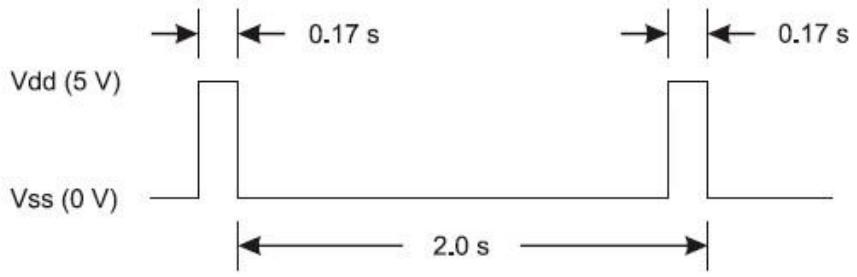
Lab 3: LED Servo Signal Monitors

The high and low signals that control servo motors must last for very precise periods of time. That's because a servo motor measures how long the signal stays high, and uses that as an instruction for how fast, and in which direction, to turn its motor.

This timing diagram shows a servo signal that would make your Shield-Bot's wheel turn full speed counterclockwise. There's one big difference though: all the signals in this timing diagram last 100 times longer than they would if they were controlling a servo. This slows it down enough so that we can see what's going on.



Example Sketch: ServoSlowMoCcw

- ✓ Enter, save, and upload ServoSlowMoCcw to the Arduino.
- ✓ Verify that the pin 13 LED circuit pulses briefly every two seconds.

```
/*
    Robotics with the BOE Shield - ServoSlowMoCcw
    Send 1/100th speed servo signals for viewing with an LED.
 */

void setup()                                    // Built in initialization block
{
  pinMode(13, OUTPUT);                          // Set digital pin 13 -> output
}

void loop()                                     // Main loop auto-repeats
{
  digitalWrite(13, HIGH);                    // Pin 13 = 5 V, LED emits light
  delay(170);                                  // ..for 0.17 seconds
  digitalWrite(13, LOW);                     / Pin 13 = 0 V, LED no light
  delay(1830);                                 // ..for 1.83 seconds
}
```

Your Turn – Two Steps to Servo Signal

Alright, how about 1/10th speed instead of 1/100th speed?

- ✓ Reduce `delay(170)` to `delay(17)`, and `delay(1830)` to `delay(183)`, and re-upload the sketch.

Is the LED blinking 10 times faster now? Divide by 10 again for a full speed servo signal—we'll have to round the numbers a bit:

- Change `delay(17)` to `delay(2)`, and `delay(183)` to `delay(18)`, then upload the modified sketch.

Now you can see what the servo signal looks like with the indicator LED. The LED is flickering so fast, it's just a glow. Since the high signal is 2 ms instead of 1.7 ms, it'll be a little brighter than the actual servo control signal—the light is spending more time on. We could use this signal and programming technique to control a servo, but there's an easier, more precise way. Let's try it with LEDs first.

How to Use the Arduino Servo Library

A better way to generate servo control signals is to include the Arduino Servo library in your sketch, one of the standard libraries of pre-written code bundled with the Arduino software.

- To see a list of Arduino libraries, click the Arduino software's Help menu and select Reference.
- Find and follow the Libraries link.

We want to take a closer look at the Servo library.

- Find and follow the Servo link.
- Follow and read the links for these functions on the Servo library page:

```
attach() writeMicroseconds()

detach()
```

Servos have to receive high-pulse control signals at regular intervals to keep turning. If the signal stops, so does the servo. Once your sketch uses the Servo library to set up the signal, it can move on to other code, like delays, checking sensors, etc. Meanwhile, the servo keeps turning because the Servo library keeps running in the background. It regularly interrupts the execution of other code to initiate those high pulses, doing it so quickly that it's practically unnoticeable.

Using the Servo library to send servo control signals takes four steps:

1. Tell the Arduino editor that you want access to the Servo library functions with this declaration at the start of your sketch, before the `setup` function.

```
#include <Servo.h>          // Include servo library
```

2. Declare and name an instance of the Servo library for each signal you want to send, between the `#include` and the `setup` function.

```
Servo servoLeft;            // Declare left servo
```

3. In the `setup` function, use the name you gave the servo signal followed by a dot, and then the attach function call to attach the signal pin. This example is telling the system that the servo signal named `servoLeft` should be transmitted by digital pin 13.

```
servoLeft.attach(13);      // Attach left signal to pin 13
```

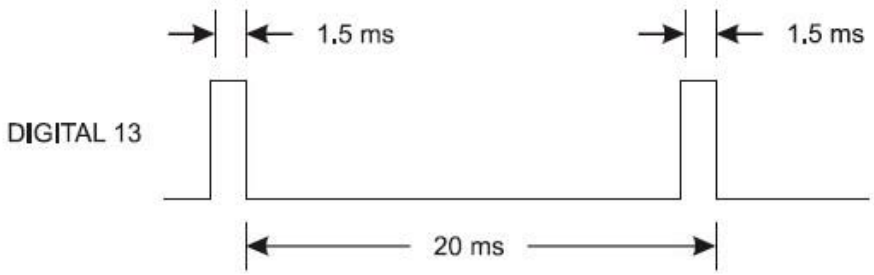
4. Use the `writeMicroseconds` function to set the pulse time. You can do this inside either the `setup` or `loop` function:

```
servoLeft.writeMicroseconds(1500); // 1.5 ms stay-still signal
```

Seconds, Milliseconds, Microseconds
A millisecond is a one-thousandth of a second, abbreviated ms.
A microsecond is a one-millionth of a second, abbreviated µs.
There are 1000 microseconds (µs) in 1 millisecond (ms). There are 1,000,000 microseconds in 1 second (s).

Example Sketch: LeftServoStayStill

For calibrating servos, your sketch will need to send signals with 1.5 ms pulses. Take a look at the timing diagram below. This stay-still signal's high pulses last 1.5 ms. That's halfway between the 1.7 ms full-speed-counterclockwise and 1.3 ms full-speed-clockwise pulses.



