Introduction to Robotics with the Raspberry Pi

DRIVING MOTORS WITH RASPBERRY PI

Jeff Cicolani
Sunday, March 9, 2017

This workshop is assuming a you are using Windows. If you are using a Mac or Linux machine, good luck… I mean, you will need to look up the proper instructions for your OS. Sorry, I'm a Windows guy, now with a little Linux, but Windows is my got to OS for general use.

Introduction

In an earlier workshop, we used the GPIO pins of the Raspberry Pi to control an LED and to receive information from an ultrasonic sensor. This week, we'll be using the GPIO pins to connect to a board designed to interact with DC motors and steppers, called a motor driver. We'll look at what a motor driver is and why they are important to what we do. We'll look at DC motors and touch, briefly, on stepper motors. Connecting and using stepper motors is a subject for a future workshop.

We will be connecting the DC motors to the motor controller and write a small program to make them turn. As part of the sample program, we'll look at how to control the speed and direction of the motors. We will also look at some of the properties of the specific motor controller selected for the workshop.

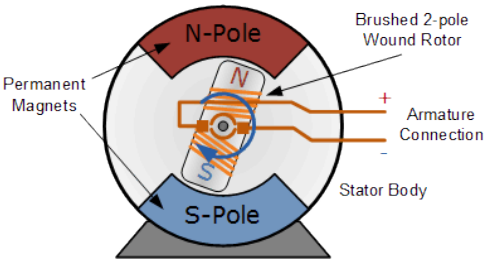
At the end of this workshop, you will have the final component needed to start building robots; motion.

Motors & Drivers

Motors convert electrical energy into rotational energy. They come in many different flavors and power virtually everything that moves. The most common type of motor is the simple DC motor. They are used in many other types of devices we call motors. For instance, a servo motor is a device that incorporates a DC motor with a potentiometer, or other feedback device, and gearing to control precise motion, be it angular or directional. We will be covering servos in a later workshop.

Types of Motors

DC Motors

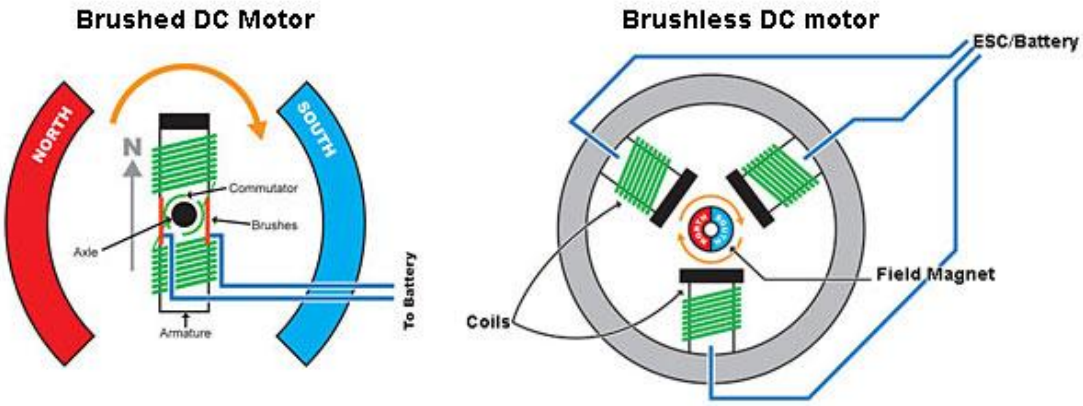


http://www.electronics-tutorials.ws/blog/pulse-width-modulation.html

DC motors consist of a series of coils within a magnetic field. When an electrical charge is placed on the coils it causes the coils to spin on their shared axis. Simple motors have the coils arranged and attached around a central shaft. As the shaft and coils spin, electrical connectivity is maintained with brushes that make contact with the shaft. The shaft is connected to other devices, such as wheels, to use the rotational energy to do something

