MicroPython Introduction (cont.)

MCHE 201 – Spring 2018

Dr. Joshua Vaughan

Rougeou 225

joshua.vaughan@louisiana.edu

@Doc_Vaughan

In-class Exercise 9

- Connect a pushbutton
- Turn on the green LED
- Once the button is pressed the first time, turn off all LEDs.
- Then, turn on 1 LED every 1s until the button is pressed again
- When the button is pressed again, print the time elapsed between button pressed to the REPL
- If more than 5s elapses:
  - Print "You took too long!!!" to the REPL
  - Turn on only the green LED again

In-class Exercise 9 Setup

```
import pyb  # import the pyboard module
import time # import the time module

# Assign the names to the onboard LEDs
RED_LED = pyb.LED(1)
GREEN_LED = pyb.LED(2)
YELLOW_LED = pyb.LED(3)
BLUE_LED = pyb.LED(4)

# Assign the input pin to variable input_pin
# We set it up as an input with a pulldown resistor
input_pin = pyb.Pin("X4", pyb.Pin.IN,
                    pull=pyb.Pin.PULL_DOWN)

# Turn on the green LED
GREEN_LED.on()
```

In-class Exercise 9 – First Press

```
# This will loop forever, checking the button every 100ms
while (True):
    # read the state of the input
    input_state = input_pin.value()

    if (input_state):
        start_time = time.ticks_ms() # save the current time
        GREEN_LED.off()
        print("Button pressed to start.")

        # We could also have another delay here, to force a
        # longer separation between the pressing of the button to
        # start the timer and pressing it to end the timer.
        time.sleep_ms(200)

        # If true, we are waiting for the second button
        button_timing = True
```

In-class Exercise 9 – Second Press

```
if (input_state):
    CODE TO RESET LED PATTERN
    print("Elapsed time = {}ms".format(time_elapsed))

    # Set button_timing to False because we are no longer in a
    # timing part of the algorithm.
    button_timing = False

    # We could also have another delay here, …
    time.sleep_ms(200)
else:
    if time_elapsed > 5000: # >5000ms
        print("You took too long!!!")

    # We no longer want to look for the "timing" button press
    button_timing = False

    # Turn off the LEDs
    CODE TO TURN OFF ALL LEDS

elif time_elapsed > 4000: # >4000ms
    BLUE_LED.on()
elif time_elapsed > 3000: # >3000ms
    YELLOW_LED.on()
elif time_elapsed > 2000: # > 2000ms
    GREEN_LED.on()
elif time_elapsed > 1000: # > 1000ms
    RED_LED.on()
```

In-class Example 10

- Connect
  - a pushbutton
  - the servomotor

- Start the servo at 0deg

- When the pushbutton is pressed, move the servo

- Only allow this to happen once per 30seconds

In-class Exercise 10 Setup

```
import pyb  # import the pyboard module
import time # import the time module

# Assign the input pin to variable input_pin
# We set it up as an input with a pulldown resistor
input_pin = pyb.Pin("X4", pyb.Pin.IN,
                    pull=pyb.Pin.PULL_DOWN)
```

```
# For the pyboard Servo 1 is connected to X1, Servo 2
# to X2, Servo 3 to X3, and Servo 2 to X4
servo1 = pyb.Servo(1)

# Set the initial angle to 0
servo1.angle(0)
```

In-class Exercise 10 Main Loop

```
# This will loop forever, checking the button every 10ms
while (True):
    input_state = input_pin.value() # Check the button state
    if (input_state):
        print("The button was pressed. Moving the servo now.")

        # Move the servo to 30 deg
        servo1.angle(30)

        # sleep for 30seconds to disallow any other action
        # during this time.
        time.sleep(30)

        # Now, move back to 0deg
        servo1.angle(0)
    else:
        print("Button not pressed. Wait, then check again.")

    time.sleep_ms(10)           # Sleep 10 milliseconds (0.01s)
```

What will happen?

```
import pyb  # import the pyboard module
import time # import the time module

counter = 0 # Set the initial value of the counter

while (True):
    value = 1 / (10 - counter)
    print("Value = {:.4f}".format(value))

    # Sleep 1s
    time.sleep(1)

    # increment the counter by 1
    counter = counter + 1
```

Try… Except

```
counter = 0 # Set the initial value of the counter

try:
    while (True):
        value = 1 / (10 - counter)

        print("Things are running smoothly...")
        print("Value = {:.4f}".format(value))

        # Sleep 1s
        time.sleep(1)

        # increment the counter by 1
        counter = counter + 1

except: # This with catch the exception
    print("Things are not so smooth anymore.")
```

If there is an exception (error) here, then…

This will run.

