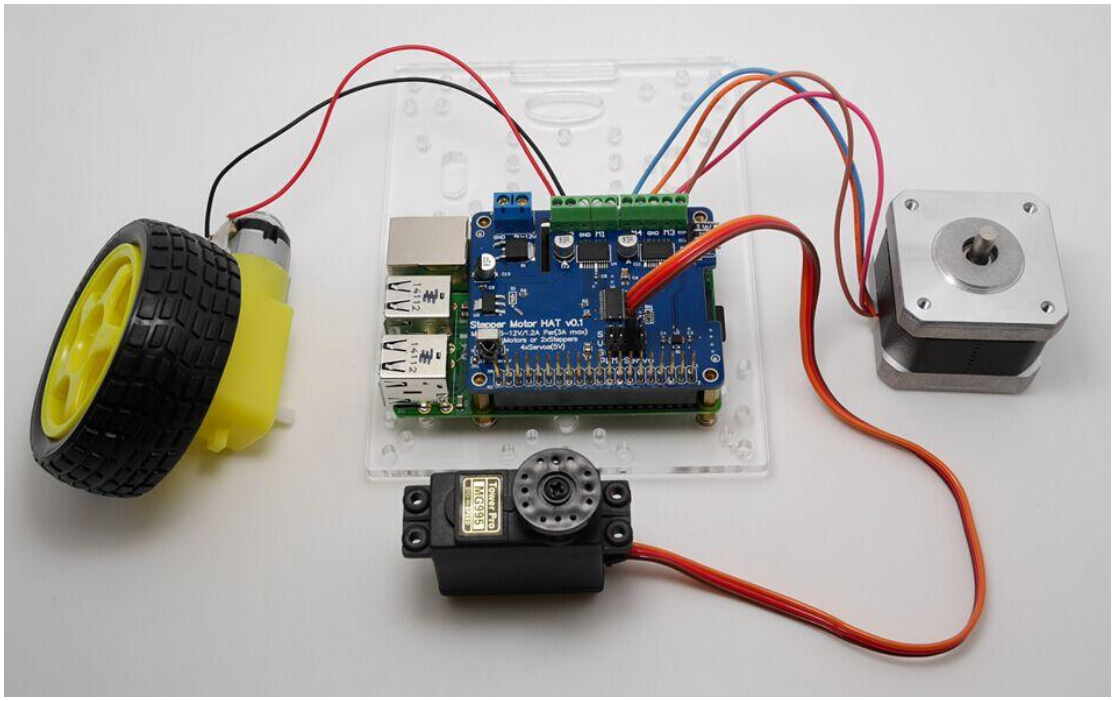
Stepper Motor HAT for Raspberry Pi

Overview



Let your robotic dreams come true with the new DC+Stepper Motor HAT. This Raspberry Pi add-on is perfect for any motion project as it can drive up to 4 DC or 2 Stepper motors with full PWM speed control,It also adds the capability to control 4 Servos with perfect timing.

Powering Motors

Motors need a lot of energy, especially cheap motors since they're less efficient.

Voltage requirements:

The first important thing to figure out what voltage the motor is going to use. If you're lucky your motor came with some sort of specifications. Some small hobby motors are only intended to run at 1.5V, but its just as common to have 6-12V motors. The motor controllers on this HAT are designed to run from **5V to 12V**.

MOST 1.5-3V MOTORS WILL NOT WORK or will be damaged by 5V power

Current requirements:

The second thing to figure out is how much current your motor will need. The motor driver chips that come with the kit are designed to provide up to 1.2 A per motor, with 3A peak current. Note that once you head towards 2A you'll probably want to put a heat-sink on the motor driver, otherwise you will get thermal failure, possibly burning out the chip.
If you don't have to take your project on the go, a 9V 1A, 12V 1A, or 12V 5A will work nicely **99% of 'weird motor problems**' are due to having a voltage mismatch (too low a voltage, too high a voltage) or not having a powerful enough supply! *Even small DC motors can draw up to 3 Amps when they stall.*

Power it up

Wire up your battery pack to the Power terminal block on the right side of the HAT. It is polarity protected but still its a good idea to check your wire polarity. Once the HAT has the correct polarity, you'll see the green LED light up
Please note the HAT does not power the Raspberry Pi, and we strongly recommend having two seperate power supplies - one for the Pi and one for the motors, as motors can put a lot of noise onto a power supply and it could cause stability problems!

