<https://www.npmjs.com/package/rpi-gpio>

# [rpi-gpio](https://www.npmjs.com/package/rpi-gpio)

양식의 맨 위

양식의 맨 아래

Control Raspberry Pi GPIO pins with io.js / node.js

## Supported hardware

* Raspberry Pi 1 Model A
* Raspberry Pi 1 Model A+
* Raspberry Pi 1 Model B
* Raspberry Pi 1 Model B+
* Raspberry Pi 2 Model B

### Unknown - please raise an issue to let us know if this works for you

* Raspberry Pi 3 Model B
* Raspberry Pi Zero

## Setup

See this guide on how to get [node.js running on Raspberry Pi](https://learn.adafruit.com/node-embedded-development/installing-node-dot-js).

This module can then be installed with npm:

npm install rpi-gpio

### Dependency

If you are having trouble installing this module make sure you are running gcc/g++ -v 4.8 or higher. [Here](https://github.com/fivdi/onoff/wiki/Node.js-v4-and-native-addons) is an installation guide.

## Usage

Firstly, make make sure you are running your application as root or with sudo, else the Raspberry Pi will not let you output to the GPIO.

Before you can read or write, you must use setup() to open a channel, and must specify whether it will be used for input or output. Having done this, you can then read in the state of the channel or write a value to it using read() or write().

All of the functions relating to the pin state within this module are asynchronous, so where necessary - for example in reading the value of a channel - a callback must be provided. This module inherits the standard [EventEmitter](http://nodejs.org/api/events.html), so you may use its functions to listen to events.

Please note that there are two different and confusing ways to reference a channel; either using the Raspberry Pi or the BCM/SoC naming schema (sadly, neither of which match the physical pins!). This module supports both schemas, with Raspberry Pi being the default. Please see [this page](http://elinux.org/RPi_Low-level_peripherals) for more details.

## API

### Methods

#### setup(channel [, direction, edge], callback)

Sets up a channel for read or write. Must be done before the channel can be used.

* channel: Reference to the pin in the current mode's schema.
* direction: The pin direction, pass either DIR\_IN for read mode or DIR\_OUT for write mode. You can also pass DIR\_LOW or DIR\_HIGH to use the write mode and specify an initial state of 'off' or 'on' respectively. Defaults to DIR\_OUT.
* edge: Interrupt generating GPIO chip setting, pass in EDGE\_NONE for no interrupts, EDGE\_RISING for interrupts on rising values, EDGE\_FALLING for interrupts on falling values or EDGE\_BOTH for all interrupts. Defaults to EDGE\_NONE.
* callback: Provides Error as the first argument if an error occurred.

#### read(channel, callback)

Reads the value of a channel.

* channel: Reference to the pin in the current mode's schema.
* callback: Provides Error as the first argument if an error occured, otherwise the pin value boolean as the second argument.

#### write(channel, value [, callback])

Writes the value of a channel.

* channel: Reference to the pin in the current mode's schema.
* value: Boolean value to specify whether the channel will turn on or off.
* callback: Provides Error as the first argument if an error occured.

#### setMode(mode)

Sets the channel addressing schema.

* mode: Specify either Raspberry Pi or SoC/BCM pin schemas, by passing MODE\_RPI or MODE\_BCM. Defaults to MODE\_RPI.

#### input()

Alias of read().

#### output()

Alias of write().

#### destroy()

Tears down any previously set up channels.

#### reset()

Tears down the module state - used for testing.

### Events

See Node [EventEmitter](http://nodejs.org/api/events.html) for documentation on listening to events.

#### change

Emitted when the value of a channel changed

* channel
* value

## Examples

### Setup and read the value of a pin

var gpio = require('rpi-gpio');

gpio.setup(7, gpio.DIR\_IN, readInput);

function readInput() {

    gpio.read(7, function(err, value) {

        console.log('The value is ' + value);

    });

}

### Setup and write to a pin

var gpio = require('rpi-gpio');

gpio.setup(7, gpio.DIR\_OUT, write);

function write() {

    gpio.write(7, true, function(err) {

        if (err) throw err;

        console.log('Written to pin');

    });

}

### Setup and write to a pin that starts as on

This example shows how to setup the pin for write mode with the default state as "on". Why do this? It can sometimes be useful to reverse the default initial state due to wiring or uncontrollable circumstances.