Try... Except... Finally

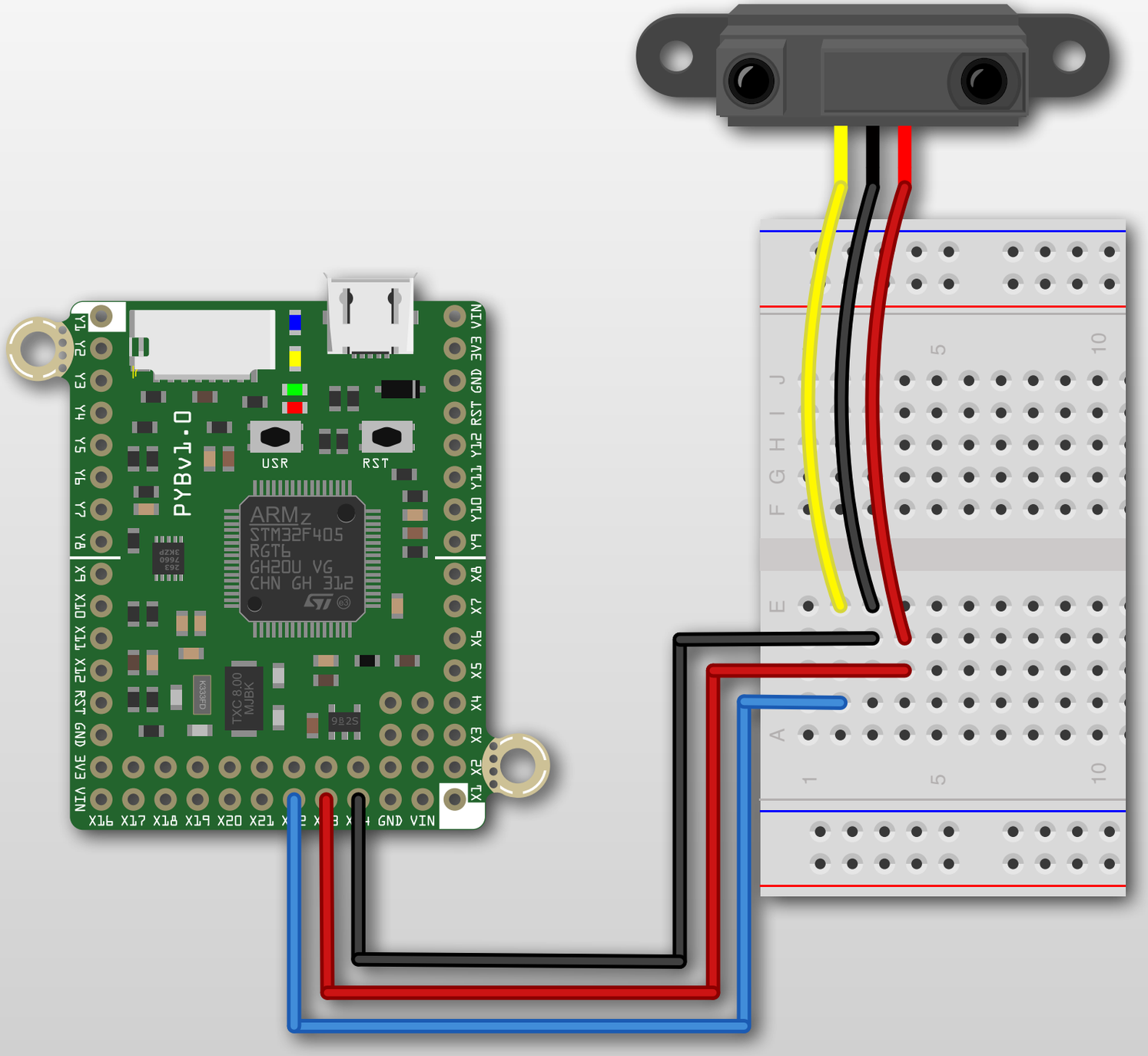
```
try:
    # Stuff to do if all is well

except: # This with catch the exception
    # Stuff to do if there is an exception

finally:
    # Stuff to do when try finishes
    # or there is an exception
```

KEY POINT!!!