Installing Software

We have a Python library you can use to control DC and stepper motors, its probably the easiest way to get started, and python has support for multithreading which can be really handy when running multiple stepper motors at onces!

Enable I2C

You will have to make I2C support work on your Pi before you begin, visit our tutorial to enable I2C in the kernel!
Before you start, you'll need to have the python smbus library installed as well as 'git', run **apt-get install python-smbus i2c-tools**

Downloading the Code from SourceForge

The easiest way to get the code onto your Pi is to hook up an Ethernet cable or with a WiFi setup, and clone it directly using 'git', which is installed by default on most distros.
Simply run the following commands from an appropriate location (ex. "/home/pi"):

```
wget
https://sourceforge.net/projects/u-geek/files/HATs/Raspi_MotorHAT/Ra
spi_MotorHAT.tar
```

```
tar xvzf Raspi_MotorHAT.tar

cd Raspi_MotorHAT
```

Install python-dev if you havent already:

```
sudo apt-get install python-dev
```

```
Location: http://jaist.dl.sourceforge.net/project/u-geek/HATs/Raspi_MotorHAT/Raspi_MotorHAT.tar
[following]
--2015-09-24 16:58:49--  http://jaist.dl.sourceforge.net/project/u-geek/HATs/Raspi_MotorHAT/Raspi_MotorHAT.tar
Resolving jaist.dl.sourceforge.net (jaist.dl.sourceforge.net)... 150.65.7.130, 2001:df0:2ed:feed::feed
Connecting to jaist.dl.sourceforge.net (jaist.dl.sourceforge.net)|150.65.7.130|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5875 (5.7K) [application/octet-stream]
Saving to: 鈥榬aspi_MotorHAT.tar鈥
Raspi_MotorHAT.tar      100%[=================================>]   5.74K  --.-KB/s   in 0.1s

2015-09-24 16:58:49 (38.5 KB/s) - 鈥榬aspi_MotorHAT.tar鈥saved [5875/5875]

pi@raspberrypi ~ $ tar xvzf Raspi_MotorHAT.tar
Raspi_MotorHAT/
Raspi_MotorHAT/DCTest.py
Raspi_MotorHAT/DualStepperTest.py
Raspi_MotorHAT/StackingTest.py
Raspi_MotorHAT/Raspi_PWM_Servo_Driver.py
Raspi_MotorHAT/StepperTest.py
Raspi_MotorHAT/Raspi_MotorHAT.py
Raspi_MotorHAT/Raspi_I2C.py
pi@raspberrypi ~ $ cd Raspi_MotorHAT
pi@raspberrypi ~/Raspi_MotorHAT $ ls
DCTest.py          Raspi_I2C.py          Raspi_PWM_Servo_Driver.py  StepperTest.py
DualStepperTest.py  Raspi_MotorHAT.py  StackingTest.py
pi@raspberrypi ~/Raspi_MotorHAT $
```

from within the Motor HAT library folder, we have a couple examples to demonstrate the different types of motors and configurations. The next few pages will explain them

Using DC Motors

DC motors are used for all sort of robotic projects.

