

Week 1

Experiment #3

Testing the Servos Individually

In this activity, you will program the BASIC Stamp to control the rotation of each of the Parallax pre-modified servos on the Boe-Bot.

How Servos Work

Normal (un-modified) hobby servos are very popular for controlling the steering systems in radio-controlled cars, boats, and planes. These servos are designed to control the position of something such as a steering flap on a radio-controlled airplane. Their range of motion is typically 90° or 180°, and they are great for applications where inexpensive, accurate high-torque positioning motion is required. The position of these servos is controlled by an electronic signal called a pulse train, which you'll get some first hand experience with shortly. An un-modified hobby servo has built-in mechanical stoppers to prevent it from turning beyond its 90° or 180° range of motion. It also has internal mechanical linkages for position feedback so that the electronic circuit that controls the DC motor inside the servo knows where to turn to in response to a pulse train.

A Parallax pre-modified servo does not have the position feedback and mechanical stoppers you find in normal hobby servos. You can send the same electronic signals (a pulse train) to a Parallax pre-modified servo as you would normally send to a hobby servo. In a hobby servo, a given pulse train makes it turn to a certain position and stay there. The same pulse train causes a Parallax pre-modified servo to turn continuously. The pulse train also sets the pre-modified servo's speed and direction. So, instead of controlling airplane flaps, the Parallax pre-modified servos are used as BASIC Stamp controlled motors that make the Boe-Bot's wheels turn.

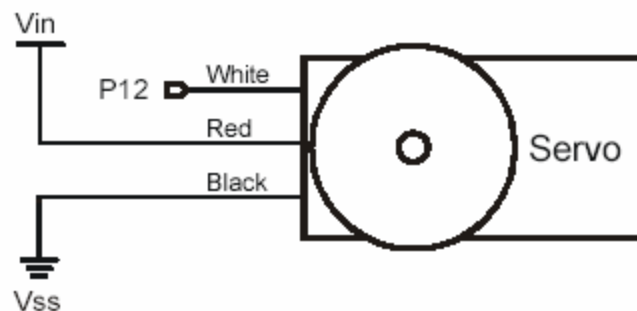


Figure 1: Servo connection schematic.

Figure 1 shows the circuit that is established when a servo is plugged into the servo port labeled 12 on the BOE Rev B's top right corner. The red and black wires connect to the servo's power source, and the white (or sometimes yellow) wire is connected to a signal source. When a servo is plugged into servo port 12, the servo's signal source is BASIC Stamp I/O pin P12.

Chapter #1: Assembling and Testing Your Boe-Bot

Figure 1.25 shows the circuit that is established when a servo is plugged into the servo port labeled 12 on the

BOE Rev B's top right corner. The red and black wires connect to the servo's power source, and the white (or sometimes yellow) wire is connected to a signal source. When a servo is plugged into servo port 12, the servo's signal source is BASIC Stamp I/O pin P12.

About Time Measurements and Voltage levels

Throughout this student workbook, amounts of time will be referred to in units of seconds (s), milliseconds (ms), and microseconds (μ s). Seconds are abbreviated with the lower-case letter s. So, one second is written as 1 s. Milliseconds are abbreviated as ms, and it means one one-thousandth of a second. One microsecond is one one-millionth of a second. The Milliseconds and Microseconds box to the right shows these equalities in terms of both fractions and scientific notation. A voltage level is measured in volts, which is abbreviated with an upper case V. The BOE has sockets labeled Vss, Vdd, and Vin. Vss is called the system ground or reference voltage. When the battery pack is plugged in, Vss is connected to its negative terminal. As far as the BOE, BASIC Stamp and serial connections to the computer are concerned, Vss is always 0 V. Vin is unregulated 6 V, and it's connected to the positive terminal of the battery pack. Vdd is regulated to 5 V by the BOE's onboard voltage regulator, and it will be used with Vss to supply power to circuits built on the BOE's breadboard.

Milliseconds and Microseconds	
$1 \text{ ms} = \frac{1}{1000} \text{ s} = 1 \times 10^{-3} \text{ s}$	
$1 \mu\text{s} = \frac{1}{1,000,000} \text{ s} = 1 \times 10^{-6} \text{ s}$	
Voltages and BOE Labels	
Vss = 0 V (ground)	
Vdd = 5 V (regulated)	
Vin = 6 V (unregulated)	

The control signal the BASIC Stamp sends to the servo's control line is called a "pulse train," and an example of one is shown in Figure 2. The BASIC Stamp can be programmed to produce this waveform using any of its I/O pins. In this activity, we'll start with I/O pin P12, which is already connected to servo port 12 by a metal trace built into the Board of Education. First, the BASIC Stamp sets the voltage at P12 to 0 V (low) for 20 ms. Then, it sets the voltage at P12 to 5 V (high) for 1.0 ms. Then, and it starts over with a low output for another 20 ms, and a high output for another 1.0 ms, and so on.

