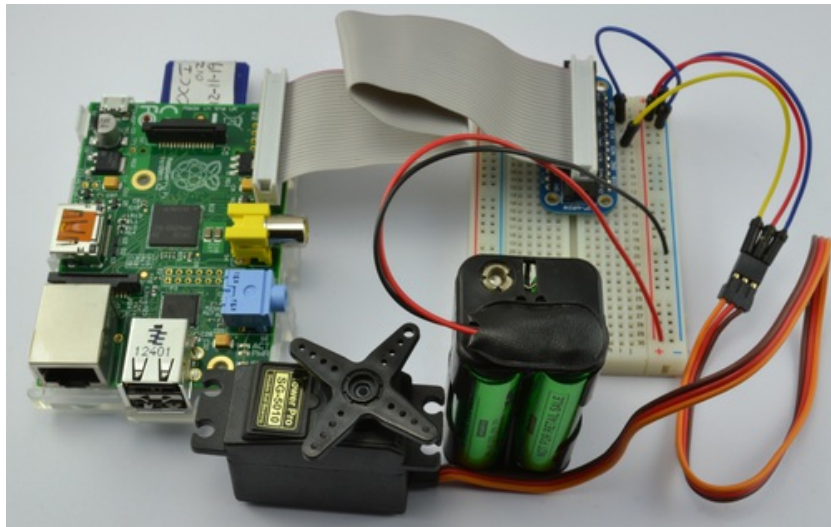




Adafruit's Raspberry Pi Lesson 8. Using a Servo Motor

Created by Simon Monk



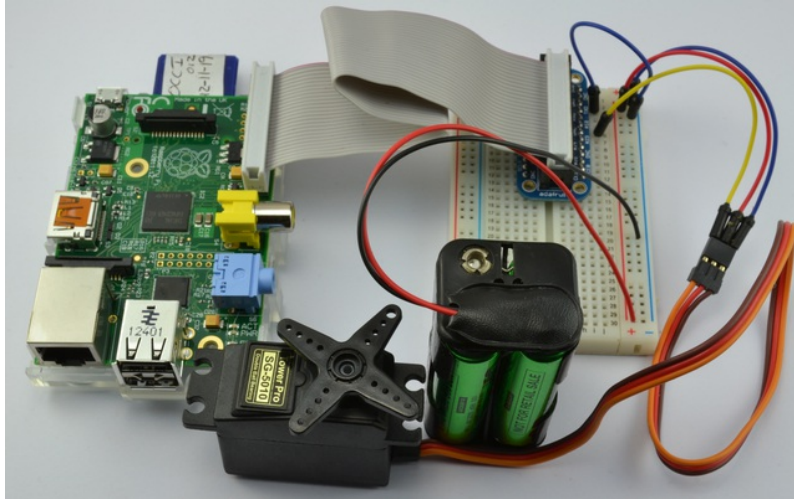
Last updated on 2013-09-11 02:00:59 PM EDT

Guide Contents

Guide Contents	2
Overview	3
Parts	4
Part	4
Qty	4
Servo Motors	6
The PWM and Servo Kernel Module	7
File	7
Description	7
Hardware	8
Software	10
Test & Configure	12

Overview

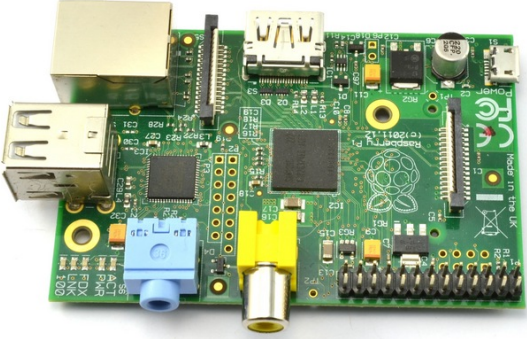

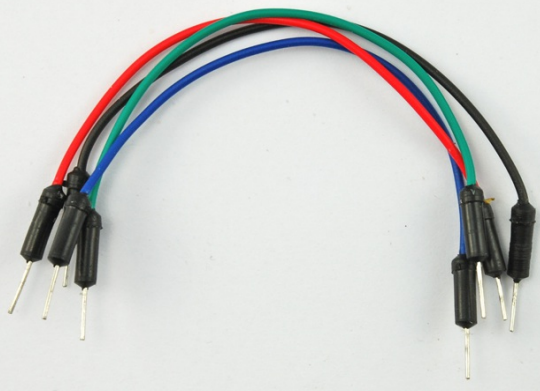
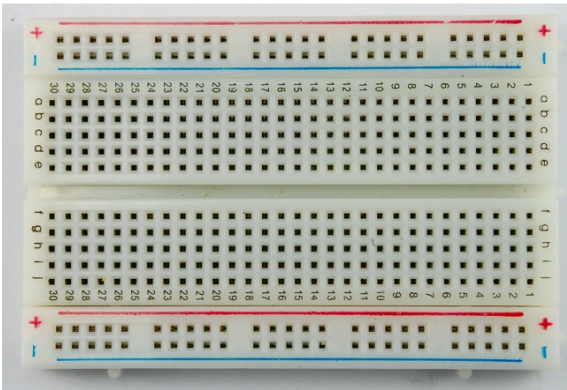
This lesson describes how to control a single servo motor using Python.