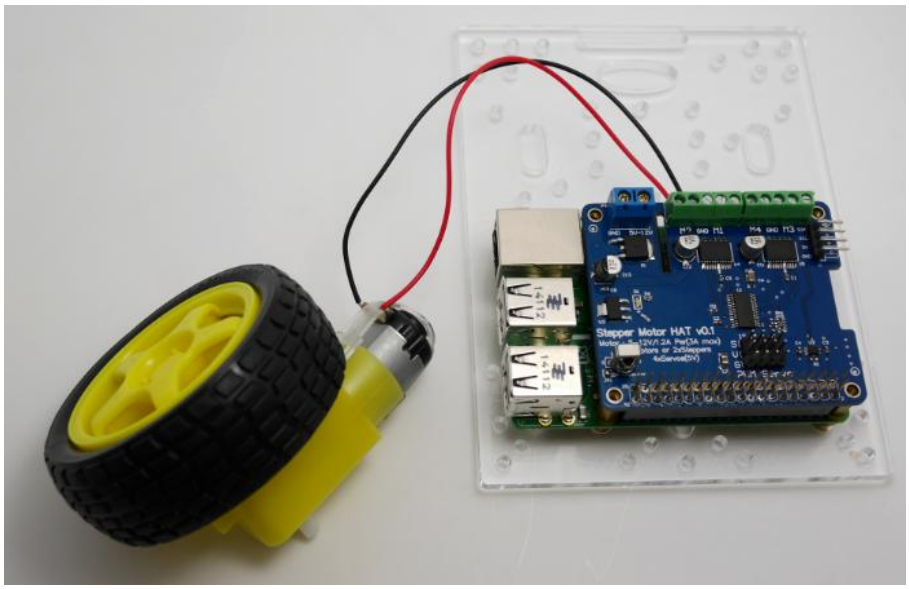
The Motor HAT can drive up to 4 DC motors bi-directionally. That means they can be driven forwards and backwards. The speed can also be varied at 0.5% increments using the high-quality built in PWM. This means the speed is very smooth and won't vary!

Note that the H-bridge chip is not meant for driving continuous loads over 1.2A or motors that peak over 3A, so this is for small motors. Check the datasheet for information about the motor to verify its OK!

Connecting DC Motors

To connect a motor, simply solder two wires to the terminals and then connect them to either the**M1**, **M2**, **M3**, or **M4.** If your motor is running 'backwards' from the way you like, just swap the wires in the terminal block

For this demo, please connect it to **M3**
Now go into the Raspi_MotorHAT / folder and run **sudo python DCTest.py** to watch your motor spin back and forth.



Note: need to external power supply to the motor power supply

DC motor control

Here's the code which shows you everything the MotorHAT library can do and how to do it.
The MotorHAT library contains a few different classes, one is the MotorHAT class itself which is the main PWM controller. You'll always need to create an object, and set the address. By default the address is 0x6F(see the stacking HAT page on why you may want to change the address)

```
# create a default object, no changes to I2C address or frequency
mh =Raspi_MotorHAT(addr=0x6F)
```

The PWM driver is 'free running' - that means that even if the python code or Pi linux kernel crashes, the PWM driver will still continue to work. This is good because it lets the Pi focus on linuxy things while the PWM driver does its PWMy things. **But it means that the motors DO NOT STOP when the python code quits**
For that reason, we strongly recommend this 'at exit' code when using DC motors, it will do its best to shut down all the motors.

```
# recommended for auto-disabling motors on shutdown!
def turnOffMotors():
```

```
        mh.getMotor(1).run(Raspi_MotorHAT.RELEASE)
        mh.getMotor(2).run(Raspi_MotorHAT.RELEASE)
        mh.getMotor(3).run(Raspi_MotorHAT.RELEASE)
        mh.getMotor(4).run(Raspi_MotorHAT.RELEASE)

atexit.register(turnOffMotors)
```

Creating the DC motor object

OK now that you have the motor HAT object, note that each HAT can control up to 4 motors. And you can have multiple HATs!
To create the actual DC motor object, you can request it from the MotorHAT object you created above with **getMotor(*num*)** with a value between 1 and 4, for the terminal number that the motor is attached to

```
myMotor = mh.getMotor(3)
```

DC motors are simple beasts, you can basically only set the speed and direction.

Setting DC Motor Speed

To set the speed, call **setSpeed(*speed*)** where speed varies from 0 (off) to 255 (maximum!). This is the PWM duty cycle of the motor

```
# set the speed to start, from 0 (off) to 255 (max speed)
myMotor.setSpeed(150)
```

Setting DC Motor Direction

To set the direction, call **run(*direction*)** where *direction* is a constant from one of the following:

- **Raspi_MotorHAT.FORWARD** - DC motor spins forward
- **Raspi _MotorHAT.BACKWARD** - DC motor spins forward
- **Raspi _MotorHAT.RELEASE** - DC motor is 'off', not spinning but will also not hold its place.