- If you are controlling hardware, it is *your* responsibility to ensure it stops safely if errors occur

- For example:
  - Wrap motor control code in try… except… that would stop the motor if any syntax errors occur
  - Wrap linear actuator code similarly
  - Have a master "finally" that turns off *all* actuators if exceptions occur

IR Sensor Hardware Setup



IR Sensor Code

- It's just an analog sensor

- Distance varies between
  - 3.1V at 4cm, and
  - 0.3V at 30cm

Outside of this range, you can't trust the values

- There is a nonlinear relationship between these values

In-class Exercise 11

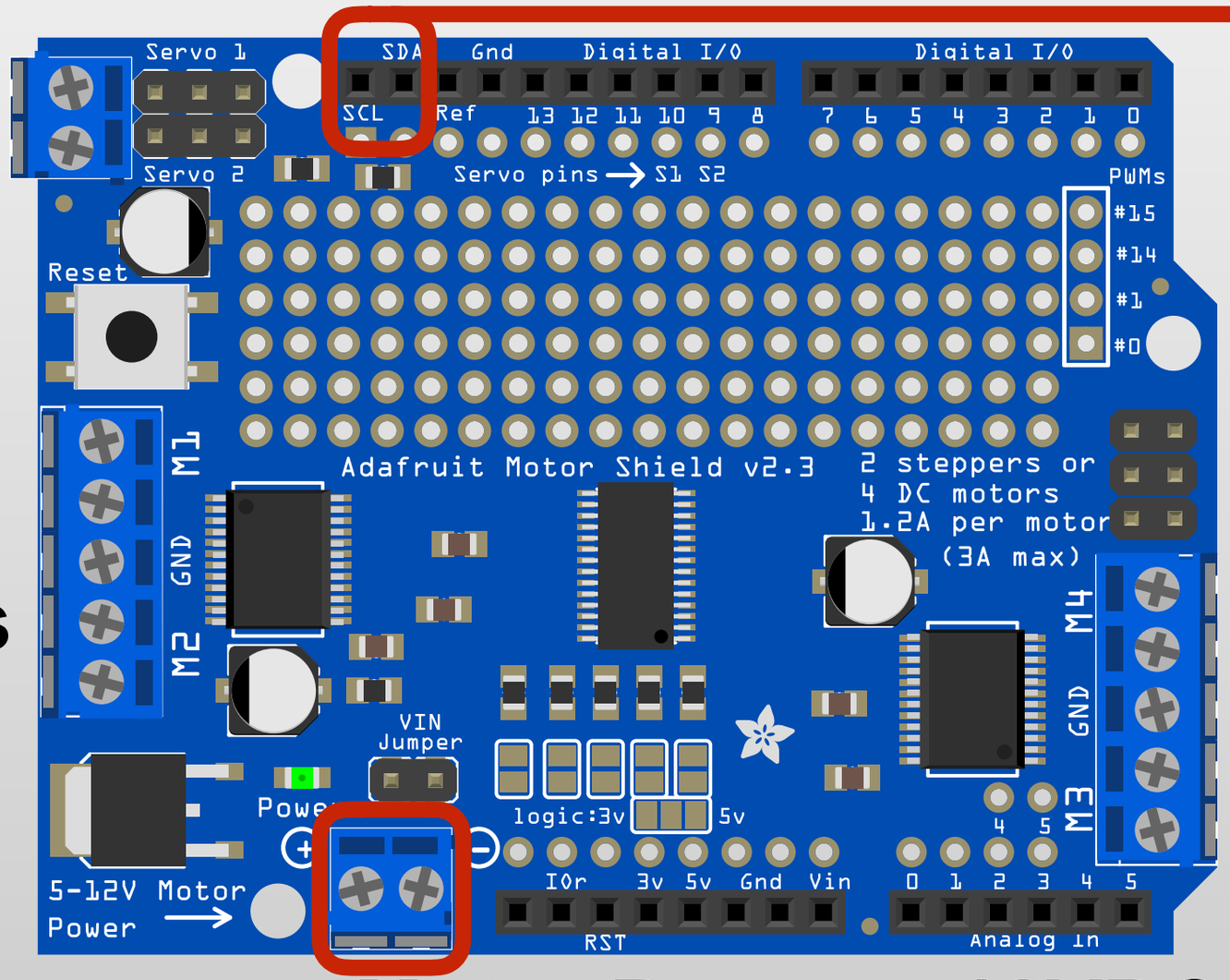
- Connect:
  - the IR sensor
  - a pushbutton
- Wait for the pushbutton to be pressed
- Once it is pressed:
  - Read the IR sensor every 100ms for 30 seconds
  - Print its value to the REPL
  - If objects are closer than 6 inches, turn on the RED LED
  - Otherwise, turn on the Green LED
- After 30 seconds, turn off all the LEDs

