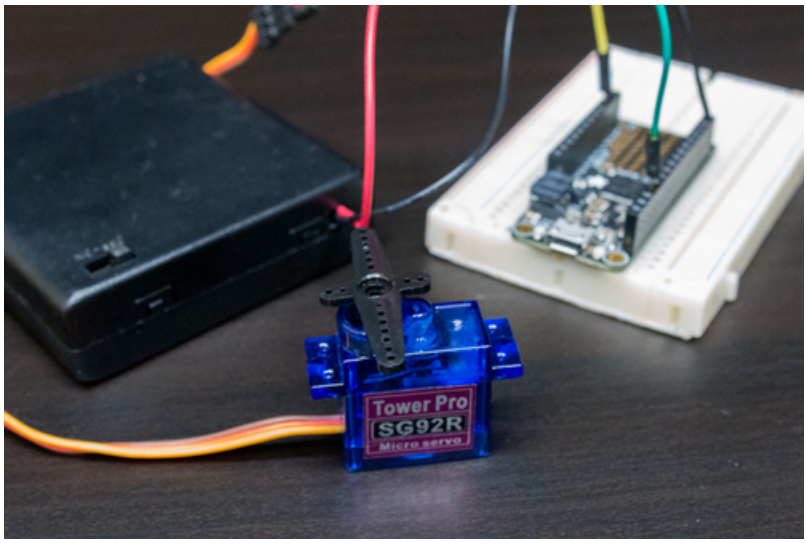
adafruit learning system

Using Servos With CircuitPython and Arduino

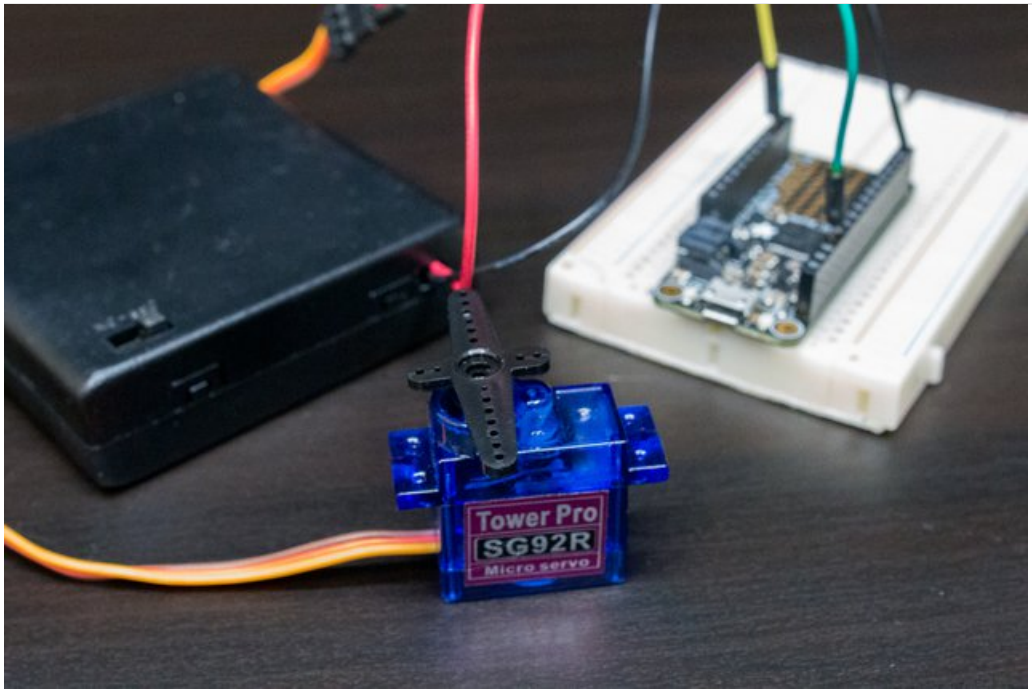
Created by Tony DiCola



Last updated on 2018-01-16 04:36:42 AM UTC

Guide Contents

Overview



Servos are tiny motors that you can control the position of by generating a special signal. You might use a servo to move something back and forth, like moving a dial to indicate a measurement or even moving a latch to open and close a door. There are even special 'continuous rotation' servos that can act like little motors with control over their speed and direction--perfect for building a simple robot! The possibilities for movement with servos are unlimited!

This guide will explore how to control servo motors from CircuitPython code. You can use simple Python code to move a servo to different positions. You can even use the CircuitPython REPL to move a servo interactively!