- ✓ Enter, save and upload LeftServoStayStill to your Arduino. The pin 13 LED should glow, about halfway between the two brightness levels you observed earlier.

```
/*
 Robotics with the BOE Shield – LeftServoStayStill
```

```
 Generate signal to make the servo stay still for centering.  */

#include <Servo.h>                    // Include servo library

Servo servoLeft;                              // Declare left servo

void setup()                                  // Built in initialization block
{
  servoLeft.attach(13);                       // Attach left signal to pin 13
servoLeft.writeMicroseconds(1500);          // 1.5 ms stay still signal
}

void loop()                                   // Main loop auto-repeats
{                                             // Empty, nothing needs repeating
}
```

Your Turn – Check a Second Control Signal with the Pin 12 LED

You'll be using this code a lot, so it's a good idea to practice declaring an instance of Servo, attaching the signal to a pin, and setting the pulse duration.

- Save LeftServoStayStill as BothServosStayStill.
- Add a second Servo declaration and name it `servoRight`.

```
    Servo servoRight;             // Declare right servo
```

- Attach your `servoRight` signal to digital pin 12.

```
    servoRight.attach(12);        // Attach right signal to pin 12
```

- Set the `servoRight` signal for 1.5 ms (1500 µs) pulses.

```
    servoRight.writeMicroseconds(1500); // 1.5 ms stay still signal
```

- Your sketch should now look like BothServosStayStill.
- Save the sketch and upload it to your Arduino.
- Verify that both LEDs are at a similar brightness level.

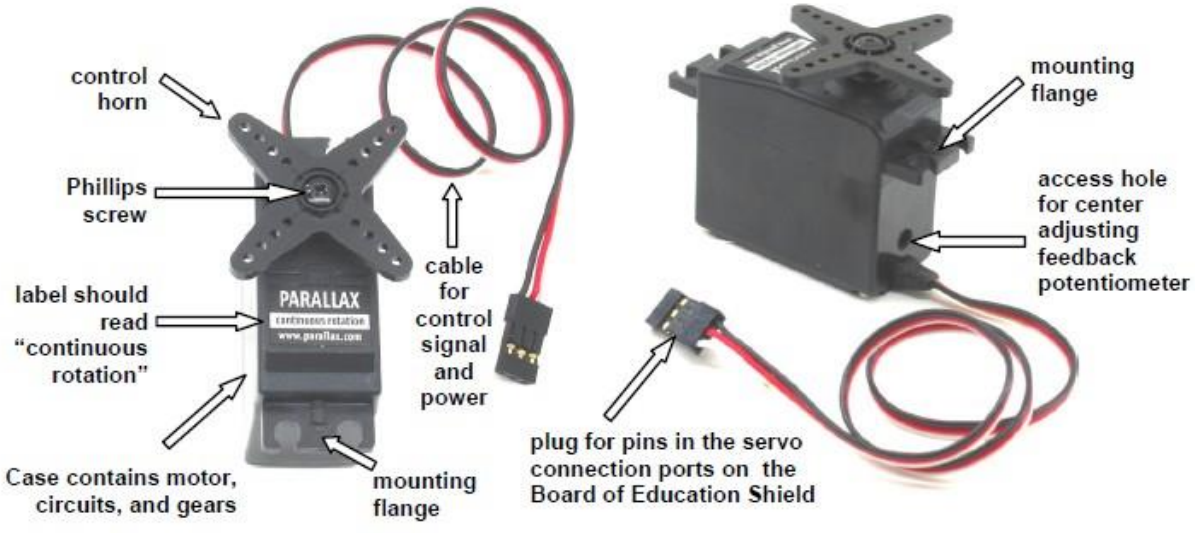
```
/*
 Robotics with the BOE Shield – BothServosStayStill
 Generate signals to make the servos stay still for centering.  */

#include <Servo.h>                            // Include servo library

Servo servoLeft;                           // Declare left servo signal S

Servo servoRight;                             // Declare right servo signal

void setup()                                  // Built in initialization block
{
 servoLeft.attach(13);                       // Attach left signal to pin 13
servoRight.attach(12);                       // Attach left signal to pin 12
```

```
  servoLeft.writeMicroseconds(1500);          // 1.5 ms stay still sig, pin 13
servoRight.writeMicroseconds(1500);         // 1.5 ms stay still sig, pin 12
}
```

```
void loop()                                   // Main loop auto-repeats

{                                             // Empty, nothing needs repeating

}
```

Connect Servo Motors and Batteries

From the robot navigation standpoint, continuous rotation servos offer a great combination of simplicity, usefulness and low price. The Parallax continuous rotation servos are the motors that will make the BOE Shield-Bot's wheels turn, under Arduino control.



In this activity, you will connect your servos to the Board of Education Shield's servo ports, which will connect them to supply voltage, ground, and a signal pin. You will also connect a battery supply to your Arduino because, under certain conditions, servos can end up demanding more current than a USB supply is designed to deliver.

Standard Servos vs. Continuous Rotation Servos
Standard servos are designed to receive electronic signals that tell them what position to hold. These servos control the positions of radio controlled airplane flaps, boat rudders, and car steering. Continuous rotation servos receive the same electronic signals, but instead turn at certain speeds and directions. Continuous rotation servos are handy for controlling wheels and pulleys.

Servo Control Horn, 4-point Star vs. Round
It doesn't make a difference. So long as it is labeled "continuous rotation" it's the servo for your BOE Shield-Bot. You'll remove the control horn and replace it with a wheel.

Connect the Servos to the BOE Shield

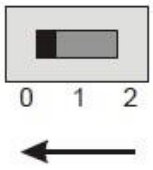
Leave the LED circuits from the last activity on your board. They will be used later to monitor the signals the Arduino sends to the servos to control their motion.

Parts List

(2) Parallax continuous rotation servos
BOE Shield with built and tested LED indicator circuits from the previous activity

Instructions

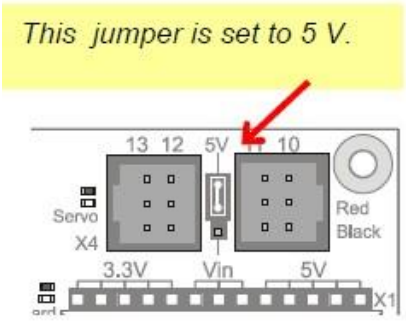
- ✓ Set your Shield's power switch to position-0.