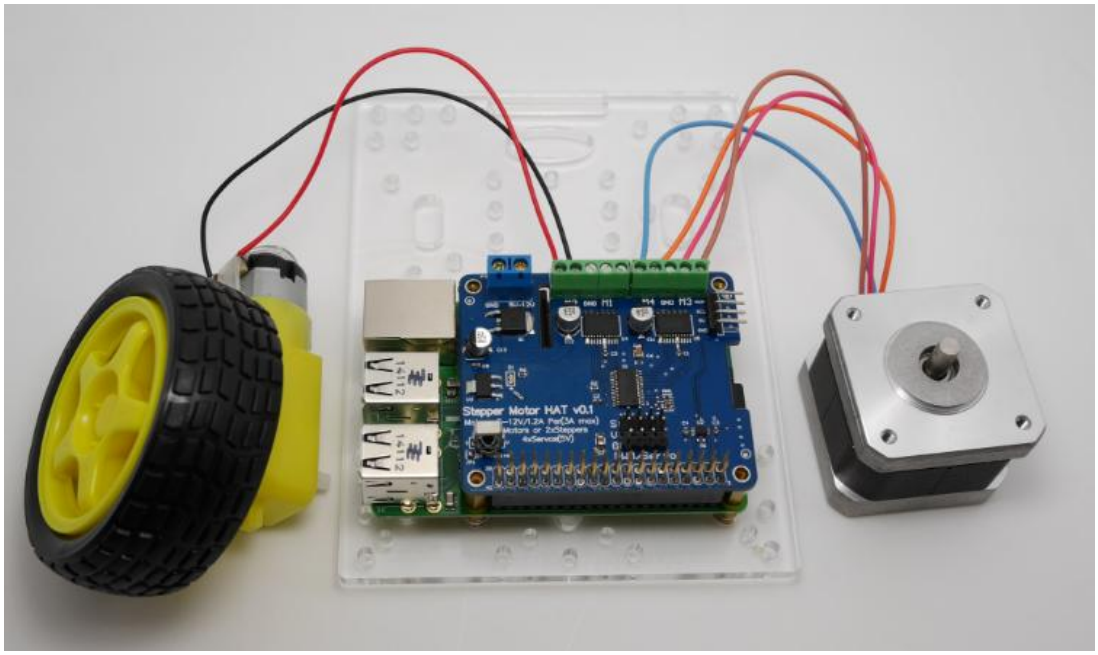
```
while (True):
        print "Forward! "
```

```
        myMotor.run(Raspi_MotorHAT.FORWARD)

        print "\tSpeed up..."
        for i in range(255):
                myMotor.setSpeed(i)
                time.sleep(0.01)

        print "\tSlow down..."
        for i in reversed(range(255)):
                myMotor.setSpeed(i)
                time.sleep(0.01)

        print "Backward! "
        myMotor.run(Raspi_MotorHAT.BACKWARD)

        print "\tSpeed up..."
        for i in range(255):
                myMotor.setSpeed(i)
                time.sleep(0.01)

        print "\tSlow down..."
        for i in reversed(range(255)):
                myMotor.setSpeed(i)
                time.sleep(0.01)

        print "Release"
        myMotor.run(Raspi_MotorHAT.RELEASE)
        time.sleep(1.0)
```

Using Stepper Motors

Stepper motors are great for (semi-)precise control, perfect for many robot and CNC projects. This HAT supports up to 2 stepper motors. The python library works identically for bi-polar and uni-polar motors

Running a stepper is a little more intricate than running a DC motor but its still very easy



Note: need to external power supply to the motor power supply

Connecting Stepper Motors

For unipolar motors: to connect up the stepper, first figure out which pins connected to which coil, and which pins are the center taps. If its a 5-wire motor then there will be 1 that is the center tap for both coils. Theres plenty of tutorials online on how to reverse engineer the coils pinout. The center taps should both be connected together to the **GND** terminal on the Motor HAT output block. then coil 1 should connect to one motor port (say **M1** or **M3**) and coil 2 should connect to the other motor port (**M2** or **M4**).

For bipolar motors: its just like unipolar motors except theres no 5th wire to connect to ground. The code is exactly the same.