In addition this guide will also show basic servo control with Arduino code too!

Before you get started it will help to familiarize yourself with servos by reading these guides:

- Adafruit Motor Selection Guide - Servo Motors
- Arduino Lesson 14: Servo Motors

Hardware

To follow this guide you'll need the following parts:

- **A servo motor.** Either a standard servo or 'continuous rotation' servo will work for this guide, but be sure it's a servo motor and not a DC motor (which requires a different method of driving it). Check out the Adafruit Motor Selection Guide for more details on the different types of motors. If in doubt grab a small micro servo to experiment with and follow this guide.
- **A ~5 volt power supply for the servo.** Servos and motors in general can pull a surprisingly large amount of power, particularly when they're stopped or meeting resistance. In some cases they can pull so much power that it damages or overwhelms your development board! To prevent issues like this it's highly recommended to use a separate power supply for the motors in your project. A 4x AA battery pack is a perfect option for powering a few servos.
- **A board running CircuitPython.** If you're controlling servos from CircuitPython you'll need a board like the Feather M0 basic which can be loaded with CircuitPython firmware. This guide will also show basic servo code for Arduino too.
- **Breadboard and jumper wires.** You'll need these parts to connect components to your development board.

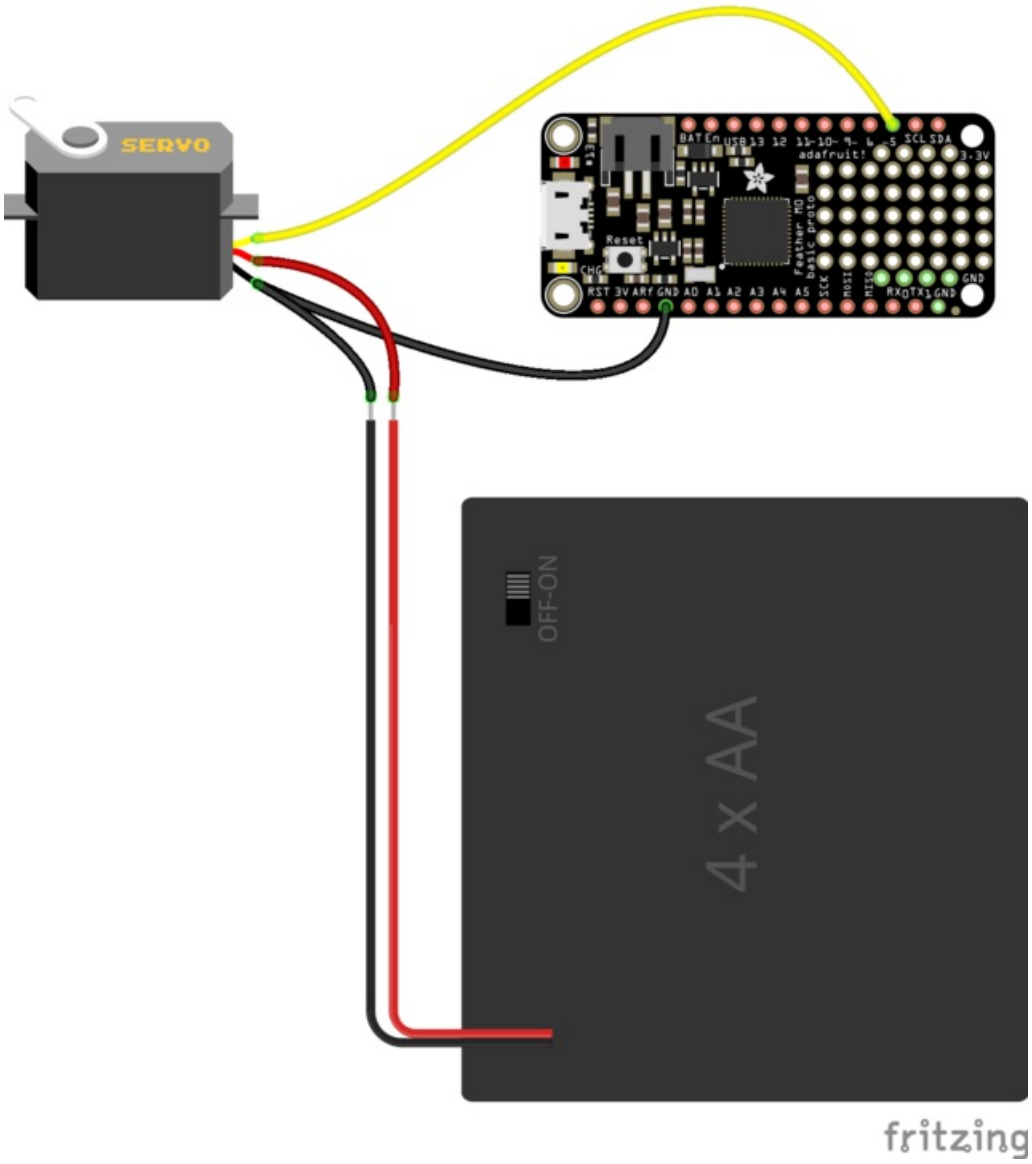
Wiring

Servo motors have three wires to connect to your board:

- **Power** - Typically a red wire which must be driven by 3-6 volts. However always check your servo datasheet or product information to make sure it's safe to power with the voltage you intend to provide. **If the voltage is too high it might damage the servo!**
- **Ground** - Typically a brown or black wire which must be connect to **both** your board and servo power supply grounds.
- **Signal** - Typically an orange or yellow wire. This wire is connected to a PWM or pulse-width modulation output on your development board. For boards like the Feather M0 look for pins with a squiggly line next to them to see those that support PWM output. For other boards check the product information or guide to see which outputs support PWM signals.

You might see a fourth wire coming from your servo motor too. Some advanced servos provide an output to help determine the position of the motor. You can ignore this fourth wire as it won't be used in this guide. However be absolutely sure you're using a servo motor and not a stepper or other type of motor if you see more or less than three wires coming from your device!

Here's an example of wiring a servo to a Feather M0 and a 4x AA power supply:



- **Servo power (red wire)** to **power supply positive voltage (red wire)**.
- **Servo ground (brown/black wire)** to both **power supply ground (black wire)** and **board GND**.
- **Servo signal (orange/yellow wire)** to **board D5** (or any other output that supports PWM signals).

CircuitPython