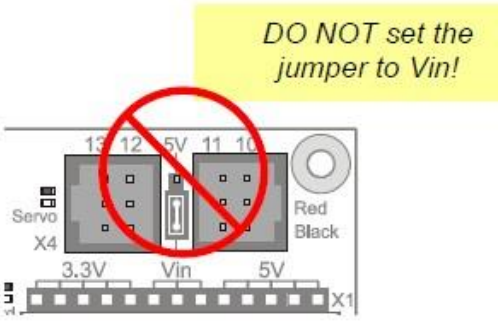


- ✓ Disconnect all sources of power from the Arduino including the USB cable.

Between the servo headers on the BOE Shield is a jumper that connects the servo power supply to either Vin or 5V. To move it, pull it upwards and off the pair of pins it covers, then push it onto the pair of pins you want it to rest on. The BOE Shield-Bot's battery pack will supply 7.5 V. Since the servos are rated for 4–6 V, we want to make sure the jumper is set to 5V. Also, a steady 5 V voltage supply will support a consistent servo speed, and more accurate navigation, than voltage that varies as batteries discharge.

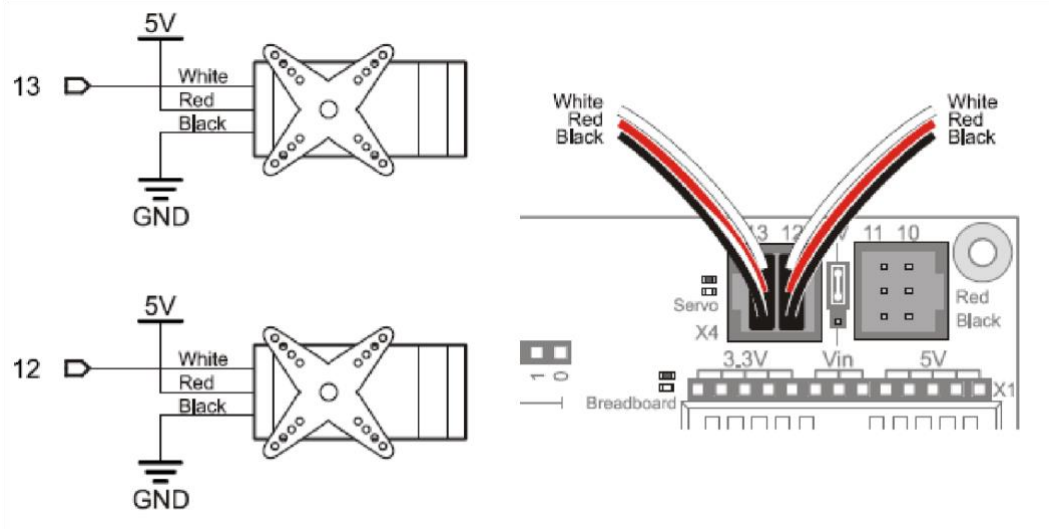
- ✓ Make sure your BOE Shield's power jumper is set to 5V; if not, set it now.





The picture below shows the schematic of the circuit you create by plugging the servos into ports 13 and 12 on the BOE Shield. Pay careful attention to wire color as you plug in the cables: the black wire should be at the bottom, and the white one should be at the top.

- ✓ Connect your servos to your BOE Shield as shown in the diagram below. The left servo connects to port 13 and the right servo connects to port 12.
- ✓ Make sure that you follow the cable colors shown in the figure, with the black wire closer to the breadboard and the white wire closer to the board's edge.



Connect the Battery Pack to the BOE Shield

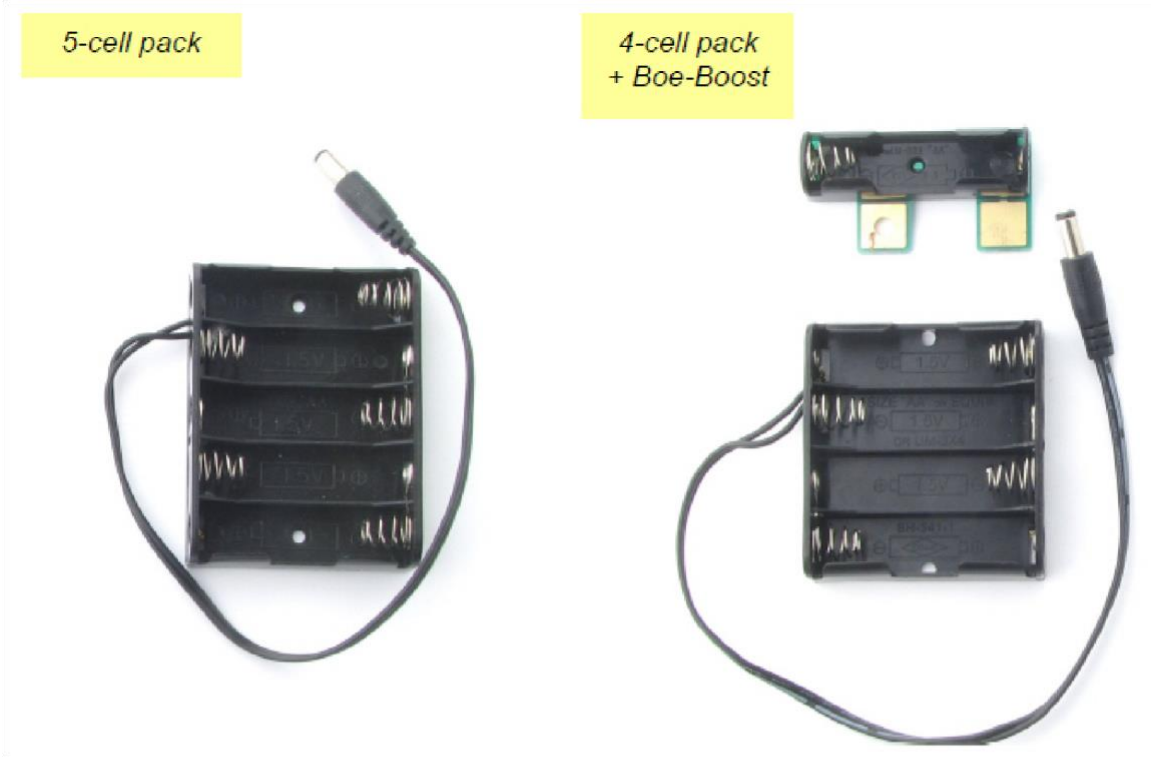
To properly power the servos, you'll need to switch to an external battery pack now. When servos make sudden direction changes or push against resistance to rotation, they can draw more current than a USB port is designed to supply. Also, it would be no fun for the BOE Shield-Bot to be tethered to the computer forever! So, from here on out we'll be using an external battery pack with five 1.5 V AA batteries. This will supply your system with 7.5 V and plenty of current for the voltage regulators and servos. From here forward, remember two things:

- ALWAYS unplug the battery pack when you are done experimenting for a while. Even when the power switch on your BOE Shield is off (position-0), the Arduino module will still draw power from the batteries.
- Unplug the programming cable too, whenever you unplug the battery pack. That way, you won't accidentally try to run the servos off of USB power.

Which Battery Pack Do You Have?

Compare your parts to the picture below to see which battery pack you have.

For the 5-cell Pack, keep going on the next page.



Rechargeable Options
The thrifty Boe-Boost (#30078) allows you to add another cell in series with a 4-cell or 5-cell pack. Adding a 6th 1.2 V AA rechargeable cell to a 5-cell pack will supply 6 x 1.2 = 7.2 V.
The Li-ion Boe-Bot Power Pack-Charger (#28988) combines a lithium-ion battery pack and recharger in one board that you can mount under your Shield-Bot.

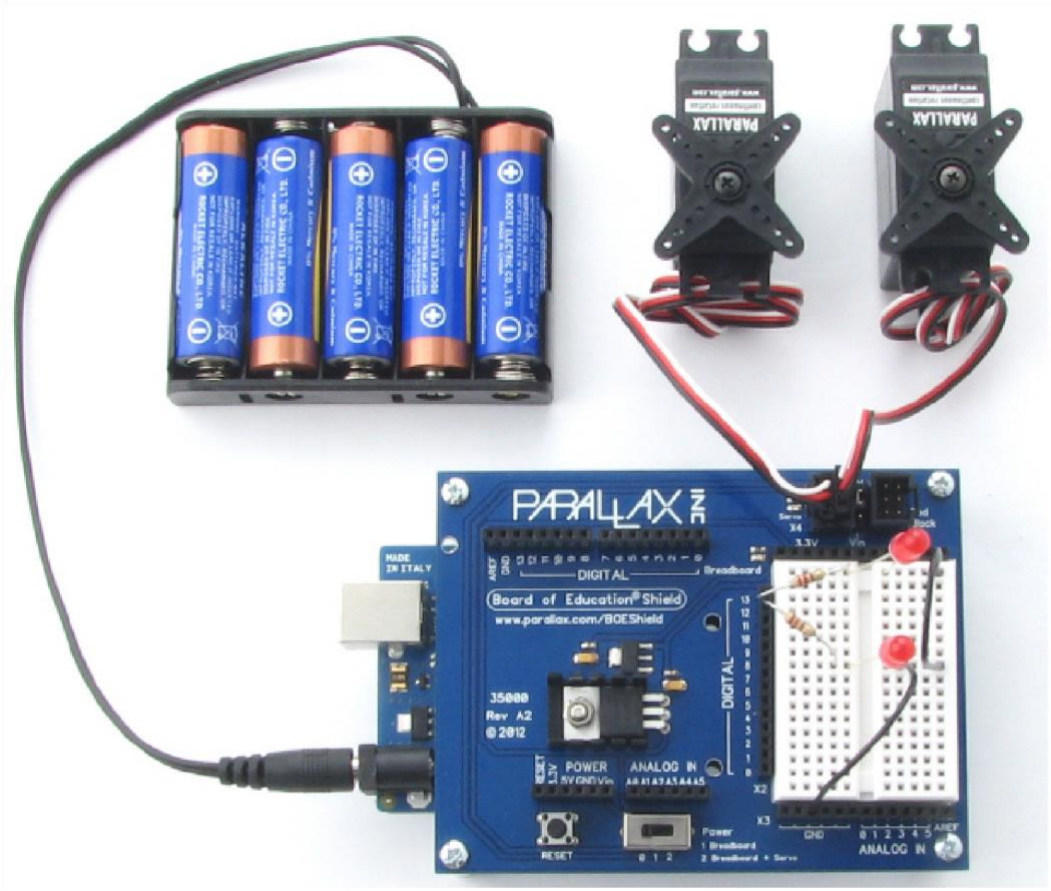
CAUTION: AC powered DC supplies are not recommended for the BOE Shield-Bot.
Some DC supplies provide much higher voltage than their rating. The BOE Shield-bot is designed for use with a 7.2–7.5 V battery supply. It will work with higher supply voltages at low loads, but the servo loads can heat up the regulator until it shuts off to protect itself.

5-cell Pack Setup

Parts List

(5) AA alkaline batteries
(1) 5-cell battery pack

- ✓ Load the batteries into the battery pack.
- ✓ Plug the battery pack into the Arduino's power jack. When you are done, it should resemble the picture below.
- ✓ Skip to Centering the Servos [18].



Centering the Servos

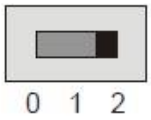
In this activity, you will run a sketch that sends the “stay-still” signal to the servos. You will then use a screwdriver to adjust the servos so that they actually stay still. This is called centering the servos. After the adjustment, you will run test sketches that will turn the servos clockwise and counterclockwise at various speeds. Tool Required

You’ll need a Phillips #1 point screwdriver with a 1/8″ (3.18 mm) or smaller shaft.