For this demo, please connect it to **M1 and M2**

Now go into the Raspi_MotorHAT / folder and run **sudo python StepperTest.py** to watch your stepper motor spin back and forth.

Stepper motor control

Here's the code which shows you everything the MotorHAT library can do and how to do it.
The MotorHAT library contains a few different classes, one is the MotorHAT class itself which is the main PWM controller. You'll always need to create an object, and set the address. By default the address is 0x6F (see the stacking HAT page on why you may want to change the address)

```
# create a default object, no changes to I2C address or frequency
```

```
mh = Raspi_MotorHAT(addr = 0x6F)
```

Even though this example code does not use DC motors, it's still important to note that the PWM driver is 'free running' - that means that even if the python code or Pi linux kernel crashes, the PWM driver will still continue to work. This is good because it lets the Pi focus on linuxy things while the PWM driver does its PWMy things.
Stepper motors will not continue to move when the Python script quits, but it's still strongly recommend that you keep this 'at exit' code, it will do its best to shut down all the motors:

```
# recommended for auto-disabling motors on shutdown!
def turnOffMotors():
        mh.getMotor(1).run(Raspi_MotorHAT.RELEASE)
        mh.getMotor(2).run(Raspi_MotorHAT.RELEASE)
        mh.getMotor(3).run(Raspi_MotorHAT.RELEASE)
        mh.getMotor(4).run(Raspi_MotorHAT.RELEASE)

atexit.register(turnOffMotors)
```

Creating the Stepper motor object

OK now that you have the motor HAT object, note that each HAT can control up to 2 steppers. And you can have multiple HATs!
To create the actual Stepper motor object, you can request it from the MotorHAT object you created above with **getStepper(*steps, portnum*)** where *steps* is how many steps per rotation for the stepper motor (usually some number between 35 - 200) ith a value between 1 and 2. Port #1 is**M1** and **M2**, port #2 is **M3** and **M4**

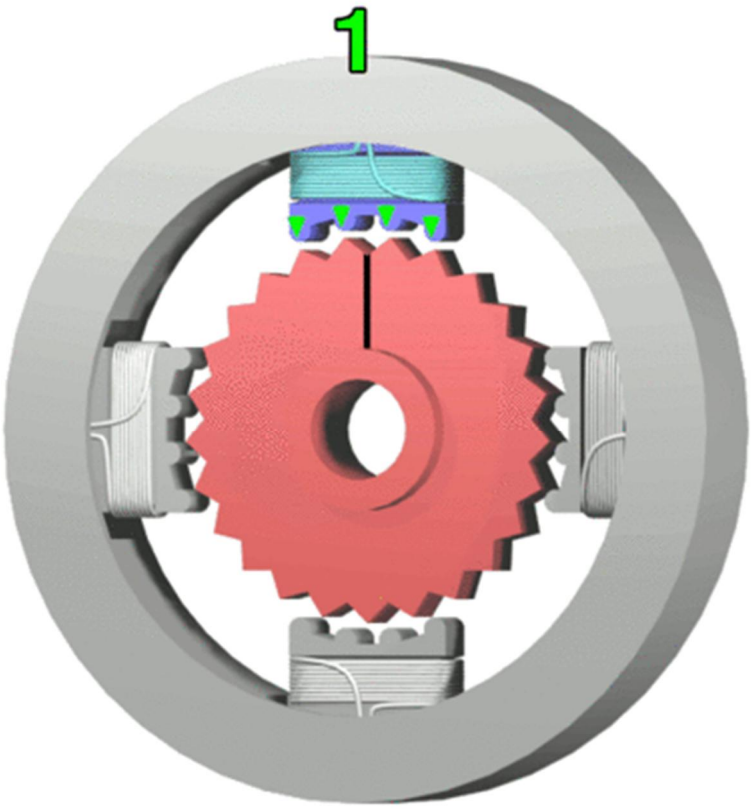
```
myStepper = mh.getStepper(200, 1)       # 200 steps/rev, motor port #1
```

