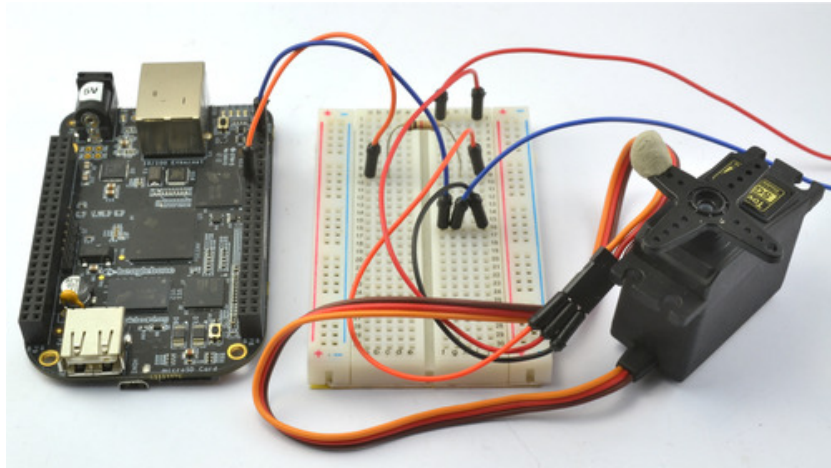


## Controlling a Servo with a BeagleBone Black

Created by Simon Monk



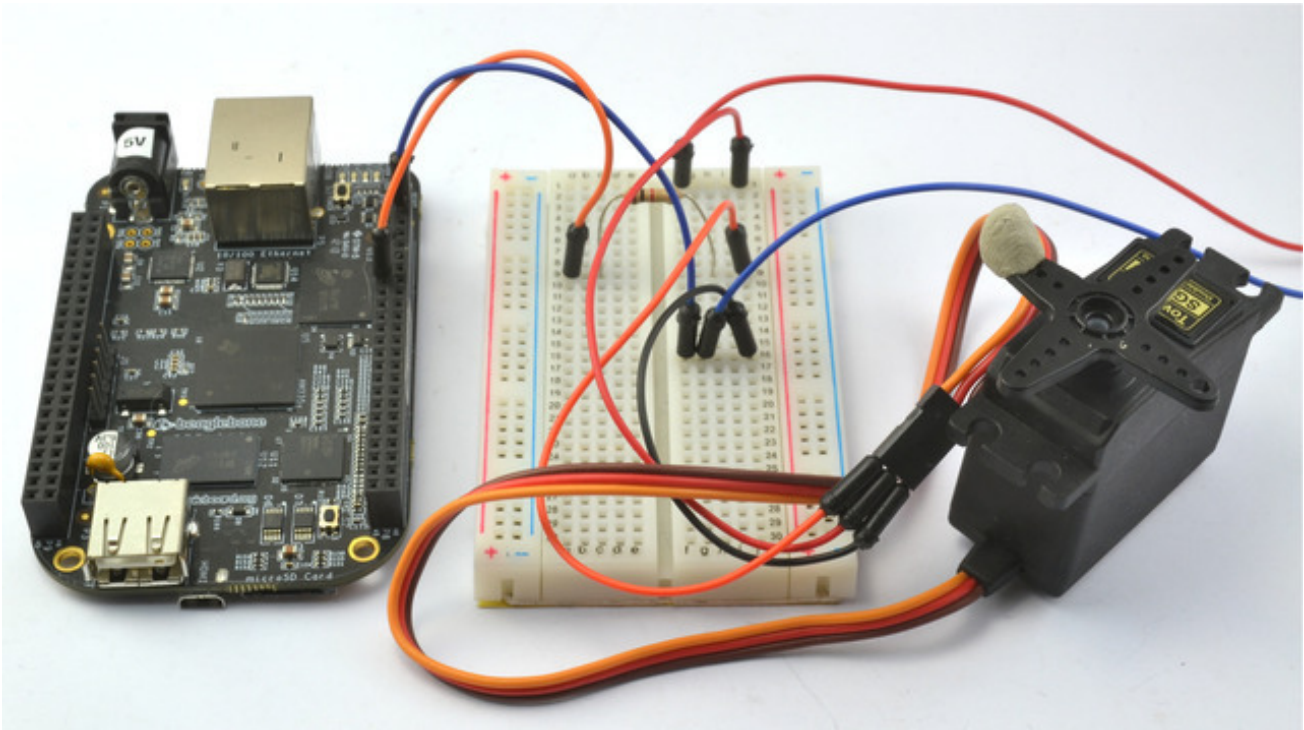
Last updated on 2014-08-14 02:45:10 PM EDT