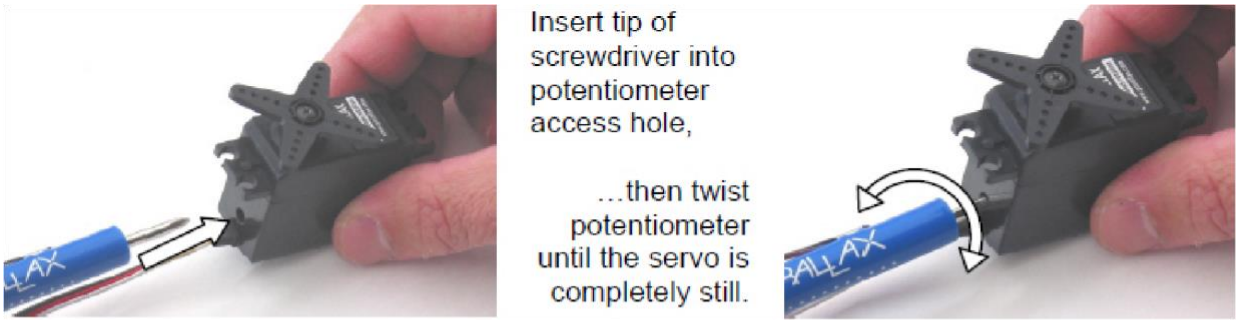
Sending the Center Signals

If a servo has not yet been centered, it may turn, vibrate, or make a humming noise when it receives the “stay-still” signal.

- ✓ Reconnect your programming cable, and re-run LeftServoStayStill.
- ✓ Set the BOE Shield's Power switch to 2, to provide power to the servos.



- ✓ Use a screwdriver to gently adjust the potentiometer in the servo as shown in Figure 2-26. Don't push too hard! Adjust the potentiometer slightly until you find the setting that makes the servo stop turning, humming or vibrating.



- ✓ Verify that the pin 13 LED signal monitor circuit is showing activity. It should glow like it did when you ran LeftServoStayStill the first time.

What's a Potentiometer?
A *potentiometer* is kind of like an adjustable resistor with a moving part, such as a knob or a sliding bar, for setting the resistance. The Parallax continuous rotation servo's potentiometer is a recessed knob that can be adjusted with a small Phillips screwdriver tip. Learn more about potentiometers in *What's a Microcontroller?* and *Basic Analog and Digital* at www.parallax.com. [19]

Your Turn – Center the Servo Connected to Pin 12

- ✓ Repeat the process for the pin 12 servo using the sketch RightServoStayStill.

```
/*
 Robotics with the BOE Shield - RightServoStayStill
 Transmit the center or stay still signal on pin 12 for center adjustment.  */

#include <Servo.h>                          // Include servo library

Servo servoRight;                           // Declare right servo

void setup()                                // Built-in initialization block
{
servoRight.attach(12);                     // Attach right signal to pin 12
servoRight.writeMicroseconds(1500);        // 1.5 ms stay still signal
}
```

```
void loop()                                    // Main loop auto-repeats

{                                              // Empty, nothing needs repeating

}
```

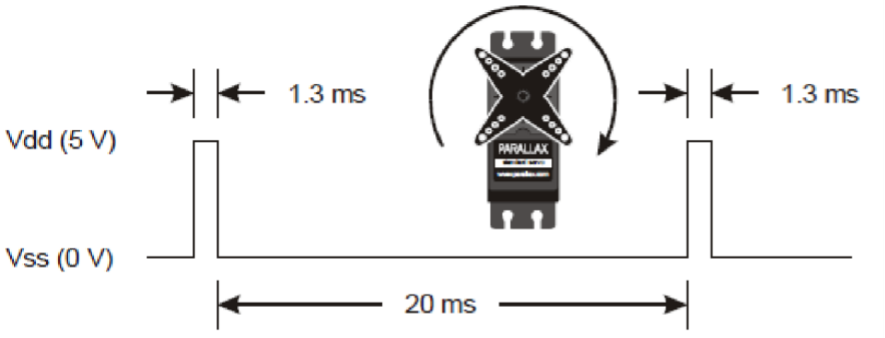
Testing the Servos

There's one last thing to do before assembling your BOE Shield-Bot, and that's testing the servos. In this activity, you will run sketches that make the servos turn at different speeds and directions. This is an example of subsystem testing—a good habit to develop.

> Subsystem testing is the practice of testing the individual components before they go into the larger device. It's a valuable strategy that can help you win robotics contests. It's also an essential skill used by engineers to develop everything from toys, cars, and video games to space shuttles and Mars roving robots. Especially in more complex devices, it can become nearly impossible to figure out a problem if
>
> the individual components haven't been tested beforehand. In aerospace projects, for example, disassembling a prototype to fix a problem can cost hundreds of thousands, or even millions, of dollars. In those kinds of projects, subsystem testing is rigorous and thorough.

Pulse Width Controls Speed and Direction

This timing diagram shows how a Parallax continuous rotation servo turns full speed clockwise when you send it 1.3 ms pulses. Full speed typically falls in the 50 to 60 RPM range.



> **What's RPM?** Revolutions Per Minute—the number of full rotations turned in one minute.
>
> **What's a pulse train?** Just as a railroad train is a series of cars, a *pulse train* is a series of pulses (brief high signals).

Example Sketch: LeftServoClockwise

Enter, save, and upload LeftServoClockwise.