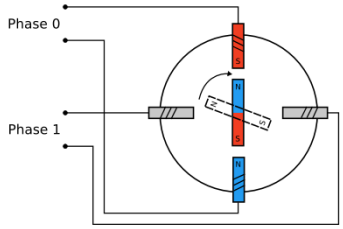
Brushless Motors

Another type of motor moves the mechanical connection to the magnets. The coils remain static and, when an electrical charge is applied, the magnets spin around the coils on a common axis. This eliminates the need for brushes and, as such, these are called brushless motors. In the hobby world, brushless motors are most commonly associated with multi-rotor aircraft. However, they are also used extensively in other areas where high speed and efficiency are required, such as in CNC spindles.



http://www.thinkrc.com/faq/brushless-motors.php

Stepper Motors



https://hackaday.io/project/978-robobarista-a-coffee-brewing-robot

All of the motors discussed so far have one or more coils working off of a single electrical charge. That charge can be positive or negative, changing the direction of the motor. Stepper motors are different. Steppers use coils with two distinct charges. This allows us to break a full rotation into multiple steps. By manipulating these charges, we can cause the motor to move to and hold position at one of these steps. This makes these motors

extremely useful for finite control in applications such as CNC machines, 3D printers, and robotics. Stepper motors will be the focus of a future workshop.

Motor Properties

There are a few things we need to keep in mind about motors with our projects. The most important are their electrical properties, specifically voltage and current.

Volts

Voltage we're already somewhat familiar with. This is a measure of electricity needed to operate our devices. The Pi is powered by 5 volts but runs on 3.3 volts. The motors we are using run on 6 volts. So, we are aware of the differences between the voltage needs of our devices. What we haven't looked at is amperage.

Amps

Amperage is a measure of current, or the amount of electricity our devices require to operate. The analogy most commonly used is that of water through a pipe. Where voltage is the size of the pipe, amperage is the amount of water flowing through it. I actually like to change the analogy to use rubber tubing. If you try to push too much water through a rubber tube, bad things can happen.

In the electronics world, this is frequently measured in smaller units of milliamps, usually noted as mA. For instance, the USB port of device is limited to 800mA of power. This happens to be the same amount of power used by the motors we selected, in a perfect world. And therein lies the rub.

Components and devices have a certain amount of power they need to work. They also have a maximum amount of power they can withstand. Since extra electrical power is converted to heat, if you exceed the maximum tolerable amperage of a device it gets hot and dies, sometimes spectacularly.

Motors and Amps

Motors are notoriously power hungry devices. They are constantly try to fulfil their purpose, to spin. When there is no load, weight or resistance, on a motor it will spin happily at its minimum current draw. However, start to add resistance and the motor will draw more and more current until it reaches the maximum it can draw, called the stall current. The stall current is, essentially, the amperage draw of a motor when the shaft is physically restrained from moving.

Motor Drivers

Most microcontrollers, microprocessors, and electronics can only handle a small amount of current. Pulling too much current and things start to burn out. Because motors usually easily exceed this maximum current, you generally don't want to connect a motor of any significant size, directly to your processor. Instead, we will use a device called a motor driver.

Motor drivers are designed for this, specific purpose. It uses the low power signal from your microcontroller to control a much larger current and/or voltage. In our case, we are using a motor controller to control 6 volts with 3.3 volts from the GPIO pins. We are doing this control through a series of components that have a much larger, 1.2A (1,200mA) current tolerance and can handle brief spikes up to 3.0A (3,000mA).

Adafruit DC and Stepper Motor HAT

The motor driver we are using for this workshop is one made by Adafruit and is available on-line at https://www.adafruit.com/products/2348. More about the motor HAT and how to use it is available on their site, as well, at https://learn.adafruit.com/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi. In fact, much of what we'll be going over came from there.

There are several reasons why this device was selected for the workshops. Not the least of which is the fact that is mounts directly to the Raspberry Pi, thus limiting the area needed for mounting electronics on the robot. As you'll learn, pretty quickly, mounting space for electronics is at a premium on most robots. Especially if you're trying to keep it rather compact. Some of the other reasons we are using this board are:

- It can control up to 4 DC motors or 2 stepper motors
- Communication is handled via the I2C serial channel allowing for multiple devices to be stacked (this is why we used the longer pins on the header)
- Because it using I2C, it has its own dedicated PWM module for controlling the motors, so we don't have to rely on the PWM on the Pi proper
- 4 H-Bridge motor control circuits with 1.2A current, 3.0A peak current, with thermal shutdown, and internal protection diodes
- 4 bi-directional motor control with 8-bit speed control (0 to 255)
- Easy connection with the use of terminal blocks
- Ready-made Python libraries