## Guide Contents

Guide Contents	2
Overview	3
You Will Need	4
Installing the Python Library	7
Wiring	8
The Python Console	9
Writing a Program	10
Servo Motors	12
Next Steps	13

# Overview

In this tutorial, you will learn how to control a servo from Python using a BeagleBone Black (BBB).

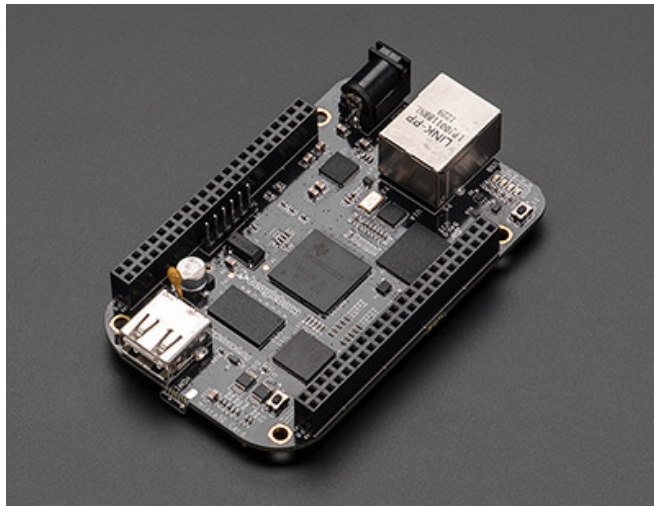


A Python test program will allow you to set the angle of the servo between 0 and 180 degrees.

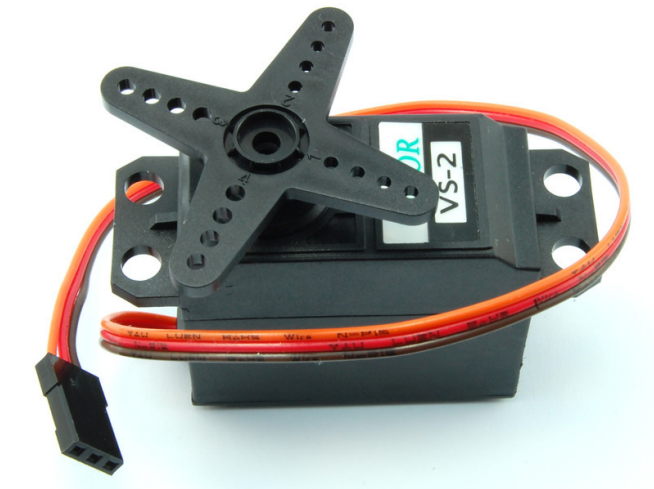
```
# python servo.py
Angle (0 to 180 x to exit):90
Angle (0 to 180 x to exit):180
Angle (0 to 180 x to exit):0
Angle (0 to 180 x to exit):x
#
```

# You Will Need

To try out this tutorial, you will need:



BeagleBone Black



Standard Servo



or a Micro Servo



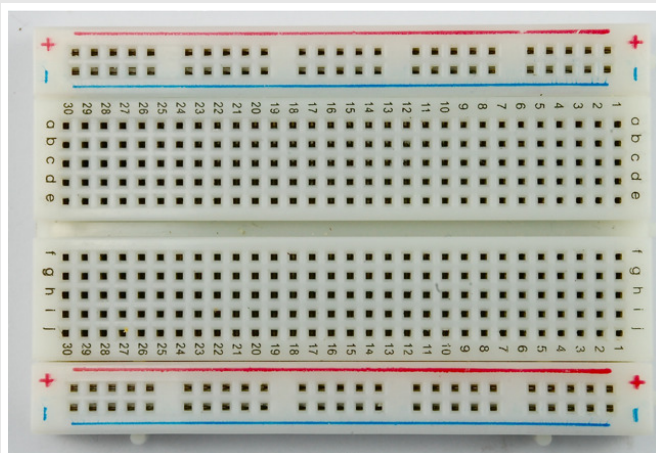
1 kΩ Resistor (optional)