Adafruit Motor Shield

- Intended for use with Arduino boards
- Microcontroller that handles low-level motor control
- Communicate with it over i²c

i²c Pins

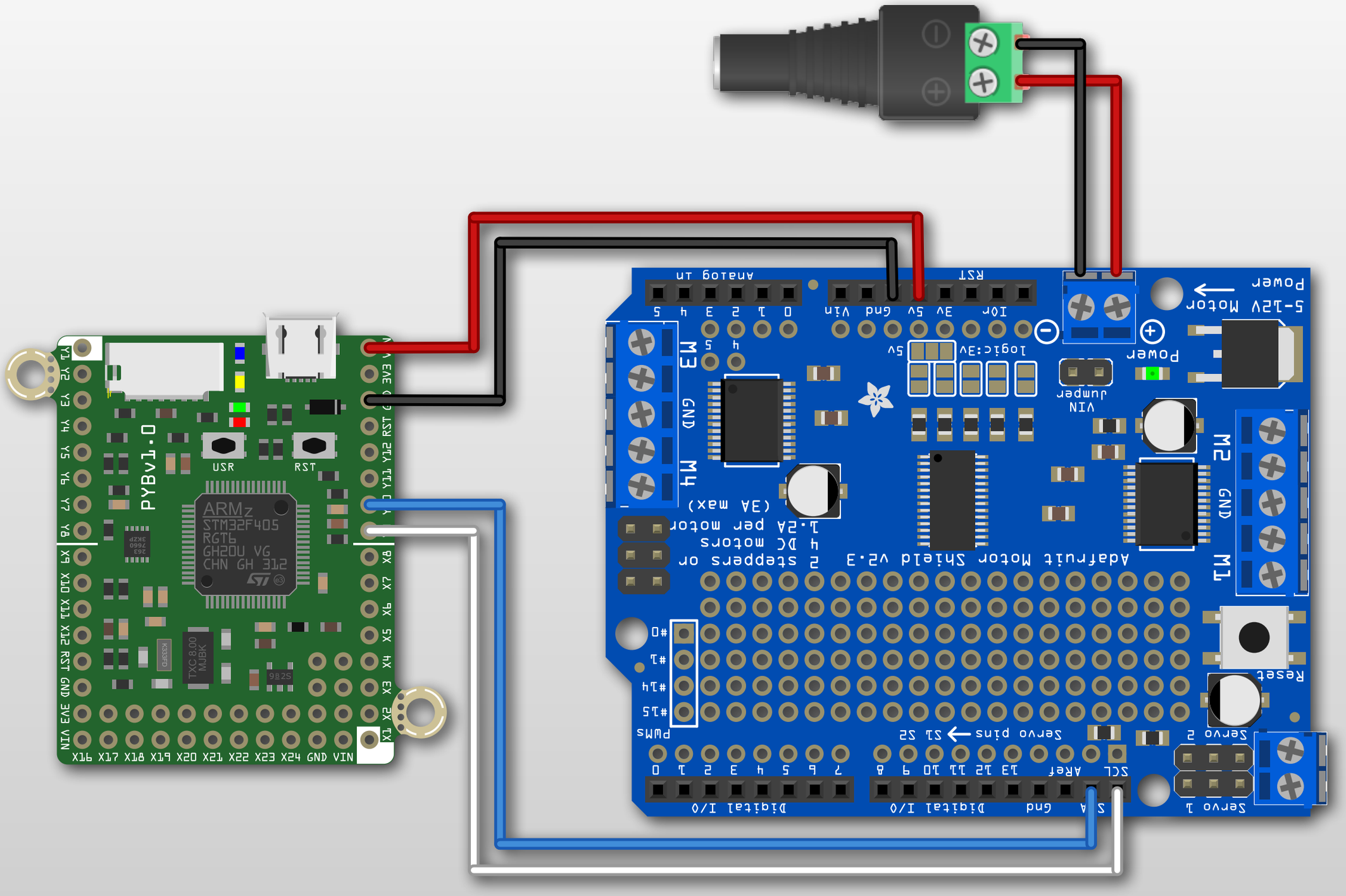
Motor outputs



Motor outputs

Motor Power – 12VDC

Connection to the pyboard