To control the servo from CircuitPython we'll use its built in PWM, or pulse-width modulation, signal generation capabilities. Be sure to read the CircuitPython analog I/O guide for more details on PWM signals!

Before you get started it will help to read these guide pages for more background information on servo motors too:

- Adafruit Motor Selection Guide - Servo Motors
- Arduino Lesson 14: Servo Motors

Be sure your hardware is wired up as shown on the previous page, and your servo power supply is turned on (you might notice the servos jerk or move slightly when the power is turned on--that's normal!).

Next make sure you are running the latest version of Adafruit CircuitPython for your board, then connect to the board's serial REPL so you are at the CircuitPython >>> prompt.

Pick Your Poison

There's *two* ways to control servos, one is lower-level - where you will control the PWM pin directly. And one is higher-level using the Motor Library

We'll go through the lower level one first

Low Level Servo Setup

First you need to import a few modules to access the PWM output capabilities of CircuitPython:

```
import board
import pulseio
```

Now you can create a PWM signal output that will drive the position of the servo:

```
servo = pulseio.PWMOut(board.D5, frequency=50)
```

There are a couple important things happening with the line above. This is an initializer which is creating an instance of the PWMOut class and part of that initialization process is specifying these two values:

- **The pin that will be the PWM output.** In this case it's pin D5 on the development board. If you're using a different output be sure to specify the right pin name here (and make sure the pin supports PWM output as mentioned on the previous page!).
- **The frequency of the PWM signal.** This is an optional value that we're specifying as a keyword argument to tell the PWM output that we want a signal with a frequency of 50 hertz. The frequency is how many times per second the servo signal changes and for most servos they expect a ~50 hz signal.

Position Control with PWM

Now we can control the position of the servo by changing the duty cycle of the PWM signal. If you aren't familiar with PWM signals and duty cycle be sure you've read the CircuitPython analog I/O guide PWM page for more background.

With a servo motor it will change its position depending on the pulse length of the signal being sent to it. Specifically a 1 millisecond high pulse tells the servo to move all the way to one extreme (left or right) and a value of 2 milliseconds

will move all the way to the opposite extreme. Any value in-between 1 to 2 milliseconds will move the servo to an appropriate in-between position. For example a value of 1.5 milliseconds will move the servo to its center or middle position.

Note for continuous rotation servos they don't have a concept of moving to a specific position. Instead they will rotate freely in a circle and change their speed depending on the pulse length sent to them. A 1 millisecond pulse moves as quickly as possible in one direction, 1.5 millisecond pulse stops movement, and 2 millisecond pulse moves as quickly as possible in the opposite direction.