var gpio = require('rpi-gpio');

gpio.setup(7, gpio.DIR\_HIGH, write);

function write() {

    gpio.write(7, false, function(err) {

        if (err) throw err;

        console.log('Written to pin');

    });

}

### Listen for changes on a pin

var gpio = require('rpi-gpio');

gpio.on('change', function(channel, value) {

    console.log('Channel ' + channel + ' value is now ' + value);

});

gpio.setup(7, gpio.DIR\_IN, gpio.EDGE\_BOTH);

### Unexport pins opened by the module when finished

var gpio = require('../rpi-gpio');

gpio.on('export', function(channel) {

    console.log('Channel set: ' + channel);

});

gpio.setup(7, gpio.DIR\_OUT);

gpio.setup(15, gpio.DIR\_OUT);

gpio.setup(16, gpio.DIR\_OUT, pause);

function pause() {

    setTimeout(closePins, 2000);

}

function closePins() {

    gpio.destroy(function() {

        console.log('All pins unexported');

    });

}

### Voltage cycling a pin

This example shows how to set up a channel for output mode. After it is set up, it executes a callback which in turn calls another, causing the voltage to alternate up and down three times.

var gpio = require('rpi-gpio');

var pin   = 7;

var delay = 2000;

var count = 0;

var max   = 3;

gpio.setup(pin, gpio.DIR\_OUT, on);

function on() {

    if (count >= max) {

        gpio.destroy(function() {

            console.log('Closed pins, now exit');

        });

        return;

    }

    setTimeout(function() {

        gpio.write(pin, 1, off);

        count += 1;

    }, delay);

}

function off() {

    setTimeout(function() {

        gpio.write(pin, 0, on);

    }, delay);

}

### Using flow control modules

Due to the asynchronous nature of this module, using an asynchronous flow control module can help to simplify development. This example uses [async.js](https://github.com/caolan/async) to turn pins on and off in series.

var gpio = require('rpi-gpio');

var async = require('async');

async.parallel([

    function(callback) {

        gpio.setup(7, gpio.DIR\_OUT, callback)

    },

    function(callback) {

        gpio.setup(15, gpio.DIR\_OUT, callback)

    },

    function(callback) {

        gpio.setup(16, gpio.DIR\_OUT, callback)

    },

], function(err, results) {

    console.log('Pins set up');

    write();

});

function write() {

    async.series([

        function(callback) {

            delayedWrite(7, true, callback);

        },

        function(callback) {

            delayedWrite(15, true, callback);

        },

        function(callback) {

            delayedWrite(16, true, callback);

        },

        function(callback) {

            delayedWrite(7, false, callback);

        },

        function(callback) {

            delayedWrite(15, false, callback);

        },

        function(callback) {

            delayedWrite(16, false, callback);

        },

    ], function(err, results) {

        console.log('Writes complete, pause then unexport pins');

        setTimeout(function() {

            gpio.destroy(function() {

                console.log('Closed pins, now exit');

            });

        }, 500);

    });

};

function delayedWrite(pin, value, callback) {

    setTimeout(function() {

        gpio.write(pin, value, callback);

    }, 500);

}

## Contributing

Contributions are appreciated, both in the form of bug reports and pull requests.

Due to the nature of this project it can be quite time-consuming to test against real hardware, so the automated test suite is all the more important. I will not accept any pull requests that cause the build to fail, and probably will not accept any that do not have corresponding test coverage.

You can run the tests with npm:

npm test

and create a coverage report with:

npm run coverage

There is also an integration test that you can run on Raspberry Pi hardware, having connected two GPIO pins across a resistor. The command to run the test will provide further instructions on how to set up the hardware:

npm run int

The tests use [mochajs](http://mochajs.org) as the test framework, and [Sinon.JS](http://sinonjs.org) to stub and mock out file system calls.

### [Manage permissions for the whole team](https://www.npmjs.com/features?utm_source=house&utm_medium=package%20page&utm_term=Manage%20permissions%20for%20the%20whole%20team&utm_content=hed&utm_campaign=orgs#features-plans-and-enterprise-pane)

Manage developer teams with varying permissions and multiple projects. [Learn more about Private Packages and Organizations…](https://www.npmjs.com/features?utm_source=house&utm_medium=package%20page&utm_term=Manage%20permissions%20for%20the%20whole%20team&utm_content=body&utm_campaign=orgs#features-plans-and-enterprise-pane)



[how? learn more](https://www.npmjs.com/package/rpi-gpio/tutorial)