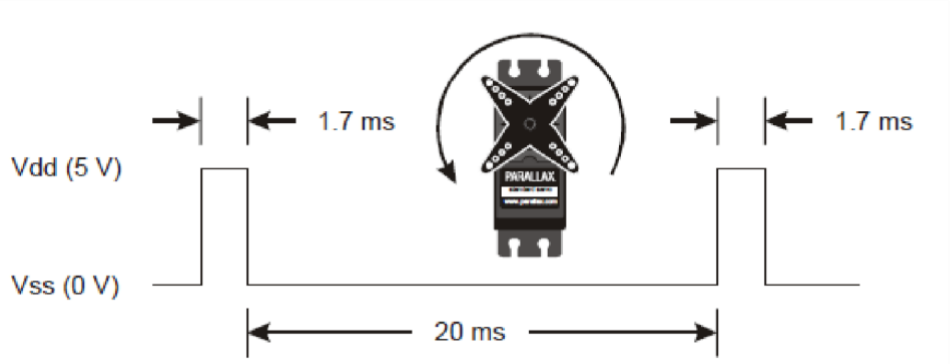
Verify that the servo's horn is rotating between 50 and 60 RPM clockwise.

```
/*
 Robotics with the BOE Shield – LeftServoClockwise
 Generate a servo full speed clockwise signal on digital pin 13.  */

#include <Servo.h>                            // Include servo library

Servo servoLeft;                              // Declare left servo

void setup()                                  // Built in initialization block
{
  servoLeft.attach(13);                       // Attach left signal to pin 13
servoLeft.writeMicroseconds(1300);          // 1.3 ms full speed clockwise
}

void loop()                                   // Main loop auto-repeats
{                                             // Empty, nothing needs repeating

}
```

Your Turn: Left Servo Counterclockwise

- Save LeftServoClockwise as LeftServoCounterclockwise.

- In **`servoLeft.writeMicroseconds`**, change **`(1300)`** to **`(1700)`**.
- Save the modified sketch and upload it to the Arduino.
- Verify that the servo connected to pin 13 now rotates the other direction, which should be counterclockwise, at about 50 to 60 RPM.



Example Sketch: RightServoClockwise

- Save LeftServoClockwise as RightServoClockwise.
- Replace all instances of `servoLeft` with `servoRight`.
- Replace all instance of `13` with `12`.
- Run the sketch and verify that the pin 12 servo is rotating between 50 and 60 RPM clockwise.

```
/*
 Robotics with the BOE Shield - RightServoClockwise
 Generate a servo full speed clockwise signal on digital pin 12.  */

#include <Servo.h>                            // Include servo library

Servo servoRight;                             // Declare left servo

void setup()                                // Built in initialization block
{
  servoRight.attach(12);                      // Attach left signal to pin 12
servoRight.writeMicroseconds(1300);
                                      // 1.3 ms full speed clockwise
}

void loop()                                   // Main loop auto-repeats
{                                       // Empty, nothing needs repeating
}
```

Your Turn – Right Servo Counterclockwise

- In `servoRight.writeMicroseconds` change `(1300)` to `(1700)`.
- Save the sketch and upload it to your Arduino.
- Verify that the pin 12 servo turns full-speed counterclockwise, about 50 to 60 RPM.

Controlling Servo Speed and Direction

For BOE Shield-Bot navigation, we need to control both servos at once.

- ✓ Enter, save, and upload ServosOppositeDirections to the Arduino.
- ✓ Verify that the servo connected to pin 13 turns counterclockwise and the one connected to pin 12 turns clockwise. Example Sketch: ServosOppositeDirections

```
/*
Robotics with the BOE Shield – ServosOppositeDirections
Generate a servo full speed counterclockwise signal with pin 13 and
full speed clockwise signal with pin 12.  */

#include <Servo.h>                            // Include servo library

Servo servoLeft;                       // Declare left servo signal Servo servoRight;
// Declare right servo signal

void setup()                                  // Built in initialization block
{
  servoLeft.attach(13);                    // Attach left signal to pin 13
servoRight.attach(12);                    // Attach right signal to pin 12

  servoLeft.writeMicroseconds(1700);       // 1.7 ms -> counterclockwise
servoRight.writeMicroseconds(1300);        // 1.3 ms -> clockwise
}

void loop()                                   // Main loop auto-repeats
{                                      // Empty, nothing needs repeating

}
```

This opposite-direction control will be important soon. Think about it: when the servos are mounted on either side of a chassis, one will have to rotate clockwise while the other rotates counterclockwise to make the BOE Shield-Bot roll in a straight line. Does that seem odd? If you can't picture it, try this:

- ✓ Hold your servos together back-to-back while the sketch is running.

> Pulse Width Modulation
> Adjusting the property of a signal to carry information is called modulation. We've discovered that servo control signals are a series of high pulses separated by low resting states. How long the high pulse lasts—how wide the high pulse looks in a timing diagram—determines the speed and direction that the servo turns. That adjustable pulse width carries the servo setting information. Therefore, we can say that servos are controlled with pulse width modulation.

Different combinations of `writeMicroseconds us` parameters will be used repeatedly for programming your BOE Shield-Bot's motion. By testing several possible combinations and filling in the Description column of Table 2-2, you will become familiar with them and build a reference for yourself. You'll fill in the Behavior column later on, when you see how the combinations make your assembled BOE Shield-Bot move.

- ✓ Save a copy of the ServosOppositeDirections sketch as TestServosTable2_2.
- ✓ Test each combination of values in `servoLeft.WriteMicroseconds(us)` and `servoRight.writeMicroseconds(us)` and record your results in a table. Upload this table with your sketches in this weeks lab upload area.

Table 2-2: writeMicroseconds(us) Combinations

Pin 13 servoLeft	Pin 12 servoRight	Description	Behavior
1700	1300	Full speed, pin 13 servo counter-clockwise, pin 12 servo clockwise.	
1300	1700		
1700	1700		
1300	1300		
1500	1700		
1300	1500		
1500	1500	Both servos should stay still (see Activity #5, page 64.)	
1520	1480		
1540	1460		
1700	1450		
1550	1300		

How To Control Servo Run Time

It's easy to control how long the servos run when using the Servo library. Once set, a servo will maintain its motion until it receives a new setting. So, to make a servo run for a certain length of time, all you have to do is insert a `delay` after each setting.