Next, if you are planning to use the 'blocking' **step()** function to take multiple steps at once you can set the speed in RPM. If you end up using oneStep() then this step isn't necessary. Also, the speed is approximate as the Raspberry Pi can't do precision delays the way an Arduino would. Anyways, we wanted to keep the Arduino and Pi versions of this library similar so we kept**setSpeed()** in:

```
myStepper.setSpeed(30)                  # 30 RPM
```


Stepping

Stepper motors differ from DC motors in that the controller (in this case, Raspberry Pi) must tick each of the 4 coils in order to make the motor move. Each two 'ticks' is a step. By alternating the coils, the stepper motor will spin all the way around. If the coils are fired in the opposite order, it will spin the other way around.
If the python code or Pi crashes or stops responding, the motor will no longer move. Compare this to a DC motor which has a constant voltage across a single coil for movement.



"StepperMotor" by Wapcaplet; Teravolt.
There are four essential types of steps you can use with your Motor HAT. All four kinds will work with any unipolar or bipolar stepper motor

1. **Single Steps** - this is the simplest type of stepping, and uses the least power. It uses a single coil to 'hold' the motor in place, as seen in the animated GIF above
2. **Double Steps** - this is also fairly simple, except instead of a single coil, it has two coils on at once. For example, instead of just coil #1 on, you would have coil #1 and #2 on at once. This uses more power (approx 2x) but is stronger than single stepping (by maybe 25%)
3. **Interleaved Steps** - this is a mix of Single and Double stepping, where we use single steps interleaved with double. It has a little more strength

than single stepping, and about 50% more power. What's nice about this style is that it makes your motor appear to have 2x as many steps, for a smoother transition between steps

4. **Microstepping** - this is where we use a mix of single stepping with PWM to slowly transition between steps. It's slower than single stepping but has much higher precision. We recommend 8 microstepping which multiplies the # of steps your stepper motor has by 8.

```
while (True):
        print("Single coil steps")
        myStepper.step(100, Raspi_MotorHAT.FORWARD, Raspi_MotorHAT.SINGLE)
        myStepper.step(100, Raspi_MotorHAT.BACKWARD, Raspi_MotorHAT.SINGLE)
        print("Double coil steps")
        myStepper.step(100, Raspi_MotorHAT.FORWARD, Raspi_MotorHAT.DOUBLE)
        myStepper.step(100, Raspi_MotorHAT.BACKWARD,Raspi_MotorHAT.DOUBLE)
        print("Interleaved coil steps")
        myStepper.step(100, Raspi_MotorHAT.FORWARD,
Raspi_MotorHAT.INTERLEAVE)
        myStepper.step(100, Raspi_MotorHAT.BACKWARD,
Raspi_MotorHAT.INTERLEAVE)
        print("Microsteps")
        myStepper.step(100, Raspi_MotorHAT.FORWARD,
Raspi_MotorHAT.MICROSTEP)
        myStepper.step(100, Raspi_MotorHAT.BACKWARD,
Raspi_MotorHAT.MICROSTEP)
```

step() - blocking steps

As you can see above, you can step mulitple steps at a time with **step()**

step(*numberofsteps*, *direction*, *type*)

Where *numbersteps* is the number of steps to take, *direction* is either FORWARD or BACKWARD and *type* is **SINGLE, DOUBLE, INTERLEAVE** or **MICROSTEP**

Note that **INTERLEAVE** will move half the distance of **SINGLE** or **DOUBLE** because there are twice as many steps. And **MICROSTEP** will move 1/8 the distance because each microstep counts as a step!

This is the easiest way to move your stepper but is **blocking** - that means that the Python program is completely busy moving the motor. If you have two motors and you call these two procedures in a row:

```
stepper1.step(100, Raspi_MotorHAT.FORWARD, Raspi_MotorHAT.SINGLE)
stepper2.step(100, Raspi_MotorHAT.BACKWARD, Raspi_MotorHAT.SINGLE)
```

Then the first stepper will move 100 steps, stop, and *then* the second stepper will start moving. Chances are you'd like to have your motors moving at once!
For that, you'll need to take advantage of Python's ability to multitask with threads which you can see in **DualStepperTest.py**
The key part of the DualStepperTest example code is that we define a function that will act as a 'wrapper' for the **step()** procedure:

```
      def stepper_worker(stepper, numsteps, direction, style):
         #print("Steppin!")
         stepper.step(numsteps, direction, style)
         #print("Done")