* [[](https://www.npmjs.com/~jamesbarwell)jamesbarwell](https://www.npmjs.com/%7Ejamesbarwell) published 4 months ago
* **0.8.1** is the latest of 19 releases
* [github.com/JamesBarwell/rpi-gpio.js](https://github.com/JamesBarwell/rpi-gpio.js)
* [MIT](http://spdx.org/licenses/MIT.html) ®

### Collaborators [list](https://www.npmjs.com/package/rpi-gpio/access)

* [](https://www.npmjs.com/~jamesbarwell)

### Stats

* **76** downloads in the last day
* **391** downloads in the last week
* **1,669** downloads in the last month
* [11 open issues](https://github.com/JamesBarwell/rpi-gpio.js/issues) on GitHub
* [5 open pull requests](https://github.com/JamesBarwell/rpi-gpio.js/pulls) on GitHub

### Try it out

* [Test rpi-gpio in your browser.](https://runkit.com/npm/rpi-gpio)

### Keywords

None

### Dependencies (3)

[async](https://www.npmjs.com/package/async), [debug](https://www.npmjs.com/package/debug), [epoll](https://www.npmjs.com/package/epoll)

### [Dependents (26)](https://www.npmjs.com/browse/depended/rpi-gpio)

[rpi-gpio-buttons](https://www.npmjs.com/package/rpi-gpio-buttons), [rpi-pin-events](https://www.npmjs.com/package/rpi-pin-events), [rpi-fan-controller](https://www.npmjs.com/package/rpi-fan-controller), [rpi-profalux-shutters](https://www.npmjs.com/package/rpi-profalux-shutters), [mewo-pi-device](https://www.npmjs.com/package/mewo-pi-device), [homestar-gpio](https://www.npmjs.com/package/homestar-gpio), [remotecamera](https://www.npmjs.com/package/remotecamera), [nodefm-rpi](https://www.npmjs.com/package/nodefm-rpi), [pinhead](https://www.npmjs.com/package/pinhead), [zetta-led-raspberrypi-driver](https://www.npmjs.com/package/zetta-led-raspberrypi-driver), [linux-lock-pi](https://www.npmjs.com/package/linux-lock-pi), [housecontroller](https://www.npmjs.com/package/housecontroller), [zetta-microphone-raspberrypi-driver](https://www.npmjs.com/package/zetta-microphone-raspberrypi-driver), [rpi\_cpu\_usage\_monitor](https://www.npmjs.com/package/rpi_cpu_usage_monitor), [pi-pir-sensor](https://www.npmjs.com/package/pi-pir-sensor), [rpi-jenkins-light](https://www.npmjs.com/package/rpi-jenkins-light), [RaspiKids](https://www.npmjs.com/package/RaspiKids), [mewo](https://www.npmjs.com/package/mewo), [raspid](https://www.npmjs.com/package/raspid), [raspberry-led-ci](https://www.npmjs.com/package/raspberry-led-ci), [iobroker.rpi2](https://www.npmjs.com/package/iobroker.rpi2), [desktop-controller](https://www.npmjs.com/package/desktop-controller), [rpi-gpio-promise](https://www.npmjs.com/package/rpi-gpio-promise), [and more](https://www.npmjs.com/browse/depended/rpi-gpio)

[Yellow Pages Canada](https://jobs-emplois.yp.ca/jobs/d%C3%A9veloppeur-full-stack-mean-1114) is hiring. [View more…](https://www.npmjs.com/whoshiring)

<https://github.com/jperkin/node-rpio>

<https://makerslabntu.wordpress.com/2015/08/22/how-to-use-raspberry-pi-to-control-a-servo-via-the-web/>

<http://desertbot.io/how-to-control-a-servo-array-with-nodejs/>

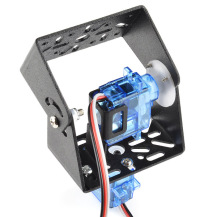
<https://bloggerbrothers.com/2017/03/04/controlling-servos-from-your-pi-no-extra-hardware-needed-using-nodejs/>

[Electronics](https://bloggerbrothers.com/category/electronics/) · [nodejs](https://bloggerbrothers.com/category/nodejs/) · [raspberry pi](https://bloggerbrothers.com/category/raspberry-pi/)