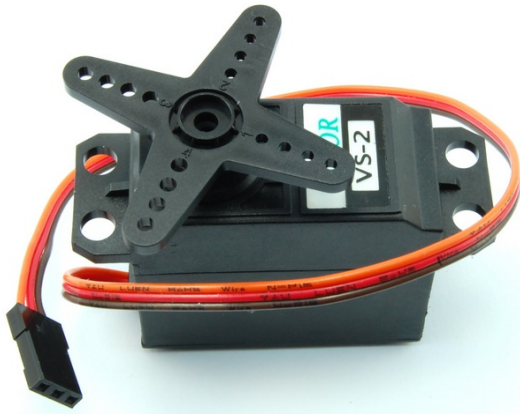


Servo motors are controlled by pulses of varying lengths. This requires fairly accurate timing. The Raspberry Pi has one pin that generates pulses in hardware, without having to rely on the operating system. Occidentalis includes an interface to make use of this pin for controlling a servo motor.

Parts

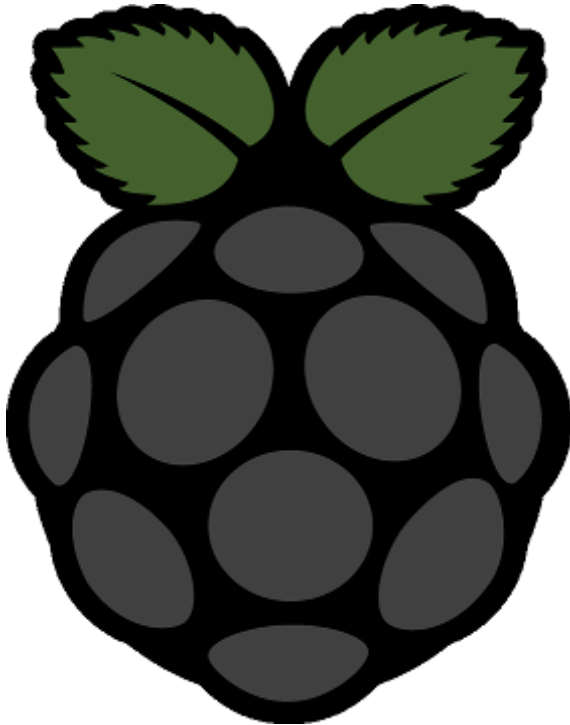
To build this project, you will need the following parts.

	Part	Qty
	Raspberry Pi	1
	Cobbler Breakout Board and 26-pin IDC cable	1
	Set of male to male jumper leads	1
	Half-size breadboard	1



Servo

1



Adafruit Occidentalis 0.2 or later operating system distribution.

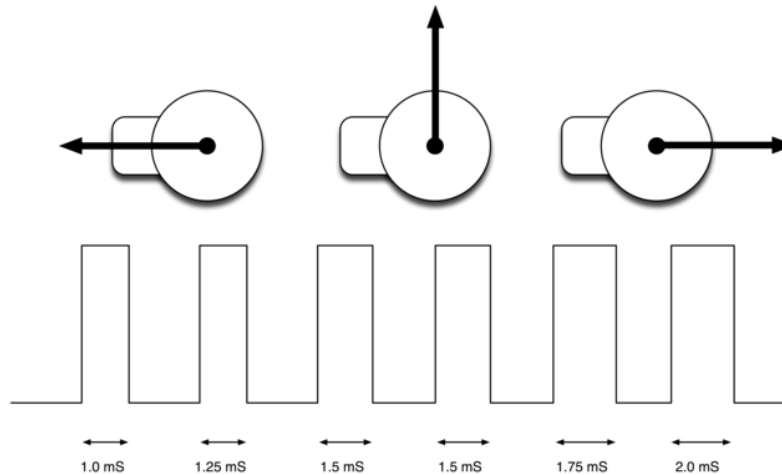


4 x AA or AAA battery holder and batteries.

1

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, 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 it will be at 180 degrees.



The end points of the servo can vary and many servos only turn through about 170 degrees. You can also buy 'continuous' servos that can rotate through the full 360 degrees.