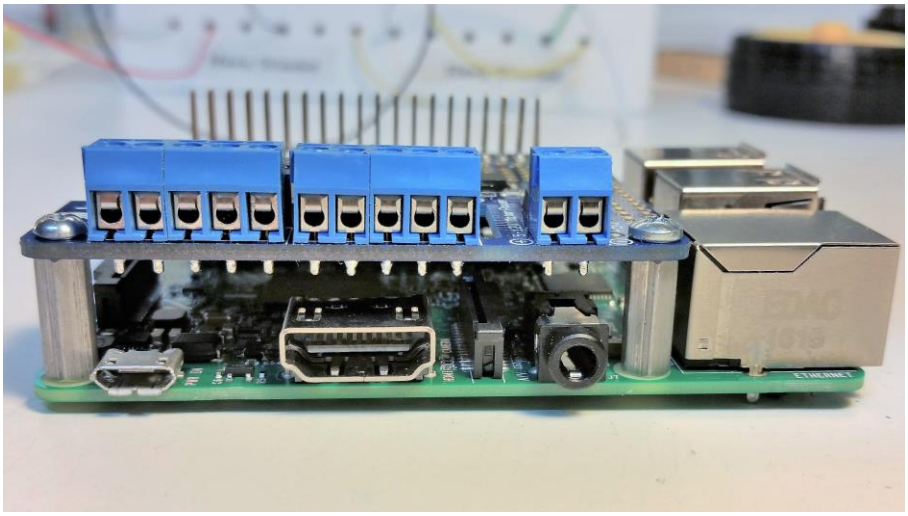
The board comes in a kit and will require soldering. If you haven't already done so, you'll need to assemble it before proceeding with the workshop. Remember, we specified longer pins for the header, so don't use the one with the kit. You can find complete instructions for assembly here: https://learn.adafruit.com/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi/assembly. Soldering is beyond the scope of this workshop, so if you're not comfortable with it, reach out to your local makerspace, they can help.

Hooking up the Motor Controller

Connecting the motor hat is pretty straight forward. Simply mount the board on the GPIO headers of the Pi. There are a couple things of note, however. First, be careful not to bend any of the pins on either the Raspberry Pi or the HAT. It is remarkably easy to do. The header pins on the motor HAT are particularly susceptible to bending.

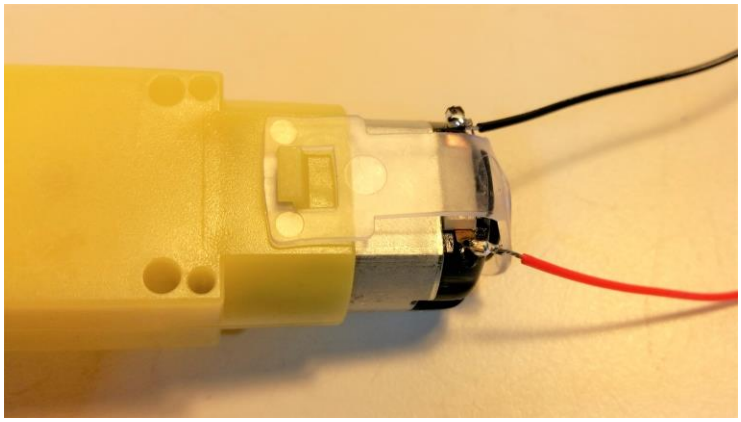
The other thing you want to be careful of is shorting the terminal blocks. You'll notice, when mounted, the solder joints are precariously close to the metal housing of the HDMI connection. There are two easy

solutions for this. First, and this is what we'll do in the workshop until you can apply the second solution, is to simply place a piece of electrical tape over the metal housings of the micro USB and HDMI connectors of the Pi. The second, and recommended fix, is to get some offsets to support that side of the HAT. Spacers and screws will also do the job. The point is, you don't want the board to sag and make contact with the housing. That probably result in a brief light show and the destruction of both the motor HAT and the Pi.