How do we control the amount of time a PWM signal is high vs. low? As mentioned in the CircuitPython analog I/O PWM page we do so by changing the duty cycle of the PWM signal. However before you set the duty cycle you'll need to convert from a desired millisecond value to a PWM duty cycle value. Remember with CircuitPython duty cycle values are specified with a 16-bit unsigned value, i.e. something from 0 to 65535. We can actually create a little Python function to help with doing this mathematical conversion:

```
def servo_duty_cycle(pulse_ms, frequency=50):
period_ms = 1.0 / frequency * 1000.0
duty_cycle = int(pulse_ms / (period_ms / 65535.0))
return duty_cycle
```

You can pass this function a pulse width in milliseconds and it will convert it into the appropriate duty cycle value to control the servo. The function assumes a frequency of 50 hz by default too, but you can change it by specifying the frequency keyword too (however you don't typically need or want to change this unless you also changed the frequency of the PWM output!).

For example see what output value you get with pulse widths of 1.0 and 2.0 milliseconds:

```
servo_duty_cycle(1.0)
servo_duty_cycle(2.0)
```

```
>>> def servo_duty_cycle(pulse_ms, frequency=50):
...     period_ms = 1.0 / frequency * 1000.0
...     duty_cycle = int(pulse_ms / (period_ms / 65535.0))
...     return duty_cycle
...
>>> servo_duty_cycle(1.0)
3276
>>> servo_duty_cycle(2.0)
6553
```

Now let's move the servo! Try changing the servo PWM output duty cycle to a value that corresponds to a 1.0 millisecond long pulse. You should see the servo move to one extreme:

```
servo.duty_cycle = servo_duty_cycle(1.0)
```

If you don't see your servo move be sure the power supply is turned on and connected to the servo. Double check all of your wiring connections and that the output for the signal supports PWM signals too!

Now move to the opposite extreme with a 2.0 millisecond long pulse:

```
servo.duty_cycle = servo_duty_cycle(2.0)
```

You should see the servo move about 180 degrees to the opposite position.

Finally move to the center position with a 1.5 millisecond pulse:

```
servo.duty_cycle = servo_duty_cycle(1.5)
```

Experiment with setting any value in-between 1.0 and 2.0 to see how the servo reacts.

Sweep Example

Here's a complete example that will sweep the servo between both extremes (1.0 and 2.0 millisecond long pulses) repeatedly. Save this as a **main.py** file on your board (and be ready when you click save as the servo might instantly start moving!):

```
import time

import board
import pulseio


# Initialize PWM output for the servo (on pin D5):
servo = pulseio.PWMOut(board.D5, frequency=50)

# Create a function to simplify setting PWM duty cycle for the servo:
def servo_duty_cycle(pulse_ms, frequency=50):
    period_ms = 1.0 / frequency * 1000.0
    duty_cycle = int(pulse_ms / (period_ms / 65535.0))
    return duty_cycle

# Main loop will run forever moving between 1.0 and 2.0 mS long pulses:
while True:
    servo.duty_cycle = servo_duty_cycle(1.0)
    time.sleep(1.0)
    servo.duty_cycle = servo_duty_cycle(2.0)
    time.sleep(1.0)
```

That's all there is to controlling a servo directly with CircuitPython with a PWM output

Position Control with Motor Library

Another way to control servos is with a handy Adafruit CircuitPython Motor module which simplifies setting the duty cycle to control servos (and even allows controlling servos from different PWM hardware like the PCA9685 board).

To follow this approach you'll need to install the Adafruit CircuitPython Motor module on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle. Our introduction guide has a great page on how to install the library bundle for both express and non-express boards.

Remember for non-express boards like the, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_motor**

You can also download the **adafruit_motor** folder from its releases page on Github.

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_motor** folder copied over.

Now connect to the board's serial REPL so you are at the CircuitPython >>> prompt. Then import the **board** and **pulseio** modules as before. We'll create a PWM output just like as shown in the first section--the motor library will take this PWM output and add an extra level of simple control on top of it:

```
import board
import pulseio
pwm = pulseio.PWMOut(board.D5, frequency=50)
```

Now let's import the **servo** submodule from the **adafruit_motor** module and create an instance of the **Servo** class from it:

```
import adafruit_motor.servo
servo = adafruit_motor.servo.Servo(pwm)
```

Notice the servo class needs to be told what PWM output the servo is connected to for your board. If you're following the wiring and this guide it will be a PWM output on board pin 5 (be sure to create this with a 50 hz frequency as shown!).