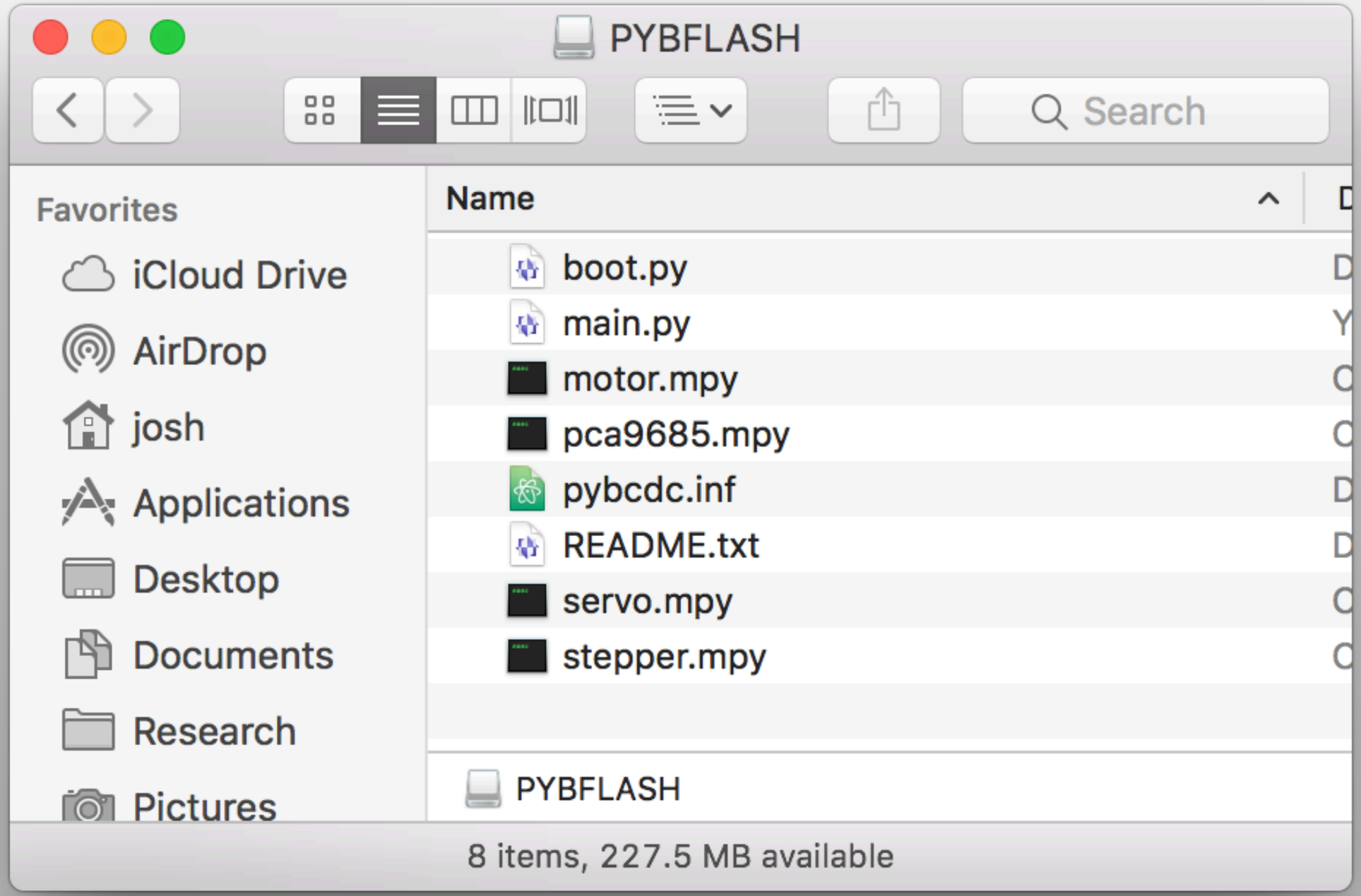
Installation of Adafruit Libraries

- Go to: `https://github.com/adafruit/micropython-adafruit-pca9685/releases`

- Download all the `.mpy` files from the most recent release there (It's version 1.1.1 as of 10/26/17.)

