command your Boe-Bot to travel: Figure 3 and Figure 4. After you get each one to work successfully, have the instructor or TA sign off on your checklist.

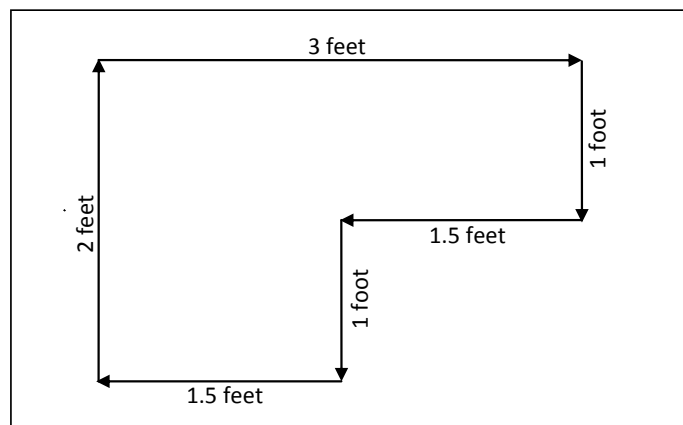


Figure 3: Navigate Path I

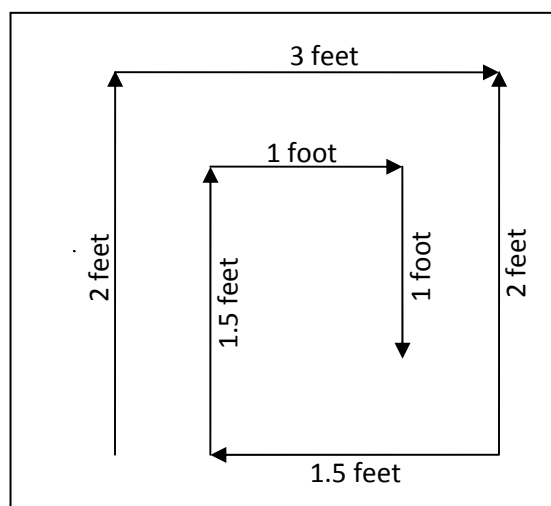


Figure 4: Path 2

Part III - Practicing Further with Java Syntax

Complete the questions on the link below to practice with topics on scope, debugging, nested loops, and code tracing:

<http://www.seas.upenn.edu/~ese112/fall09/java/sdnc.html>

Discussion Questions

Note: If you're asked for pseudo code, then it does not imply your actual Java code (your Java programs should be submitted on Blackboard).

1. Provide an overview on how your group approached the navigation path with dimensions and route.
2. Explain each of your methods that you wrote to allow the Boe-Bot to navigate any path with given dimensions and route (give pseudo code, not actual java code).
3. What changes did you make to your Path I program (Figure 3) to accomplish the Path II navigation (Figure 4)?
4. How would you program the Boe-Bot go backwards in a straight line for a certain distance? Explain your approach. Give pseudo code of your algorithm.
5. Answer all questions from Part III in the write-up.

Submission Guidelines

Submit the following on paper at the beginning of the next lab:

1. Your table of distance values from the servo experiment
2. Discussion questions
3. Signed checklist

Submit all of your Java programs from to Blackboard Digital Drop Box in one zipped folder using the format on the ESE112 website under the Course Information section. Only **one** submission per group is required. Make sure that your files mention the person who you are working with (at the very beginning of the .java file).