The PWM and Servo Kernel Module

Adafruit and Sean Cross have created a kernel module that is included with the [Occidentalis](http://adafru.it/aQx) (<http://adafru.it/aQx>) distribution. For details of creating an Occidentalis follow [this link](http://adafru.it/aVr) (<http://adafru.it/aVr>). If you want to use the module with Raspbian or some other distribution, then there is help on installing the kernel module into your environment [here](http://adafru.it/aQx) (<http://adafru.it/aQx>).

The module is called **PWM** and Servo because as well as controlling servo motors, the module can also produce PWM (Pulse Width Modulation) signals that can be used (with extra electronics) to control the power to motors or lights. We will not be using the PWM feature in this lesson.

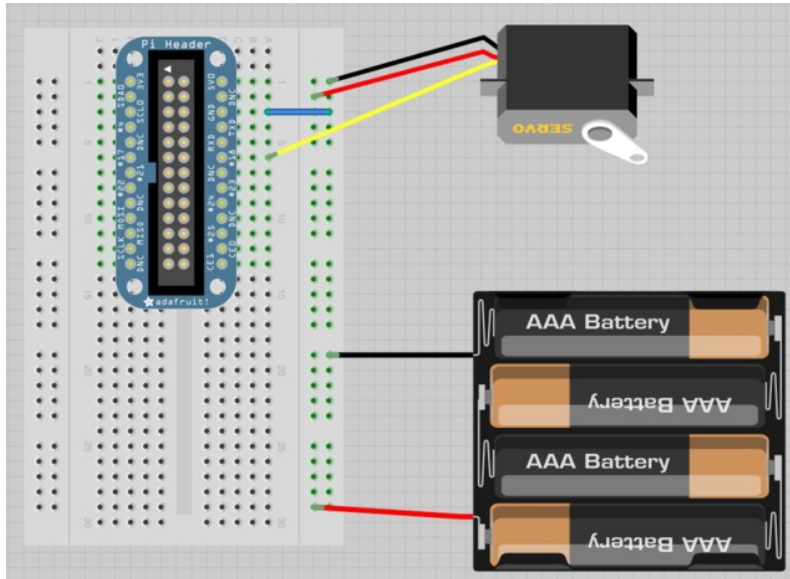
The PWM and Servo Module uses a file type of interface, where you control what the output pin and therefore the servo is doing, by reading and writing to special files. This makes it really easy to interface with in Python or other languages.

The files involved in using the module to drive a servo are listed below. All the files can be found in the directory `/sys/class/rpi-pwm/pwm0/` on your Raspberry Pi.

File	Description
active	This will be 1 for active 0 for inactive. You can read it to find out if the output pin is active or write it to make it active or inactive.
delayed	If this is set to 1, then any changes that you make to the other files will have no effect until you use the file above to make the output active.
mode	Write to this file to set the mode of the pin to either pwm, servo or audio. Obviously we want it to be servo. Note that the pin is also used by the Pi's audio output, so you cannot use sound at the same time as controlling a servo.
servo_max	Write to this file to set the maximum value for a servo position. We will set this to be 180 so that we can easily set the servo to any angle between 0 and 180.
servo	The value that you write to this file sets the servo pulse length in relation to servo_max. So if we set it to 90, with a servo_max of 180, it will set the servo to its center position.

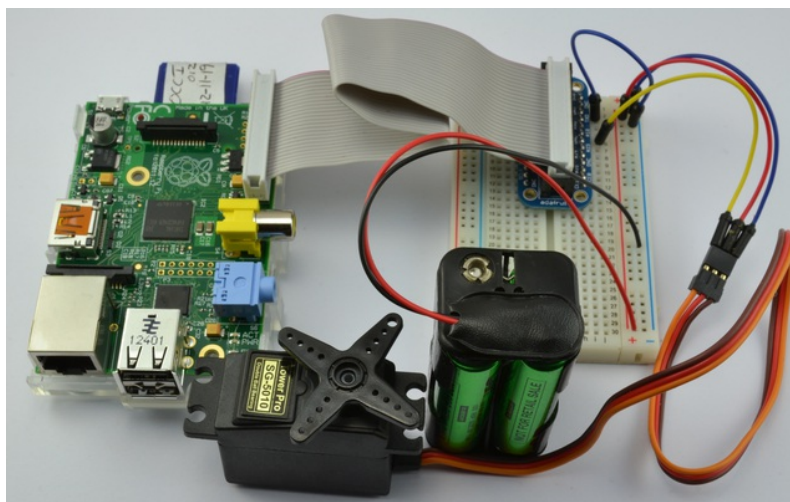
Hardware

There is only one pin on the Pi that is capable of producing pulses in this way (GPIO pin 18). This will be connected to the control pin of the servo. The power to the servo is provided by an external battery as powering the servo from the Pi itself is likely to cause it to crash as the Servo draws too much current as it starts to move. Servos require 4.8-6V DC power to the motor, but the signal level (pulse output) can be 3.3V, which is how its OK to just connect the signal line directly to the GPIO output of the Pi.