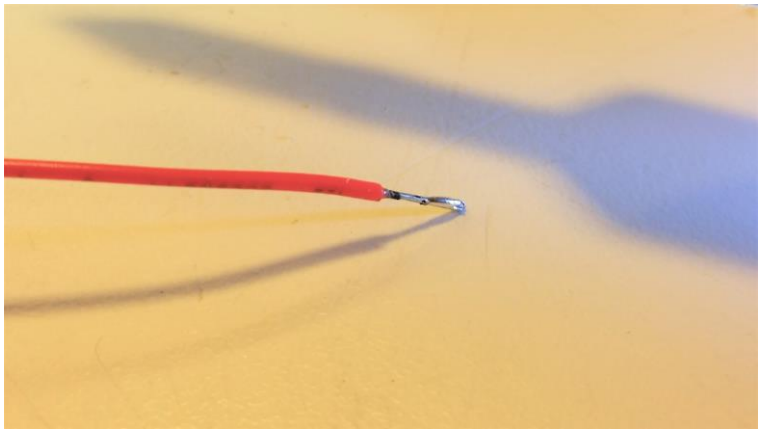
Once the board is mounted, and safely insulated from shorting, it's time to connect the motors. For the first tutorial we're only going to use one motor. The second piece of code will control two, so we might as well get them connected now.

But before we can do that, we have to prep out motors. Now, if your motors came with the leads attached, then you're ahead of the game. If not, you'll need to solder leads to your motor. I tend to use black and red wires of appropriate size for the motors in question. I also like to make sure the leads on each of the motors match. Meaning, the black wire goes to the same pole on each of the motors, and the red to the same poles on each motor. This way I don't have to second guess how things are connected, later.



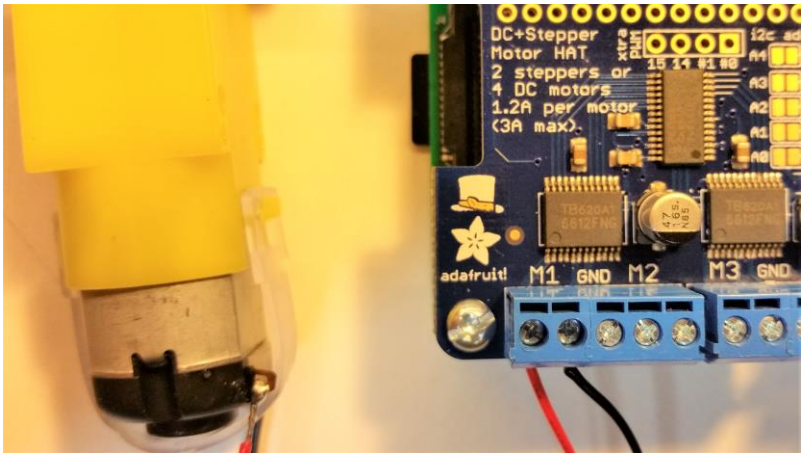
In this case, I'm using 26AWG stranded wire. There are generally two types of lead wire; stranded and solid. Solid is more rigid and excellent for jumpers or situations where there isn't going to be a lot of moment.

Stranded wire consists of multiple thinner wires housed in a single sheathing. It is more flexible and ideal for applications where there will likely be movement. Stranded wire is a little harder to work with and the ends going into the terminal blocks should be tinned, or coated in solder. This will make that end rigid and connect better in the terminal block.



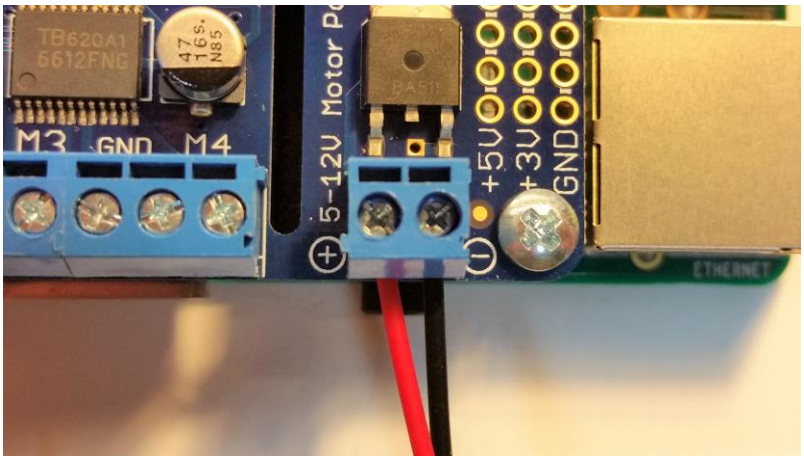
The next step is to connect the motors to the terminal block.