# Controlling Servos from your PI (No extra hardware needed) using NodeJS

[March 4, 2017](https://bloggerbrothers.com/2017/03/04/controlling-servos-from-your-pi-no-extra-hardware-needed-using-nodejs/) [mamacker](https://bloggerbrothers.com/author/mamacker/)

[](https://bloggerbrothers.com/2017/02/08/servo-triggers/servo/)

[](https://bloggerbrothers.com/2017/02/08/servo-triggers/pantilt-1l/)

[](https://bloggerbrothers.com/2017/02/08/servo-triggers/wriw9h/)

[](https://bloggerbrothers.com/2017/02/08/servo-triggers/gpma1217servoa/)

# For a little servo background…

See my article on the servo-trigger:

[Servo Triggers – Quickest, Fastest, Easiest way to make your servos dance](https://bloggerbrothers.com/2017/02/08/servo-triggers/)

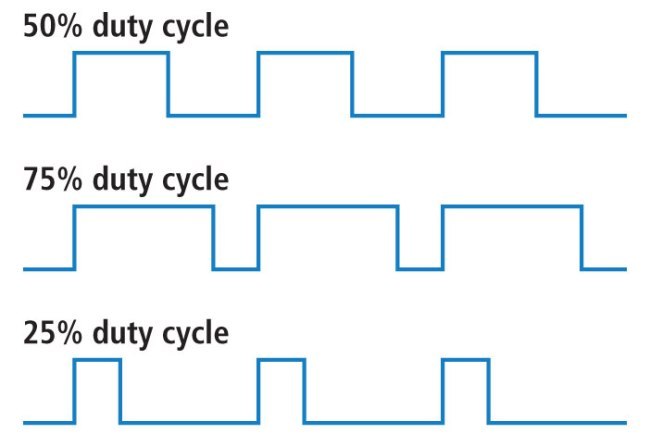
# To install node…

See my NodeJS installation article:

[Installing NodeJS on a Raspberry PI](https://bloggerbrothers.com/2017/03/04/installing-nodejs-on-a-raspberry-pi/)

# Then… a little something on PWM

Ok, a little background:  Controlling servos works by using something called **PWM**or **Pulse Width Modulation.**  Servos have little computers in them that expect a signal that looks something like this:



So now imagine that a servo can turn 180 degrees.  If the controller of the servo wishes for the servo to position itself at that full rotation of 180 that blue line, which represents the voltage on the raspberry pi’s pin, would be “high” the whole time.  If it wanted to be at 0 degrees, that blue line would be “low” the whole time.  Then for the values in between, that line is high for the fraction of 180 the angle represents.  The “25% duty cycle”, is an example of the PI wishing to get the servo to go to: 180/4 = 45 degrees.

Now that is a waaaay short explanation.  There is a better one from the folks over at Sparkfun:

<https://learn.sparkfun.com/tutorials/pulse-width-modulation>

# Installing the software…

First, make sure you have **git** installed:

sudo apt-get install git

Then install the repo that contains the magic that is able to expose PWM to user-space (a term for lower permission access):

git clone https://github.com/richardghirst/PiBits.git

cd PiBits

cd [ServoBlaster](https://github.com/richardghirst/PiBits/tree/master/ServoBlaster)

cd [user](https://github.com/richardghirst/PiBits/tree/master/ServoBlaster/user)

make

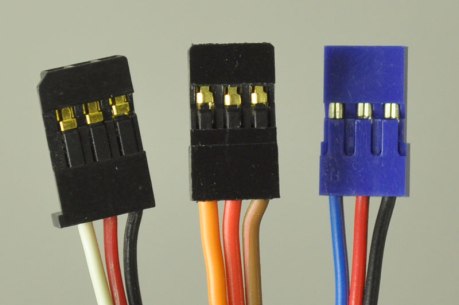
sudo make install

That installs the servo driver software.

Next we’re going to install a little script to make interacting with that driver easy:

npm install pi-servo-blaster.js

# Wiring up a servo… or lots of servos…

[](https://www.pololu.com/blog/16/electrical-characteristics-of-servos-and-introduction-to-the-servo-control-interface)