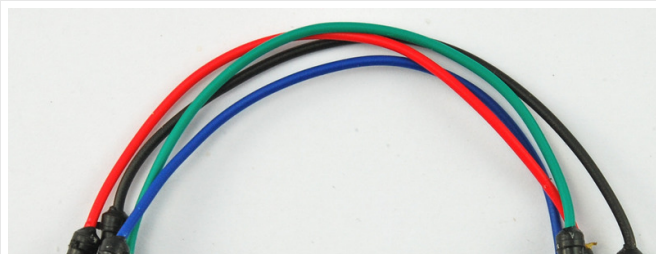
5V Power Supply



Female Screw Terminal adaptor



Half-sized Breadboard



Male to Male Jumpers



The 1 k $\Omega$  resistor is not strictly necessary, but will protect your BBB from damage if something should go wrong in the servo.

# Installing the Python Library

This tutorial uses Ångström Linux, the operating system that comes pre-installed on the BBB.

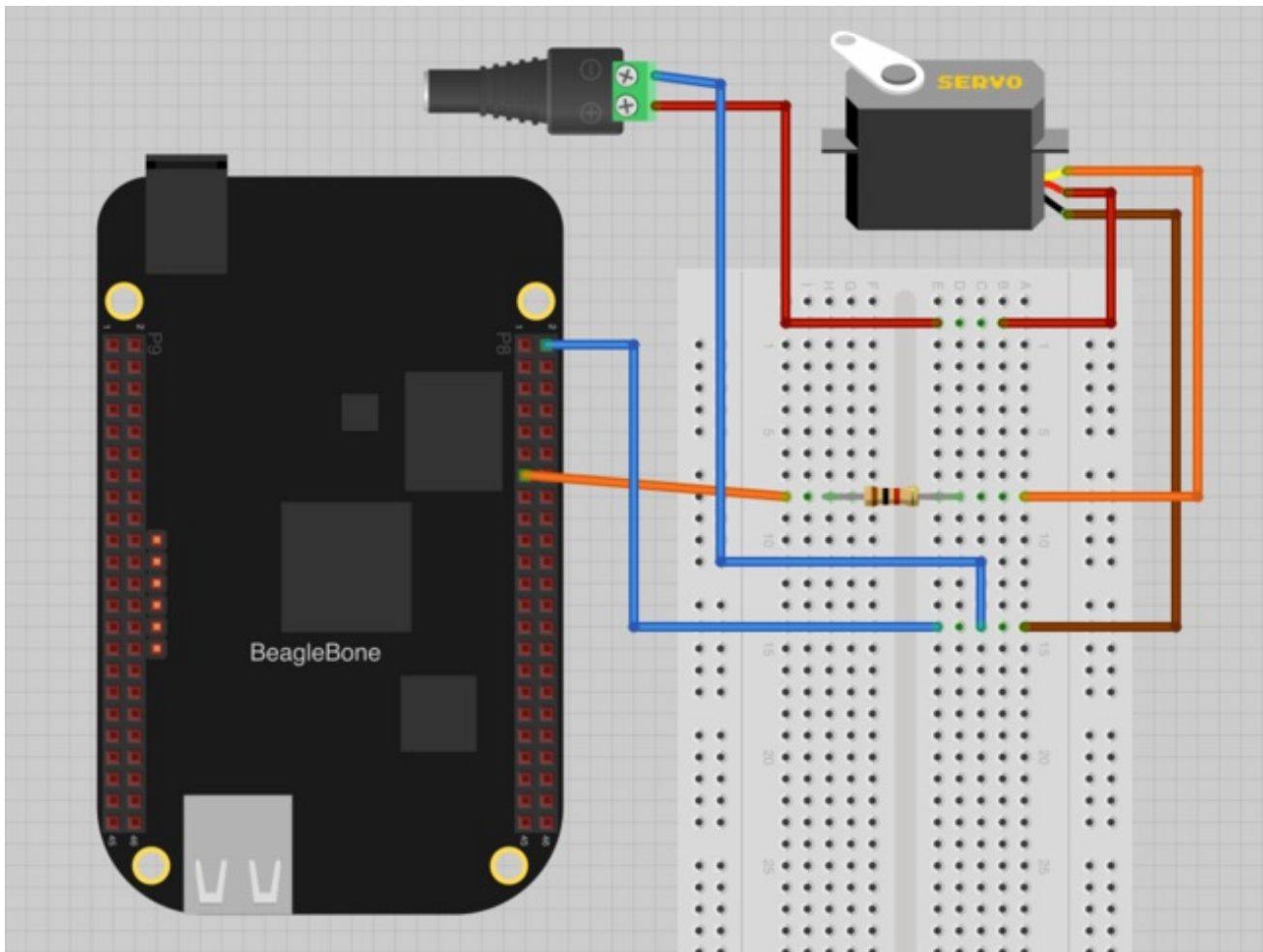
Follow the instructions here, to install the Python IO BBIO library.