1. Make sure the terminal blocks are open with the screw inside the block all the way to the top. Be sure to not remove the screw. You'll need to use a pretty fine Phillips head screwdriver.
2. Insert one of the tinned leads into the hole on the side of the terminal block marked M1. It doesn't matter, at this time, which wire goes to which port, as long as both wires go to different ports for the same driver, in this case, M1.
3. Tighten the screw corresponding to the hole you inserted the wire into.
4. Repeat the procedure for the second lead from the motor.

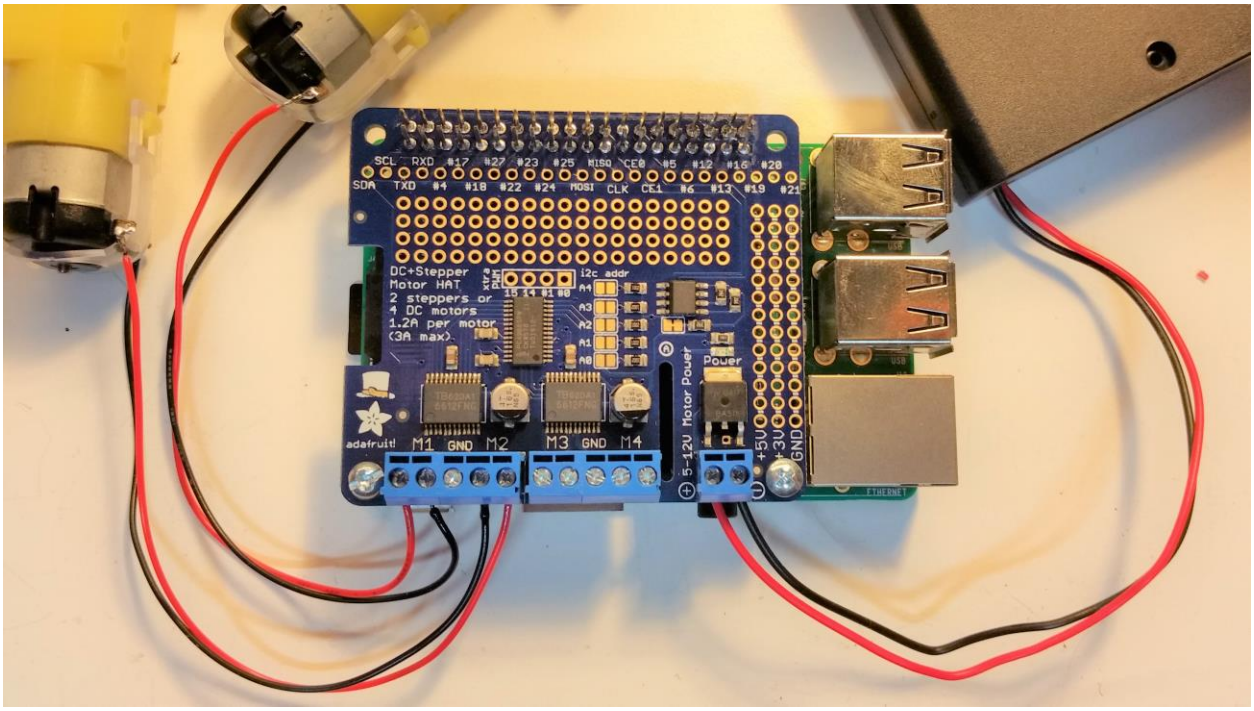


At this point, if you are so inclined, you can connect the second motor as well. I tend to reverse the order of the leads since they are destined for opposite sides of the robot. You want a FORWARD command to turn the left motor one direction and the right motor the other. If they both turn in the same direction, electrically, then the robot will just spin in place.