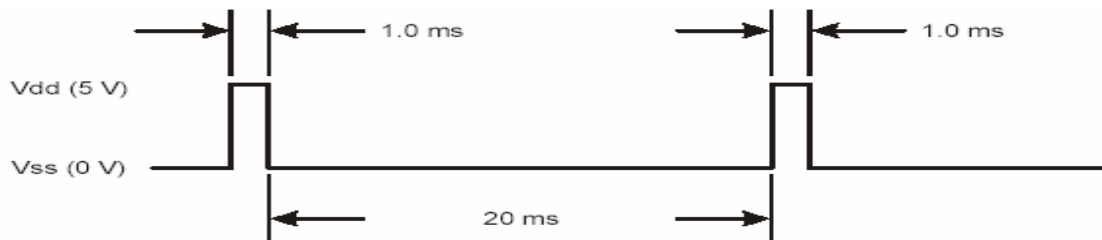


Figure 2: Pulse Train

This pulse train has 1.0 ms high time and a 20 ms low time. The high time is the main ingredient for controlling a servo's motion, and it is most commonly referred to as the pulse width. In this example, we are working with 1 ms wide pulses. Since these pulses go from low to high (0 V to 5 V) for a certain amount of time, they are called positive pulses. Negative pulses would involve a resting state that's high with pulses that drop low. Pulse trains have some other technical descriptions such as duty and duty cycle.

A pre-modified servo can be pulsed to make its output shaft turn continuously. The pulse widths for pre-modified servos range between 1.0 and 2.0 ms for full speed clockwise and counterclockwise respectively. If you give a pre-modified servo 1.25 ms pulses, it will turn clockwise at roughly half of full speed. If you give a pre-modified servo 1.90 ms pulses, the servo will turn at almost full speed counterclockwise. The "center pulse width" is 1.5 ms, and that makes the servo stay still. If the servo turns very slowly in response to 1.5 ms pulse.

Figure 3 shows the Boe-Bot's front, back, left and right. Use this diagram as your guide when you see instructions about making the Boe-Bot moves forward/backward, examining the right or left wheels, etc.

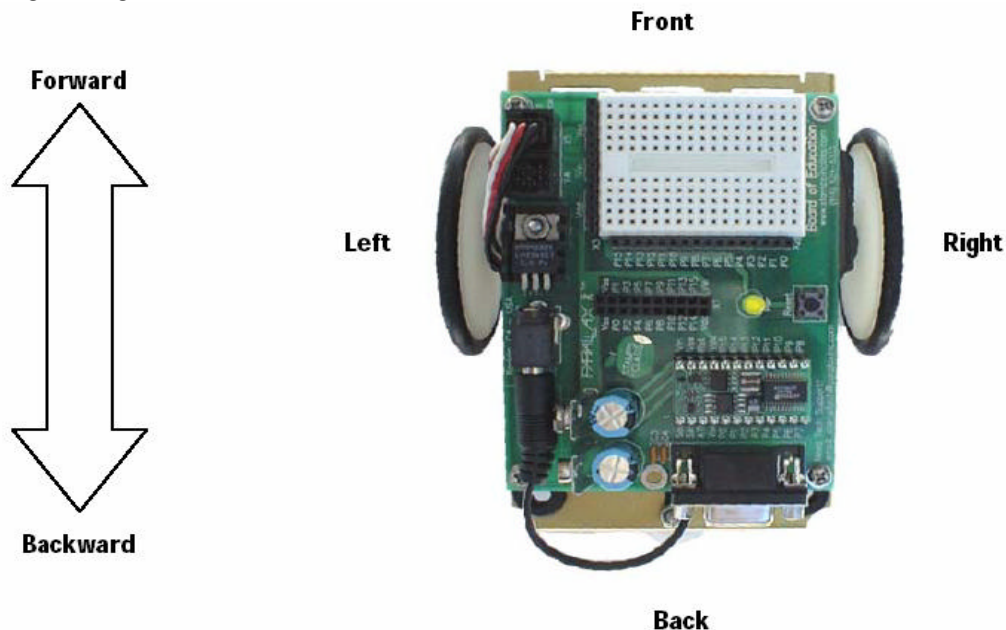


Figure 3: Boe-Bot from the Driver's Seat

Let's start by programming the Boe-Bot's right wheel to turn full speed ahead. For the right side of the Boe-Bot, this means the wheel has to turn clockwise, which means it needs to receive 1 ms pulses every 20 ms or so.