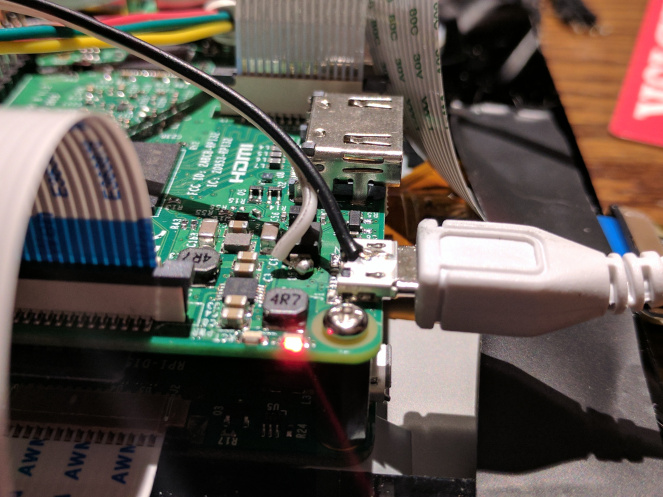
All servos have three wires:

1. Ground – Black or Brown
2. Power (usually 3-6V) – Red
3. Signal – White, Orange, or Blue

The trouble with servos is that they draw lots of power.  They are little motors that spin really fast, and gear up to drive with lots of force.  Your PI cannot source that current through its pins because those pins are, in most cases, directly connected to microscopic wires or transistors inside the main chip.  Those micro wires can only handle around 50ma – your servo needs upwards of 500ma!

# Method 1: Tap into the power entering the PI

So you need to get right to the source of the power… before it gets to the chip.  Like this:



Here you are taking the power straight from the power supply for the whole PI system.  This works if there is enough amperage on your supply that isn’t yet used by the rest of your system.

In that image, the Black of the servo is soldered to the housing of the USB connector.  The White(which is later connected to the servo red) is soldered to a point that is directly connected to the hot on the USB connector… at 5V.

See the video below to see the whole wiring harness in action.

# Method 2: Use a separate power supply.

The other, more reliable, method is to use a separate supply.  When using this method – a crucial thing to keep in mind is the Grounds of the two systems must be tied together.  To accomplish this, feel free to use one of the ground pins on the PI.

I will normally use something like this:



The standard size for most barrel connectors is 5.5mm, with an inside diameter of its pin is 2.5mm.  There is a great overview article about barrel connectors here:

<https://learn.sparkfun.com/tutorials/connector-basics/power-connectors>

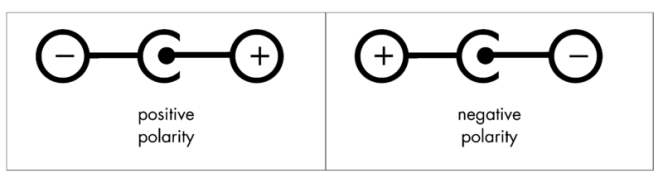
In any case, if you get yourself a couple of those – you’ll be able to use most of these:

[](https://www.adafruit.com/products/276)

<https://www.adafruit.com/products/276>

If you use the screw terminals in the above image, clamp two wires into the negative side, and use one of them to tie into the Raspberry PI’s Ground pin.

Take care to make sure the voltage you provide the servo is between 3V and 6V.  Read the back of the power supply, it will have all the details.  Also, just in case you get a weird one, check the polarity. It should have the picture seen here on the left.

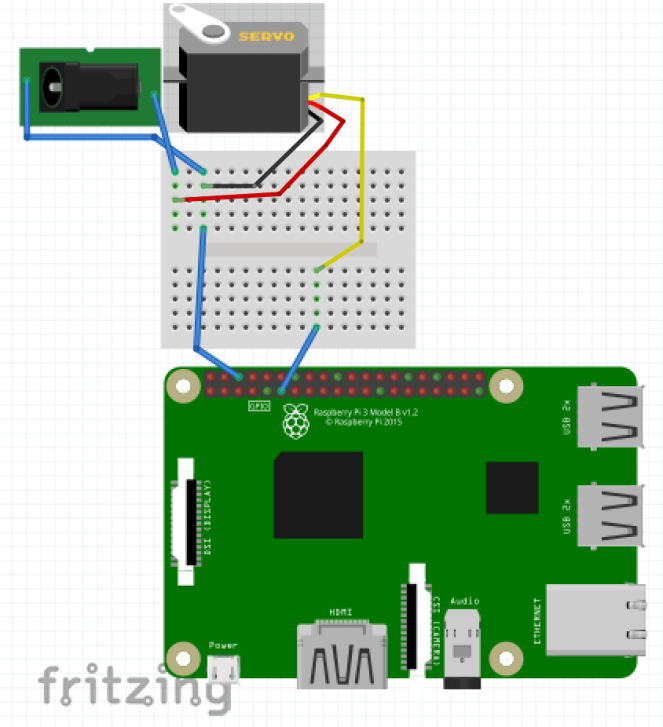
[](https://learn.sparkfun.com/tutorials/connector-basics/power-connectors) (Source: [Sparkfun’s Article](https://learn.sparkfun.com/tutorials/connector-basics/power-connectors))

Finally, make sure it can provide enough current to the device.  Each servo is different, but a simple one that is moving without load takes about 180ma.  So there is probably a surge of up to 300, and if you put a load on the little guy, it could spike even higher.