<http://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black> (<http://adafru.it/cgh>)



# Wiring

Wire up the solderless breadboard using the header leads as shown below.



You can use male to male jumper wires from the screw terminal to the breadboard. Connect an external 5VDC power supply to DC power jack

We will use pin P8\_13 as the PWM output to control the servo. The only other connection that we need from the BBB is GND.

There is more information about all the pins available on the P8 and P9 connectors down each side of the BBB here: <http://stuffwemade.net/hwio/beaglebone-pin-reference/> (<http://adafru.it/cgi>)



# The Python Console

Before writing a Python program to allow us to set the servo to an angle between 0 and 180, we can try some experiments in the Python Console.

To launch the Python Console type:

```
# python
Python 2.7.3 (default, Apr 3 2013, 21:37:23)
[GCC 4.7.3 20130205 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

First, we need to import the library, so enter the command:

```
>>> import Adafruit_BBIO.PWM as PWM
```

Now enter the commands below into the Python Console one at a time, and you should see the servo change position.

```
>>> PWM.start("P8_13", 95.0, 60)
>>> PWM.set_duty_cycle("P8_13", 97.0)
>>> PWM.stop("P8_13")
>>> PWM.cleanup()
```

**NOTE: If your servo doesn't move you might need to change the polarity of the PWM signal and try again.** Add a 4th parameter to the PWM.start function with a value of 1 (one), this will invert the PWM signal so it's low when normally high and vice versa.

Here's the same code but with the inverse parameter set:

```
>>> PWM.start("P8_13", 95.0, 60, 1)
>>> PWM.set_duty_cycle("P8_13", 97.0)
>>> PWM.stop("P8_13")
>>> PWM.cleanup()
```

# Writing a Program

Exit the Python Console by typing:

```
>>> exit()
```

This should take you back to the Linux prompt.

Enter the following command to create a new file called servo.py