1. You may want to set the Boe-Bot on something to keep it's wheels from touching the ground during these tests. Otherwise, you will see the Boe-Bot spin around in circles since only one wheel is turning.
2. Enter Program Listing 1.2 into the Stamp Editor.

```
' Robotics! v1.5, Program Listing 1.2: Right Wheel Full Speed Ahead.
' {$Stamp bs2}                                ' Stamp Directive.

low 12                                          ' Set P12 to output-low.

loop:                                         ' Target label for last command - "goto loop".

    pulsout 12, 500                           ' Send 1.0 ms pulses to P12
    pause 20                                 ' every 20 ms.

goto loop                                     ' Send program to "loop:" label.
```

1. Save the program using a descriptive name, such as "Boe-Bot 1.2". You can do this by clicking the File Menu and selecting Save (or Save As... if you are renaming the file). Then enter the "Boe-Bot 1.2" into the File name: field, and make sure that the Save as type: field is set to "BASIC Stamp 2 files (*.bs2)".
2. Run the program by clicking Run and selecting Run.
3. Verify that, as you're looking at the wheel from the Boe-Bot's right side that it is turning clockwise fairly rapidly (about 37 RPM).

How the Right Wheel Full Speed Ahead Program Works

- Look up each of the following new commands in Appendix C: PBASIC Quick Reference or in the BASIC Stamp Manual before continuing: **low**, **pulsout**, **pause**, **goto**.

As with the previous program example, the first line of the program is a descriptive comment and the Stamp Directive is on the second line. In this program there are also comments (that begin with apostrophes) to the right of each PBASIC command. With the exception of the Stamp Directive, when entering the commands into the Stamp Editor, you don't need to include the comments. Aside from the Stamp Directive, comments and other ways of documenting your programs become important later on if you start writing code that's more complex or that somebody else needs to work with.

The command **low 12** does two things. It sets BASIC Stamp I/O pin P12 to output, then it sets its output value low. As an output, P12 can send voltage signals, as opposed to being set to **input**, which means P12 would listen for signals instead. Setting the output value low means that the voltage P12 sends is the same as Vss, 0 volts. If the command **high 12** were used instead, P12 would send a high signal, which would be Vdd, 5 volts.

When a word that's not a PBASIC command is followed by a colon, it's called a "label". As you get more and more familiar with PBASIC, you'll start to recognize which words are commands and which words are labels automatically. The **loop:** label works together with the command **goto loop** at the end of the program. The **loop:** label is a place holder, and whenever the program gets to the command **goto loop**, it causes the program to start executing commands at the **loop:** label again. The result is that the **pulsout** and **pause** commands get executed over and over, which causes a continuous pulse train to be sent to the servo.

The **loop:...goto loop** program structure is called an infinite loop. An infinite loop means that part of the code in the program is executed over and over again with no code that allows it to stop repeating the same set of instructions. Often with normal computer programs, this is a problem. However, infinite loops are commonly used in microcontroller programming. In fact, most microcontroller programs, including the ones in this text, are written within the framework of an infinite loop. With the BASIC Stamp, you can always end an infinite loop by disconnecting the power or using the Stamp Editor to run a different program.

The command **pulsout 12, 500** sends a 1.0 ms pulse to P12. The **pulsout** command has two arguments, **pin** and **period**. The **pin** argument refers to the I/O pin the BASIC Stamp sends the voltage pulse to, and **period** refers to how long the voltage pulse lasts. The I/O pin number makes sense; the number 12 refers to I/O pin P12. What about the **period** argument of **500**? How does that correspond to a 1.0 ms pulse? The answer is that the **period** argument for the **pulsout** command has to be specified in 2 μ s increments. So, if you want the pulse to last for 1.0 ms, you have to use a number that gives you 1.0 ms when it's multiplied by 2 μ s increments. Here is proof that a **pulsout period** of **500** fits the bill.

$$\begin{aligned} 500 \times 2 \mu s &= 500 \times (2 \times 10^{-6}) s \\ &= 1000 \times 10^{-6} s \\ &= 1.0 \times 10^{-3} s \\ &= 1.0 ms \end{aligned}$$

The command **pause 20** is much more obvious. That's because the **period** argument for the **pause** command is specified in 1 ms increments. So, if you want a 20 ms pause, **pause 20** does the job.