There are a few optional keyword parameters that you might specify in the initializer too. These aren't shown and are useful for using very specialized or custom servos with different ranges or pulse width values--for simple servos you don't typically need to set these values:

- **actuation_range** - The range in degrees of the servo movement. The default is 180 degrees.
- **min_pulse** - The minimum position pulse length in microseconds (default 1000 us).
- **max_pulse** - The maximum position pulse length in microseconds (default 2000 us).

Controlling the servo is simple once you have the class created, just set the **angle** property to a value from 0 to 180 degrees!

```
servo.angle = 0
servo.angle = 90
servo.angle = 180
```

Notice each time you set angle the servo springs to life and moves!

If you don't see the servo moving be sure you have it wired and powered exactly as shown. Also make sure you've created the PWM output exactly as shown above too (note the 50 hz frequency!).

There's another type of servo class in the module that you might find useful too, the **ContinuousServo** class. This is for controlling continuous rotation servos that move completely around like a small motor instead of point at specific angles. For these servos you control their speed (or throttle). Create an instance just like with the **Servo** class above:

```
continuous = adafruit_motor.servo.ContinuousServo(pwm)
```

With these servos you set the **throttle** property to a value from -1 to 1 (or anything in between, including fractional values). Where -1 is fully speed 'backwards' and 1 is full speed 'forwards'. A value of 0 should stop the motor (but your servo might need to be trimmed a bit and have a small value set that stops its movement instead of zero, experiment with values yourself to see).

```
continuous.throttle = -1   # Full backwards
continuous.throttle = 0    # Stop
continuous.throttle = 1    # Full forwards
continuous.throttle = 0.5  # Half speed forwards
```

Here's the same complete servo sweep example but written for the motor module. Remember save this as **main.py** on your board to have it run:

```
import time

import board
import pulseio

import adafruit_motor.servo


# Initialize PWM output for the servo (on pin D5) and servo:
pwm = pulseio.PWMOut(board.D5, frequency=50)
servo = adafruit_motor.servo.Servo(pwm)

# Main loop will run forever moving between 0 and 180 degree extremes.
while True:
    servo.angle = 0
    time.sleep(1.0)
    servo.angle = 180
    time.sleep(1.0)
```

Check out all of the examples in the Adafruit CircuitPython Motor module too!

Arduino

You can also control a servo motor from Arduino in a similar way as CircuitPython with Arduino's Servo library. There are actually quite a few resources and guides for using Arduino to control servos, so this page will just highlight how to use a servo in Arduino in the same way as with CircuitPython from the previous page. For more exploration of servos and Arduino be sure to check out the entire Arduino Lesson 14: Servo Motors guide.

First make sure your servo is wired to your board exactly as shown on the hardware page of this guide. You'll also need the Arduino IDE installed and configured to upload to your board. Remember Arduino sketches have to be written entirely up front and uploaded to the board--you can't interactively control servos like you can with CircuitPython.

To control the duty cycle of the servo signal with Arduino we'll use the Servo library instead of directly controlling the PWM output signal. This is necessary because Arduino doesn't support as much control over PWM output as CircuitPython. In particular you can't easily change the frequency of a PWM output to the 50hz value required by servos. Luckily you can use a library to do this frequency control and PWM signal generation for you internally.

Upload the following sketch to your board and it will move the servo between its two extreme positions (a 1.0 and 2.0 millisecond pulse length):

```
#include <Servo.h>

// Create a servo instance.
Servo servo;

void setup() {
  // Attach servo output to pin 5.
  servo.attach(5);
}

void loop() {
  // Move to position 0, or a 1.0 millisecond long pulse:
  // Remember the servo module in Arduino takes in a position in degrees
  // from 0 to 180 instead of a pulse length in milliseconds or other value.
  servo.write(0);
  // Delay for a second.
  delay(1000);
  // Move to position 180, or a 2.0 millisecond long pulse and pause again.
  servo.write(180);
  delay(1000);
}
```

Notice the servo library controls the servo using a slightly different method of specifying the position in degrees instead of the raw PWM pulse length in milliseconds. Internally the servo library is just converting that position value to a PWM pulse length like you saw with the CircuitPython page of this guide. You can specify a position of 0 for one extreme (1.0 millisecond long pulse), 90 for the center or middle position (1.5 millisecond long pulse), and 180 for the opposite extreme (2.0 millisecond long pulse).

That's all there is to controlling a servo with Arduino!