You'll repeat the procedure for the 4 AA battery pack to the power terminals. Make sure the red lead goes to the positive (+) side and the black goes to the negative (-) side.



With both motors and the batter pack connected, you are ready to get coding!



Turning Motors with Python

With the motor HAT mounted and your motors and motor power supply connected, it's time to boot up the Pi and log on.

Installing the Library

Once you've started up and connected to your Pi you'll need to install the Python libraries for the motor HAT. These are available from the Adafruit GitHub site.

1. Open a terminal window
2. Navigate to your Python code directory. In my case it is `TRG-RasPi_Robot`
3. Enter:

```
git clone https://github.com/adafruit/Adafruit-Motor-HAT-Python-Library.git
```

   then

```
cd Adafruit-Motor-HAT-Python-Library
```

4. Intall the Python development libraries

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

5. Install the motor HAT libraries

```
sudo python setup.py install
```

At this point your Pi is updated with the necessary libraries and it's time to get started with the code.

The Code

Before we get started code, there is one quick note of importance. Previously we've been using Python 3 for just about everything. For this workshop we will be using Python 2.7. Why? Well, the default libraries provided by Adafruit are in Python 2.7. There may be Python 3.x libraries, but we are going with the default libraries for this workshop.

As we've discovered in earlier workshops, anytime you use the GPIO pins, you will need to run your code as a super user (`sudo`). There are a couple ways of doing it. One is to save your Python code, make the file executable, and then run the program with `sudo`. This is the proper way of doing things for code you'll be running in the future. For the workshop, we'll be taking a shortcut. We'll be launching the Idle IDE using sudo from the command line. That makes any programs run from that instance of Idle run with `sudo`.

Turning a Single Motor

1. Open a terminal window. You can use the same one we were using for the installation, but as long as Idle is open, this terminal window will be locked.
2. Type

```
sudo idle
```

3. In the Idle IDE, create a new file and save it as "motors.py"
4. Enter the following code:

```
from Adafruit_MotorHAT import Adafruit_MotorHAT as amhat
```

```
from Adafruit_MotorHAT import Adafruit_DCMotor as adcm

import time

# create a motor object
mh = amhat(addr=0x60)
myMotor = mh.getMotor(1)

# set start speed
myMotor.setSpeed(150)

while True:
    # set direction
    myMotor.run(amhat.FORWARD)

    # wait 1 second
    time.sleep(1)

    # stop motor
    myMotor.run(amhat.RELEASE)

    # wait 1 second
    time.sleep(1)
```

5. Save the file
6. Press F5 to run the program

Let's go through the code.

We start by importing the objects we need from the Adafruit_MotorHAT library and assign them aliases so we don't have to write the whole name each time we use them. We also import the time library for our delays later in the code.

```
from Adafruit_MotorHAT import Adafruit_MotorHAT as amhat
from Adafruit_MotorHAT import Adafruit_DCMotor as adcm

import time
```

Next, we create an instance of the motor object. To do this, we tell Python we are using the motor HAT located at the default I2C address, 0x60. We then create a motor object for the motor attached to M1, or motor 1. This will let us access the methods and properties of the motor.

```
mh = amhat(addr=0x60)
myMotor = mh.getMotor(1)
```

Before we turn on the motors, for this program, we'll be setting the start speed at a little over half speed.

```
myMotor.setSpeed(150)
```

Now we wrap the remainder of the motor drive code in a while loop. As long as the value `True` is true, this code will keep executing.

```
while True:
```

The code to drive the motor is very simple. We drive the motor forward for one second then stop the motor for one second. The program will just keep doing this.

```
    # set direction
    myMotor.run(amhat.FORWARD)

    # wait 1 second
    time.sleep(1)

    # stop motor
    myMotor.run(amhat.RELEASE)

    # wait 1 second
    time.sleep(1)
```

Hit ctrl-c on the keyboard.

Note that the program ended, but the motor continues to turn. That's because the motor HAT is free running. This means the controller will continue with the last command received from the Pi. If we don't tell it to stop, it won't.

Now we are going to do something interesting, something we haven't done before. We'll wrap the motor drive code into a try/except block. This is a piece of code that allows us to capture any errors that occur and work with them gracefully. In this particular case, we are going to be using the try/except block to capture the KeyboardInterrupt event. This event is triggered when we use ctrl-c to exit a program.