Your Turn

1. Write a program to make the right wheel turn full speed counterclockwise.
2. Write a program to make the left wheel turn full speed counterclockwise
3. Write a program to make the right wheel turn full speed clockwise.

Running Both Servos

When you assembled the Boe-Bot in Activity #2, you plugged the servo on the right side of the Boe-Bot into P12 and the servo on the left side of the Boe-Bot into P13. Figure 4, shows a schematic of the circuit you created by doing this. The servo on the Boe-Bot's right side is connected to I/O line P12 and the servo on the Boe-Bot's left is connected to P13. Each servo is also connected to Vin (the battery pack's positive terminal) and Vss (the battery pack's negative terminal).

The easy part about making the Boe-Bot roll forward is that you include two **pulsout** commands, one for each servo. The difficult part can be figuring out what the **period** arguments should be.

Take a look at the right side of the Boe-Bot. To make this wheel turn forward, the servo has to turn clockwise. This means a **period** argument less than center. A **period** of 500 will work well for full speed ahead. Now look at the left side of the Boe-Bot. To make this wheel turn forward, the servo has to turn counterclockwise. Now instead of 500, a **period** of 1000 is needed.

Enter and run Program Listing 1.5 and observe the results.

If the Boe-Bot rolled backward instead of forward, the servo lines were swapped. It means that the servo plugged into servo port 12 should be plugged into servo port 13 and visa-versa.

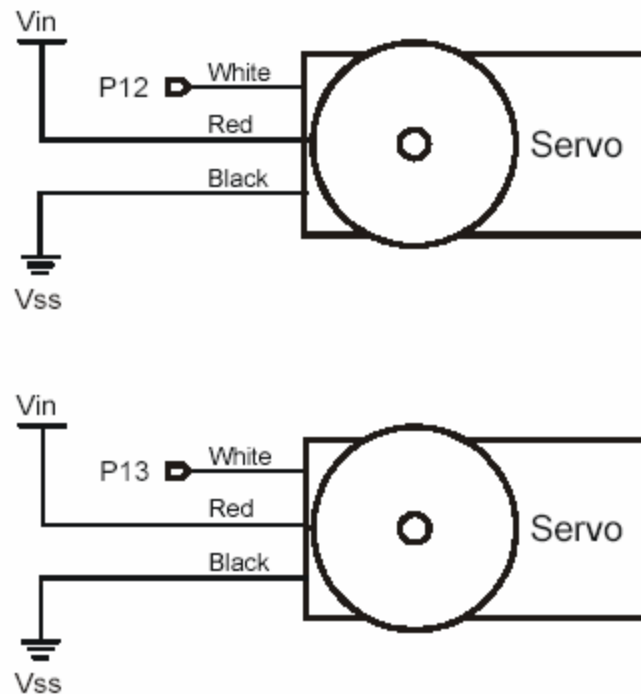


Figure 4: Servo connection schematic

```
' Robotics! v1.5, Program Listing 1.5: Full Speed Ahead - both servos.
' {$Stamp bs2}                ' Stamp Directive.

low 12                        ' Set P12 to output-low.
low 13                        ' Set P13 to output-low.

loop:                         ' Label for "goto loop to return to"

  pulsout 12, 500              ' Send 1.0 ms pulses to P12
  pulsout 13, 1000             ' Send 2.0 ms pulses to P13
  pause 20                     ' every 20 ms.

goto loop                     ' Send program to "loop: " label.
```

Your Turn

After you make each change listed below, make sure to run the modified version of the program and observe what the Boe-Bot does differently.

1. Swap the **pulsout *period*** arguments to make the Boe-Bot to roll backward. In other words, instead of using the commands **pulsout, 12, 500** and **pulsout 13, 1000**, use the commands **pulsout 12,1000**, and **pulsout 13, 500**. This should make the Boe-Bot travel backward instead of forward.
2. Try setting both **pulsout *period*** arguments to the center value of 750 to make the Boe-Bot stay still.
3. Try setting both **pulsout *period*** arguments to 500 and run the modified program. It will make the Boe-Bot rotate counterclockwise in place.
4. Try setting both **pulsout *period*** arguments to 1000 and run the modified program. It will make the Boe-Bot rotate clockwise in place.

So far, the Boe-Bot can be programmed to roll forward or backward or to rotate in place. During some of your testing, small variations in servo performance were discovered and corrected in software.