```
nano servo.py
```

Now paste the code below into the editor window.

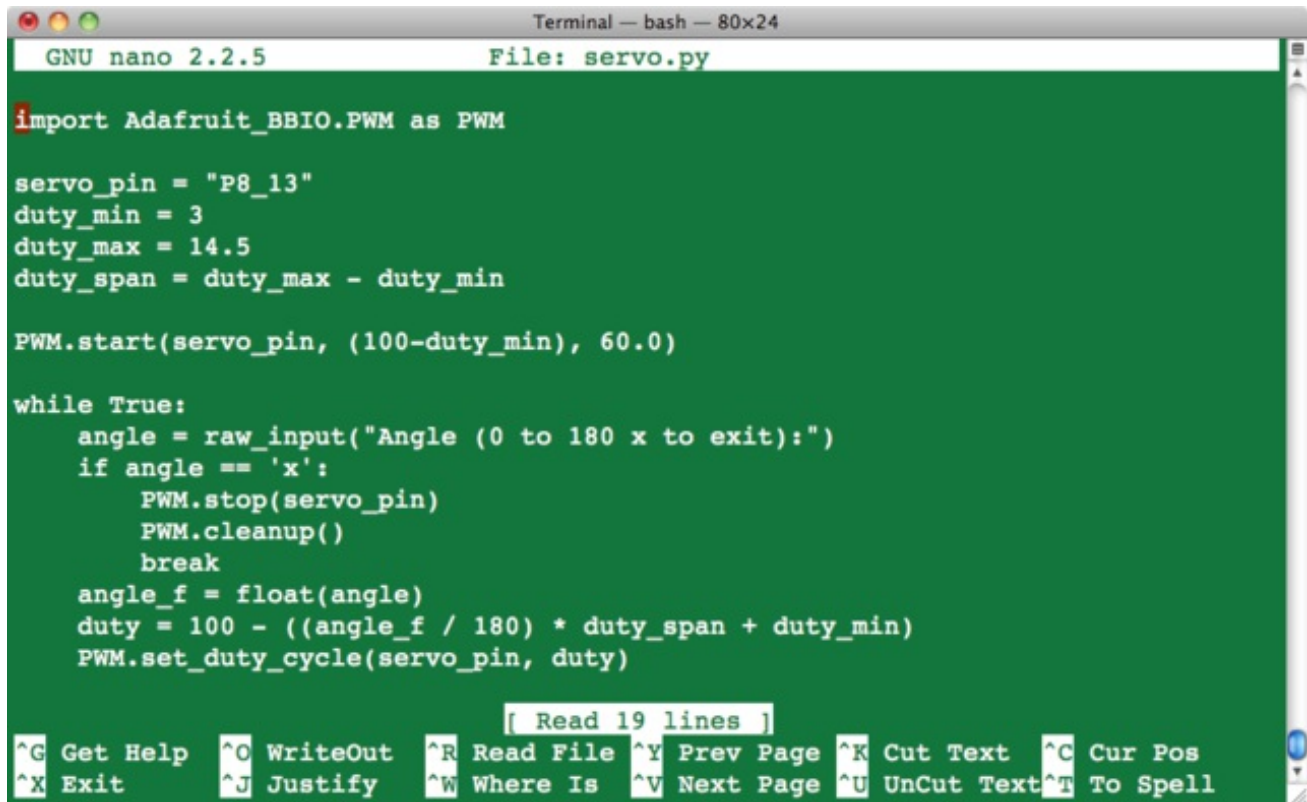
**NOTE: Don't forget to add the inverse parameter to the PWM.start function if you found it was required to make your servos move in the previous page!**

```
import Adafruit_BBIO.PWM as PWM

servo_pin = "P8_13"
duty_min = 3
duty_max = 14.5
duty_span = duty_max - duty_min

PWM.start(servo_pin, (100-duty_min), 60.0)

while True:
    angle = raw_input("Angle (0 to 180 x to exit):")
    if angle == 'x':
        PWM.stop(servo_pin)
        PWM.cleanup()
        break
    angle_f = float(angle)
    duty = 100 - ((angle_f / 180) * duty_span + duty_min)
    PWM.set_duty_cycle(servo_pin, duty)
```



```
GNU nano 2.2.5 File: servo.py

import Adafruit_BBIO.PWM as PWM

servo_pin = "P8_13"
duty_min = 3
duty_max = 14.5
duty_span = duty_max - duty_min

PWM.start(servo_pin, (100-duty_min), 60.0)

while True:
    angle = raw_input("Angle (0 to 180 x to exit):")
    if angle == 'x':
        PWM.stop(servo_pin)
        PWM.cleanup()
        break
    angle_f = float(angle)
    duty = 100 - ((angle_f / 180) * duty_span + duty_min)
    PWM.set_duty_cycle(servo_pin, duty)

[ Read 19 lines ]

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

To start the program, enter the command:

```
# python servo.py
Angle (0 to 180 x to exit):90
Angle (0 to 180 x to exit):180
Angle (0 to 180 x to exit):0
Angle (0 to 180 x to exit):x
#
```

Entering a value between 0 and 180 will set the servo's angle accordingly.