The Pi Cobbler is used to link the Raspberry Pi to the breadboard. If you have not used the Cobbler before take a look at [Lesson 4 \(http://adafru.it/aTH\)](http://adafru.it/aTH) in this series.

Servo motors generally come with three pin sockets attached. The red and brown sockets supply power (positive to red) and the third yellow or orange socket is for the control signal. To link the socket to the breadboard, use the male-to-male jumper wires.



Software

This project does not require any Python libraries to be installed.

The Python program to make the servo sweep back and forth is listed below:

```
# Servo Control
import time
def set(property, value):
    try:
        f = open("/sys/class/rpi-pwm/pwm0/" + property, 'w')
        f.write(value)
        f.close()
    except:
        print("Error writing to: " + property + " value: " + value)

def setServo(angle):
    set("servo", str(angle))

set("delayed", "0")
set("mode", "servo")
set("servo_max", "180")
set("active", "1")

delay_period = 0.01

while True:
    for angle in range(0, 180):
        setServo(angle)
        time.sleep(delay_period)
    for angle in range(0, 180):
        setServo(180 - angle)
        time.sleep(delay_period)
```

To make writing to the files easier, I have written a utility function called just 'set'. The first argument to this is the file to be written to (property) and the second argument the value to write to it.

So, the program starts with a few file writes to turn the delay mode off, set the mode to be 'servo', set the maximum servo value to be 180 and finally to make the output pin active.

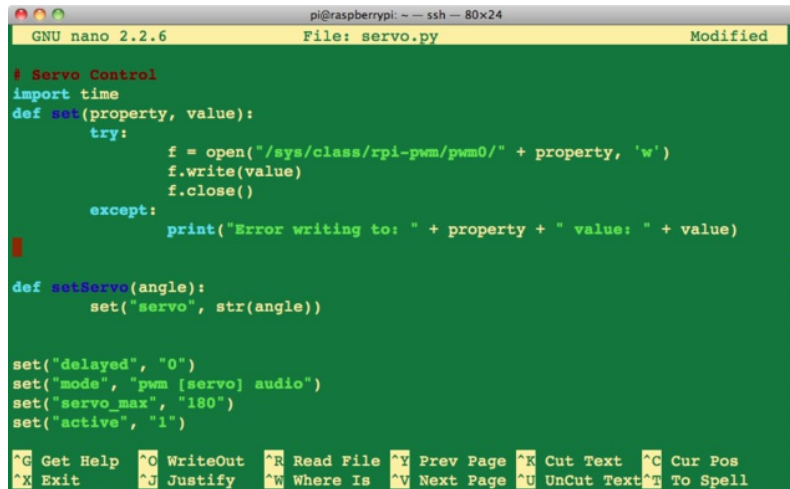
A variable (delay_period) is used to contain the time in seconds between each step of the servo.

The while loop will just continue forever or until the program is interrupted by pressing CTRL-C. Within the loop there are two near identical 'for' loops. The first counts the angle up from 0 to 180 and the second sets the servo angle to 180 - angle, moving the servo arm back from 180 to 0 degrees.

To install the software, connect to your Pi using SSH and then type the command:

```
$ nano servo.py
```

Paste the code above into the editor and then do CTRL-X and Y to save the file.

A screenshot of a terminal window showing the GNU nano 2.2.6 text editor. The editor is editing a file named 'servo.py'. The code inside the file is as follows:

```
# Servo Control
import time
def set(property, value):
    try:
        f = open("/sys/class/rpi-pwm/pwm0/" + property, 'w')
        f.write(value)
        f.close()
    except:
        print("Error writing to: " + property + " value: " + value)

def setServo(angle):
    set("servo", str(angle))

set("delayed", "0")
set("mode", "pwm [servo] audio")
set("servo_max", "180")
set("active", "1")
```

The bottom of the screen shows the nano editor's command shortcuts: ^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, and ^T To Spell.

To run the servo program just type the following command into your SSH window:

```
$ python servo.py
```

The servo should start to move straight away.

Test & Configure

If you want to make the servo move faster, try changing `delay_period` to a smaller value, say 0.001. Then to slow it down try increasing it to 0.1.

If you want to control more than one servo, then the easiest way to do this is to use something like the [Adafruit I2C 16 channel servo / pwm controller \(http://adafru.it/815\)](http://adafru.it/815). [This tutorial \(http://adafru.it/aPh\)](http://adafru.it/aPh) explains how to use it.