- Copy them to the pyboard PYBFLASH (or micro-SD card if you are running your code from there)

PYBLASH after install



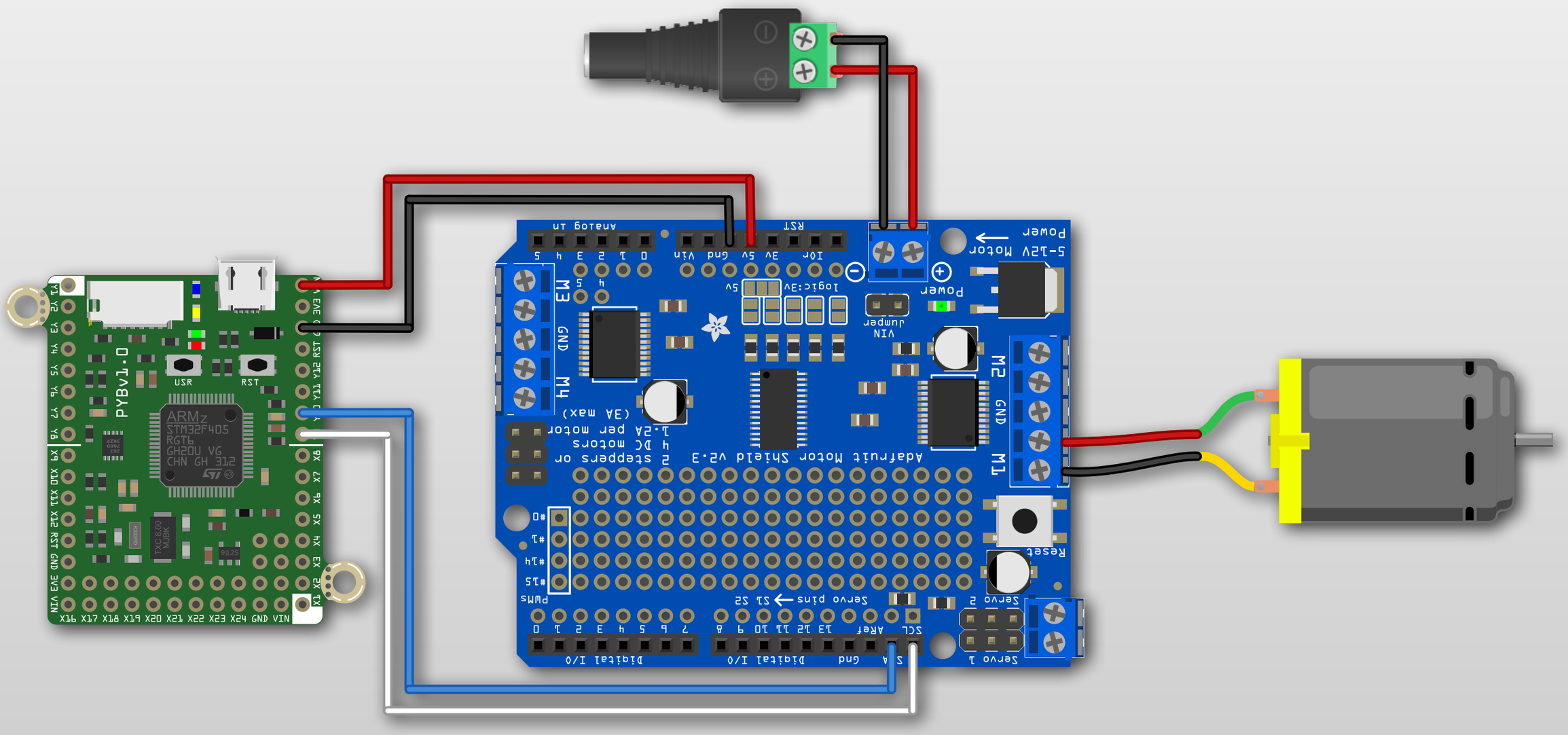
Initialization in MicroPython

```
# Well use the machine module i2c implementation.
import machine

# Initialize communication with the motor driver
i2c = machine.I2C(scl=machine.Pin("Y9"),
                  sda=machine.Pin("Y10"))
```

This should never need to change in your code.

DC Motor Hardware Setup

- Motor can be plugged into M1, M2, M3, or M4
- Do ***NOT*** let conductors on the leads touch



AVOID!!! – You *will* break the board.