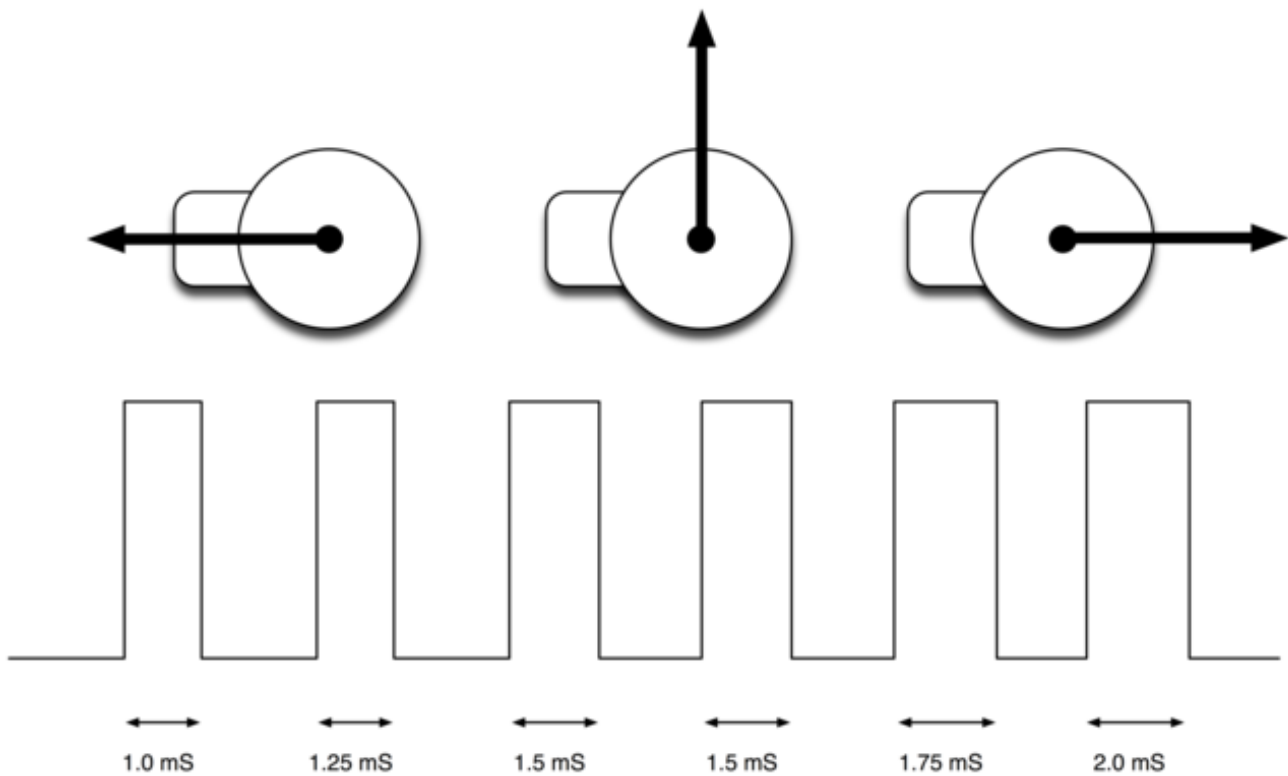
When you want to stop the program, enter 'x'.

You may find that your servo judders at one end of its range or does not give a full 180 degree range of movement. If this is the case, try tweaking the values in `duty_min` and `duty_max`.

When you enter 'x', the PWM is stopped and 'cleanup' is run, otherwise the PWM signal would continue in the background even after the program had stopped running.

# Servo Motors

The position of the servo motor is set by the length of a pulse. The servo expects to receive a pulse roughly every 20 milliseconds. If that pulse is high for 1 millisecond or less, then the servo angle will be zero, if it is 1.5 milliseconds, then it will be at its centre position and if it is 2 milliseconds or more it will be at 180 degrees.



"Continuous" Servos, also called "360 Servos" work very similarly, so you can use them just like a position servo. Instead of the absolute position, a continuous servo will adjust its speed with different pulse widths

This example uses the PWM feature of the GPIO library to generate the pulses for the servo. The PWM frequency is set to 60 Hz so that the servo will receive a pulse roughly every 17 milliseconds.

The length of the pulse is changed by adjusting the duty cycle over the fairly narrow range of 3 to 14.5 percent. These figures were estimated and then tweaked a bit to give a maximum range of the servo being used.

## Next Steps

---

If you wanted to, you could attach three more servos to the GPIO pins P8\_19, P9\_14 and P9\_16. They could all share the same external 5-6VDC power supply without any problem.

### **About the Author.**

As well as contributing lots of tutorials about Raspberry Pi, Arduino and now BeagleBone Black, Simon Monk writes books about open source hardware. You will find his books for sale [here \(http://adafru.it/caH\)](http://adafru.it/caH) at Adafruit.