1. Change the code for the while loop to read as follows:

```
try:
    while True:
        # set direction
        myMotor.run(amhat.FORWARD)

        # wait 1 second
        time.sleep(1)

        # stop motor
        myMotor.run(amhat.RELEASE)

        # wait 1 second
        time.sleep(1)

except KeyboardInterrupt:
    myMotor.run(amhat.RELEASE)
```

2. Run the program
3. Let it run for a moment and press ctrl-c
   The motor will now stop when the program exits.

Python now captures the KeyboardInterrupt event and executes that last line of code before executing. The code releases the motor, simply turning it off.

Turning Two Motors

That's great and dandy, but our robot is going to have two motors and we want them to be operated independently. We also want them to be able to change speed and direction.

Demanding, aren't you?

Well, we're doing that next.

In order to operate multiple motors, you simply need to create a different instance of the motor object for each one. Assuming you connected both your motors, in the following code, we will be creating two motors and giving commands to each. We are also changing both the speed and direction of the motors.

1. Create a new Python file from Idle.
2. Save the file as two_motors.py
3. Enter the following code:

```
from Adafruit_MotorHAT import Adafruit_MotorHAT as amhat,
Adafruit_DCMotor as adcm

import time

# create 2 motor objects
mh = amhat(addr=0x60)

motor1 = mh.getMotor(1)
motor2 = mh.getMotor(2)

# set start speed
motor1.setSpeed(0)
motor2.setSpeed(0)

# direction variable
direction = 0

# wrap actions in try loop
try:
    while True:
        # if direction = 1 then motor1 forward and motor2 backward
        # else motor1 backward and motor2 forward
        if direction == 0:
            motor1.run(amhat.FORWARD)
            motor2.run(amhat.BACKWARD)
        else:
            motor1.run(amhat.BACKWARD)
            motor2.run(amhat.FORWARD)

        # ramp up the speed from 1 to 255
        for i in range(255):
            j = 255-i

            motor1.setSpeed(i)
            motor2.setSpeed(j)
```

```
            time.sleep(0.01)

        # ramp down the speed from 255 to 1
        for i in reversed(range(255)):
            j = 255-i

            motor1.setSpeed(i)
            motor2.setSpeed(j)

            time.sleep(0.01)

        # wait half a second
        time.sleep(0.5)

        # change directions
        if direction == 0:
            direction = 1
        else:
            direction = 0

# kill motors and exit program on ctrl-c
except KeyboardInterrupt:
    motor1.run(amhat.RELEASE)
    motor2.run(amhat.RELEASE)
```

4. Save the file
5. Press F5 to run the program

For the most part the code is the same. What we did is added a couple for loops to count up to 255 and back down again. We created two variables to hold this value, the second one inverts the value by subtracting it from 255. We also have a variable to track the direction the motors are turning. Once both motors have sped up and down again, we change directions and do it again. We are using the exit code we did before.

That's it. Now you know how to turn motors.

Explore

At this point you have everything you need to know to build a robot.

A robot basically consists of three things;

- Interaction with the environment (sensors)
- Decision making (code), and
- Motion or actuation.

We've talked about how to get information in and out of the Raspberry Pi using two methods. We've read a sensor directly from the GPIO pins. We've also read the same sensor through an Arduino.

We've used both Python on the Pi and C on the Arduino to blink an LED.

Now we've used a motor controller to turn two motors.

All of the bases are covered.

Good luck.

Ok, we're not done yet.

In our next workshop, we'll actually build our robot, if you haven't already. I will have some chassis kits available at cost. Alternatively, you can purchase a simple chassis kit on-line. I am a fan of the simple kits from https://www.servocity.com/kits/robot-kits. Some of these kits are also available at Fry's Electronics (for the same price as the Servo City web site). These use the same motors in the workshop specification which are low voltage, low amperage DC motors. At a regular current of 190mA and a stall current of 250mA, you can connect several of them to a single channel of the motor HAT without worrying about it.

Once we have robot in hand, we'll start putting them into action. First with simple object avoidance. Then we'll get a little aggressive and go sumo with them.