- Stripping too much wire from:
  - the motor connections
  - the power connections to the shield

- Keeping stalled motors powered

- Reversing a motor without stopping it first

Entirely-avoidable Carnage

DC Motor Setup and Core Functions

```
# We also need to import the DC motor code from the library
import motor

# And, then initialize the DC motor control object
motors = motor.DCMotors(i2c)

# The number should be
# motor number = (number on the motor driver board - 1)
# For example, M1 on the board is motor 0, M2 on the board is motor
1, etc
MOTOR_NUMBER = 0 # DC motor M1

# To control the motor, give it a speed between -4095 and 4095
motors.speed(MOTOR_NUMBER, 2048)   # Go ~1/2 speed in one direction

# To stop, issue a speed of 0
motors.speed(MOTOR_NUMBER, 0)

# There is also a brake() command
motors.brake(MOTOR_NUMBER)
```

Stepper Motors

- NEMA-17 size
- 200 steps/rev
- 12V 350mA

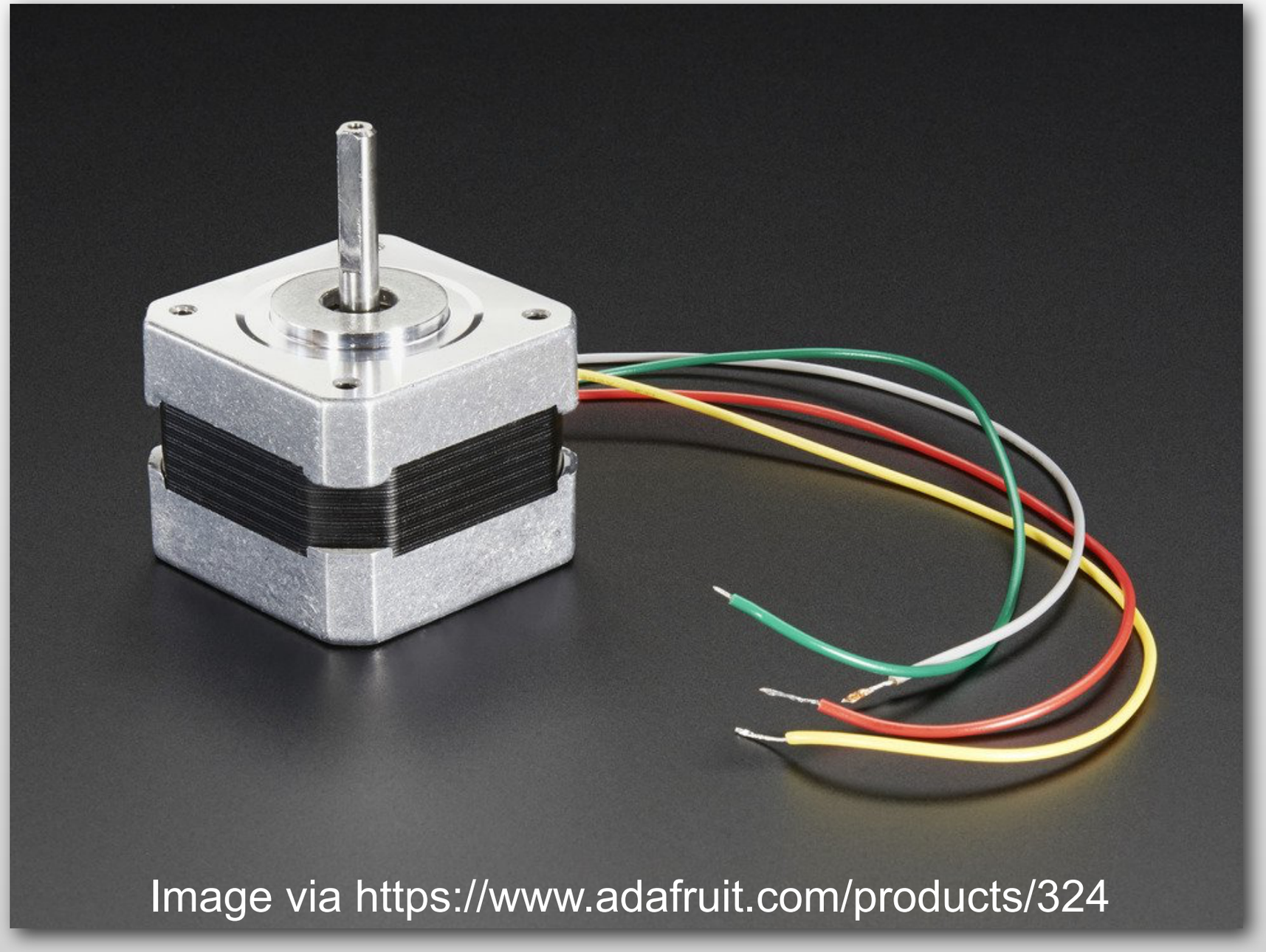


Image via https://www.adafruit.com/products/324

Connecting to the Motor Shield

- Gray & Green to M1 or M3
- If Gray & Green in M1, Gold & Red to M2
- If Gray & Green in M3, Gold & Red to M4