For a good list of available servos and their specifications see:

<https://www.servocity.com/servos/hitec-servos#standard>

# Wiring it up…



Note, this works for doing lots of servos too.  In fact, the raspberry pi can control up to 8, the only other limit you’ll run into is that of your power supply.

# Finally… some code!

This is a simple program to “sweep” the servo.  ServoBlaster only gives you access to control the PWM duration. So, as you can see in the code, I wrote a tiny function that converts **angle** to PWM percentage.

Save this into: index.js

var piblaster = require('pi-servo-blaster.js');

function angleToPercent(angle) {

return Math.floor((angle/180) \* 100);

}

var curAngle = 0;

var direction = 1;

setInterval(() => {

piblaster.setServoPwm("**P1-11**", angleToPercent(curAngle) + "%");

console.log("Setting angle at: ", curAngle, angleToPercent(curAngle));

curAngle += direction;

// Change direction when it exceeds the max angle.

if (curAngle >= 180) {

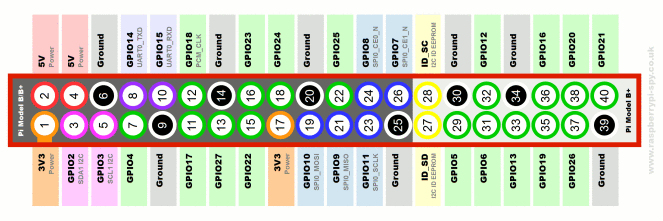
direction = -1;

} else if (curAngle <= 0) {

direction = 1;

}

}, 10);

For this to work, you should use the control pin which is on the left and six down from the top.  Just like we did in the Fritzing up above.

In this case the pin is labeled **11**, and **GPIO17.**Then notice the ground pin was tied into the barrel connector.

The run the code like this:

node index.js

You should see the servo sweep like this:

If you have trouble,  add a comment and I’ll try to add more detail in the article for anything that gets in your way.

To find out more about this library – check out the author’s site:

<https://github.com/richardghirst/PiBits/tree/master/ServoBlaster>

The javascript library we use to interact with this – just writes to the FIFO file like his examples do.

# Warnings!

**First –** some servos can be very strong.  Please take care not to get pinched.

**Second** – power is going to be your next biggest gotcha.  This article is about getting you around all that.

**Third** – this does not work on the PI Zero!  Pi zero halted when I attempted to use this library… I haven’t tracked down why just yet.

**Fourth** – there are many different kinds of servos.  Not all are 180 degree, some are 90, some are 360, some are continuous rotation.  Each of these cases will need slightly different code in the **angleToPercent** function.  Here is an example of that earlier code where the angle is a little easier to munge.

var piblaster = require('pi-servo-blaster.js');

const MAX\_ANGLE = **180**;

function angleToPercent(angle) {

return Math.floor((angle/MAX\_ANGLE) \* 100);

}

var curAngle = 0;

var direction = 1;

setInterval(() => {

piblaster.setServoPwm("**P1-11**", angleToPercent(curAngle) + "%");

console.log("Setting angle at: ", curAngle, angleToPercent(curAngle));

curAngle += direction;

// Change direction when it exceeds the max angle.

if (curAngle >= MAX\_ANGLE) {

direction = -1;

} else if (curAngle <= 0) {

direction = 1;

}

}, 10);

# If all of this fails…

There is hardware that can help.  First, triggers like these:

[Servo Triggers – Quickest, Fastest, Easiest way to make your servos dance](https://bloggerbrothers.com/2017/02/08/servo-triggers/)

There are also boards that can help separate the power for you.  This one from adafruit has some great tutorials.

<https://www.adafruit.com/products/2327>

And other tutorials:

<https://learn.adafruit.com/adafruits-raspberry-pi-lesson-8-using-a-servo-motor?view=all>

Have fun!