Example Sketch: ServoRunTimes

- ✓ Enter, save, and upload ServoRunTimes to your Arduino.
- ✓ Verify that both servos turn full speed clockwise for 3 seconds, then counterclockwise for 3 seconds, then stop.

```
/*
 Robotics with the BOE Shield – ServoRunTimes
 Generate a servo full speed counterclockwise signal with pin 13 and
full speed clockwise signal with pin 12.  */

#include <Servo.h>                              // Include servo library
```

```
Servo servoLeft;                          // Declare left servo signal Servo
servoRight;                           // Declare right servo signal

void setup()                              // Built in initialization block
{
servoLeft.attach(13);                    // Attach left signal to pin 13
servoRight.attach(12);                    // Attach right signal to pin 12

servoLeft.writeMicroseconds(1300);          // Pin 13 clockwise

servoRight.writeMicroseconds(1300);        // Pin 12 clockwise   delay(3000);
// ..for 3 seconds   servoLeft.writeMicroseconds(1700);
                                          // Pin 13 counterclockwise


servoRight.writeMicroseconds(1700);
                                          // Pin 12 counterclockwise
delay(3000);                                 // ..for 3 seconds
servoLeft.writeMicroseconds(1500);           // Pin 13 stay still
servoRight.writeMicroseconds(1500);          // Pin 12 stay still
}

void loop()                                 // Main loop auto-repeats


{                                       // Empty, nothing needs repeating


}
```

Assemble and Test your BOE Shield-Bot

This chapter contains instructions for building and testing your BOE Shield-Bot. It's especially important to complete the testing portion before moving on to the next chapter. By doing so, you can help avoid a number of common mistakes that could otherwise lead to mystifying BOE Shield-Bot behavior. Here is a summary of what you will do:

1. Build the BOE Shield-Bot.
2. Re-test the servos to make sure they are properly connected.
3. Connect and test a speaker that can let you know when the BOE Shield-Bot's batteries are running low.

Assembling the BOE-Shield-Bot

This activity will guide you through assembling the BOE Shield-Bot, step by step. In each step, you will gather a few of the parts, and then assemble them so that they match the pictures. Each picture has instructions that go with it; make sure to follow them carefully.

Servo Tools and Parts

All of the tools needed are common and can be found in most households and school shops. They can also be purchased at local hardware stores. The Parallax screwdriver is included in the Robotics Shield Kit, and the other two are optional but handy to have.

Tools

(1) screwdriver, Phillips #1
(1) 1/4″ combination wrench (optional but handy)
(1) needle-nose pliers (optional)

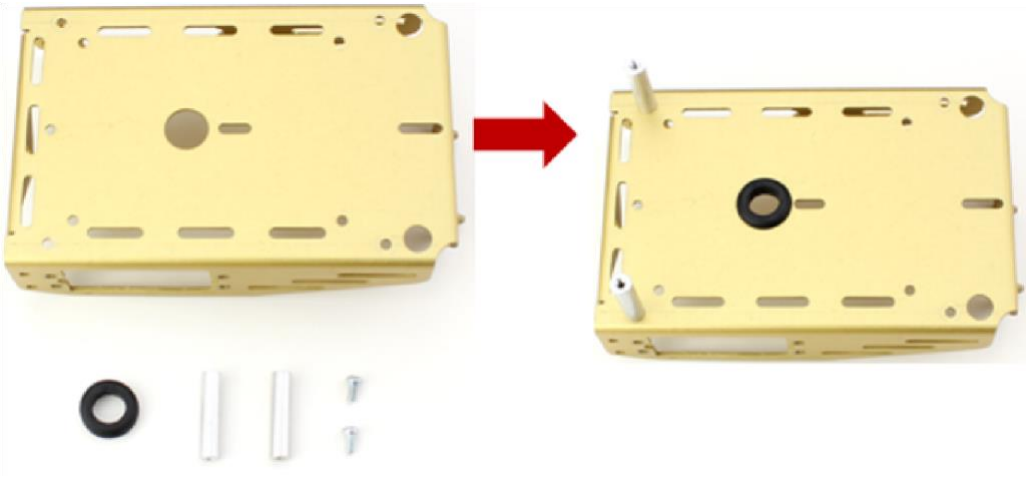
- ✓ Disconnect the programming cable and battery pack from your Arduino.
- ✓ Disconnect the servos from the BOE Shield.
- ✓ Set the BOE Shield's 3-position power switch to position 0.

Mount the Topside Hardware

Parts List

(1) robot chassis
(2) 1″ standoffs (removed from BOE Shield)

(2) pan-head screws, 1/4″ 4-40 (removed from BOE Shield)
(1) rubber grommet, 13/32″



Instructions

- Remove the 1″ aluminum standoffs from the BOE Shield, and save the standoffs and screws.
- Insert the 13/32″ rubber grommet into the hole in the center of the chassis.
- Make sure the groove in the outer edge of the rubber grommet is seated on the metal edge of the hole.
- Use two 1/4″ of the 4-40 screws to attach two of the standoffs to the top front of the chassis as shown.
- Save the other two standoffs and screws for a later step.

Remove the Servo Horns

Parts List

(2) Parallax continuous rotation servos, previously centered

Instructions

- Use a Phillips screwdriver to remove the screws that hold the servo control horns on the output shafts.
- Pull each horn upwards and off the servo output shaft.
- Save the screws; you will need them again soon.