```

We have some commented-out print statements in case you want to do some debugging.
Then, whenever you want to make the first stepper move, you can call:

```
st1 = threading.Thread(target=stepper_worker, args=(myStepper1, numsteps,
direction, stepping_style))
st1.start()
```

Which will spin up a background thread to move Stepper1 and will return immediately. You can then do the same with the second stepper:

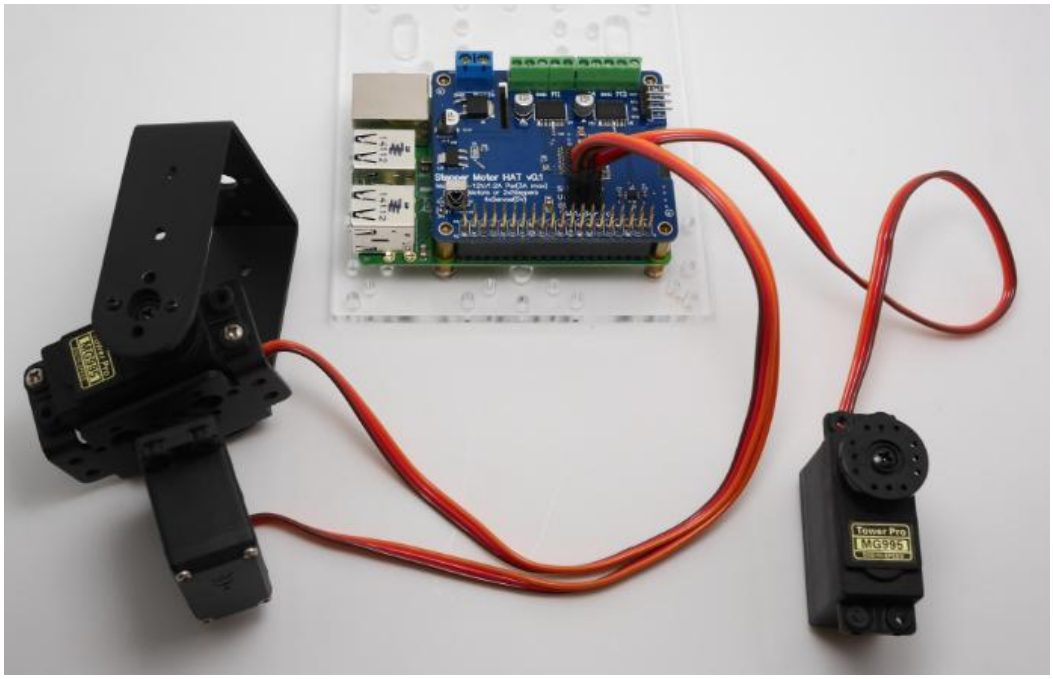
```
st2 = threading.Thread(target=stepper_worker, args=(myStepper2, numsteps,
direction, stepping_style))
st2.start()
```

You can tell when the stepper is done moving because the stepper thread you created will 'die' - test it with **st2.isAlive()** or **st2.isAlive()** - if you get **True** that means the stepper is still moving.

Using "Non-blocking" oneStep()

OK lets say you want a lot of control over your steppers, you can use the one `oneStep(direction, stepstyle)` which will make a single step in the style you request, with no delay. This will let you step exactly when you like, for the most control

Connecting Servos



Note: need to external power supply to the motor power supply

Connecting a Servo

Most servos come with a standard 3-pin female connector that will plug directly into the headers on the Motor HAT headers. Be sure to align the plug with the ground wire (usually black or brown) with the bottom row and the signal wire (usually yellow or white) on the top.

Adding More Servos

Up to 4 servos can be attached to one board. If you need to control more than 4 servos, additional boards can be stacked as described on the next page.
Note:MotoHAT's servo channel: 0,1,14,15

Stepper Servo control

For this demo, please connect it to Channel 0

Now go into the Raspi_MotorHAT / folder and run **sudo python ServoTest.py** to watch your servo spin back and forth.