5

Mount the Servos on the Chassis

Mount the Servos on the Chassis

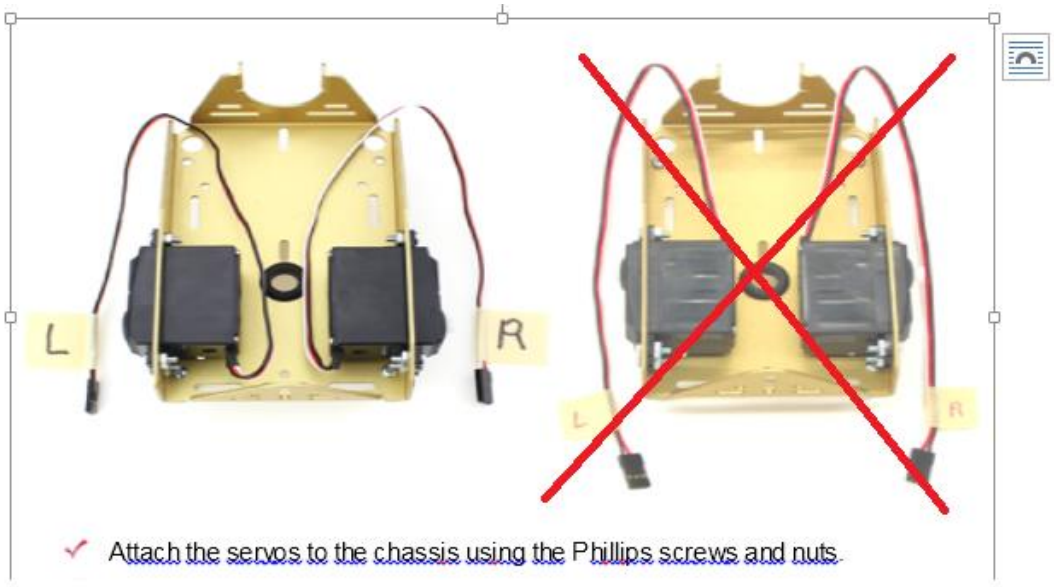
Parts List

(2) BOE Shield-Bot Chassis, partially assembled.
(2) Parallax continuous rotation servos
(8) pan Head Screws, 3/8″ 4-40
(8) nuts, 4-40 masking tape pen

Instructions:

1. Outside-forward (left) — the servos' mounting tabs seat outside the chassis, with their potentiometer access ports facing toward the front of the chassis. This allows easy access to adjust the potentiometers on an assembled robot, and also makes servo replacement quick. However, this gives the BOE Shield-Bot a longer, wider wheel base, so it will be a little less nimble on maneuvers and may need more pulses to make turns.

5



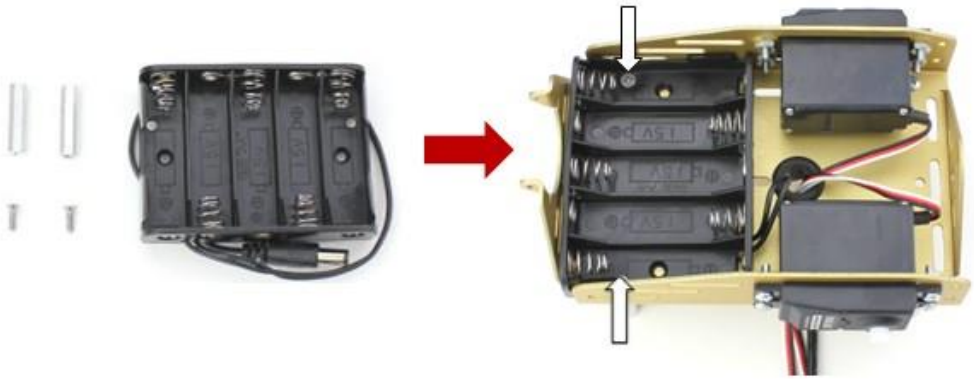
Mount the Battery Pack

Your kit may include a 4-cell battery pack or a 5-cell battery pack. Instructions for both options are included here. Don't forget the last step, for both versions, at the bottom of the page.

5-cell Battery Pack

Parts List

(2) flat-head Phillips screws, 3/8″ 4-40
(2) 1" standoffs (removed from BOE Shield previously) (1) 5-cell battery pack with 2.1 mm center-positive plug

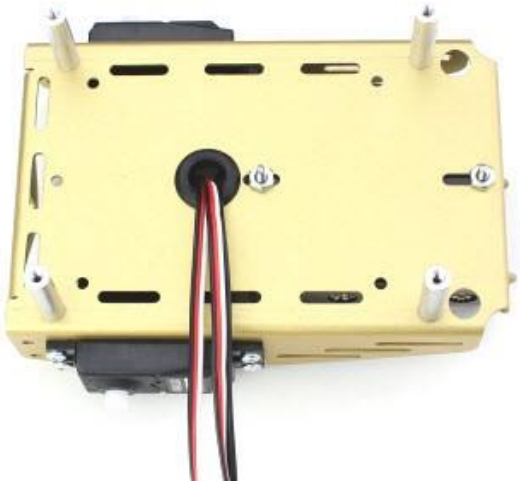


Instructions

- Place the empty battery pack inside the chassis positioned as shown above.
- Insert the two flat-head screws through the inside of the battery pack. Use the smaller set of holes that line up with the chassis mounting holes for the front standoffs, shown by the arrows.

6

- From the top of the chassis, thread a 1" standoff on each screw and tighten.
- as shown below.



Pull the battery pack’s power cord and servo lines through the rubber grommet hole in the center of the chassis,

6

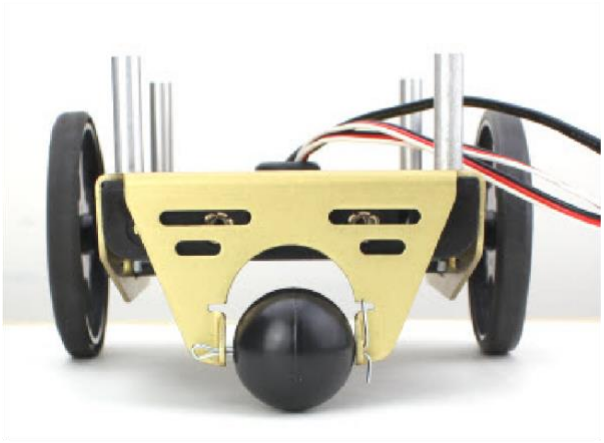
Mount the Wheels

Parts List

(1) 1/16″ cotter pin
(1) tail wheel ball
(2) rubber band tires
(2) plastic machined wheels
(2) screws saved when removing the servo horns

Instructions

The robot's tail wheel is merely a plastic ball with a hole through the center. A cotter pin holds it to the chassis and
functions as an axle for the wheel.



- Line up the hole in the tail wheel with the holes in the tail portion of the chassis.
- Run the cotter pin through all three holes (chassis left, tail wheel, chassis right).
- Bend the ends of the cotter pin apart so that it can't slide back out of the hole.
- Press each plastic wheel onto a servo output shaft, making sure the shaft lines up with, and sinks into, the wheel' recess, then secure with the saved servo screws.
- Stretch each rubber band tire and seat it on the outer edge of each wheel.