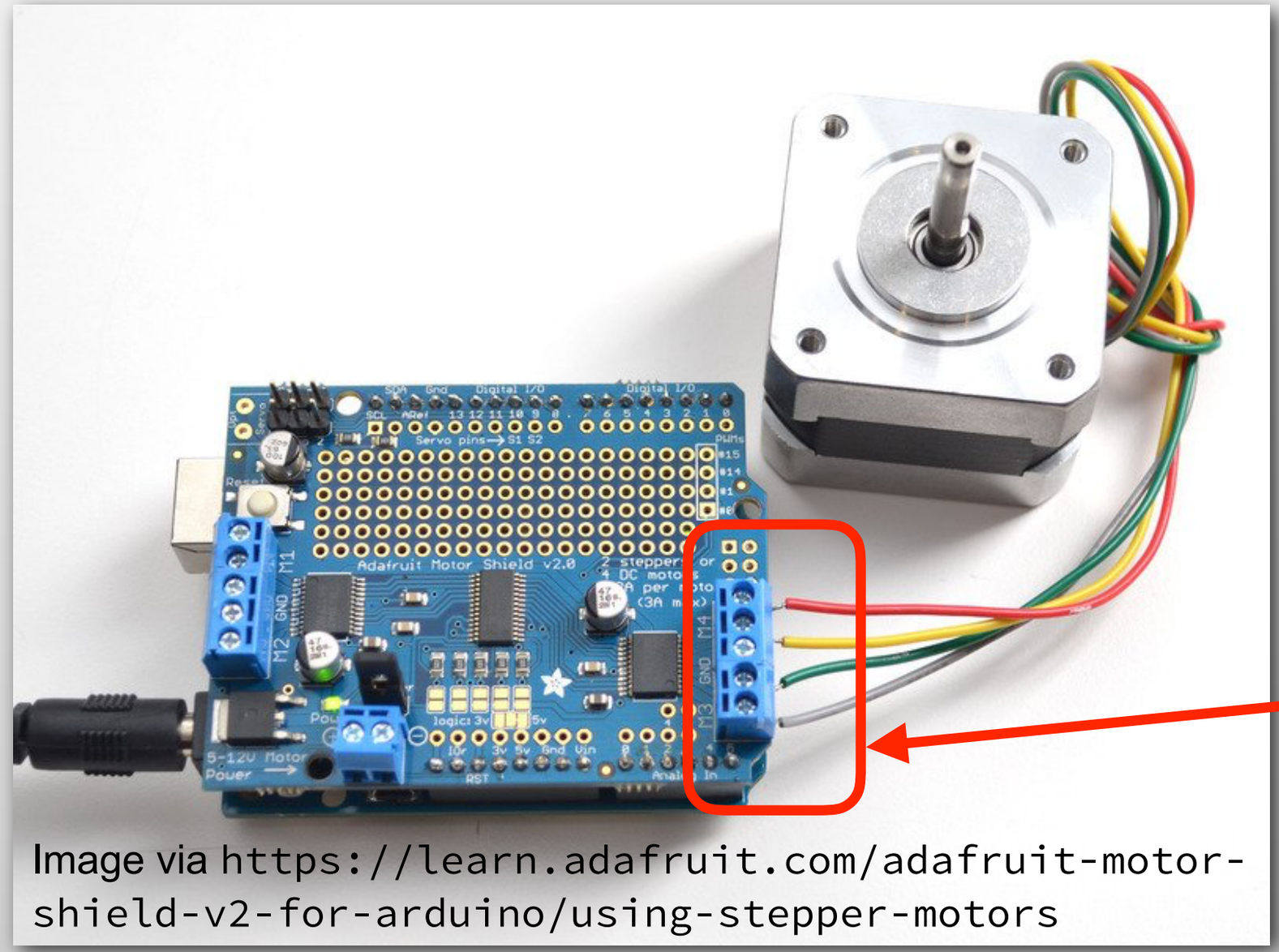
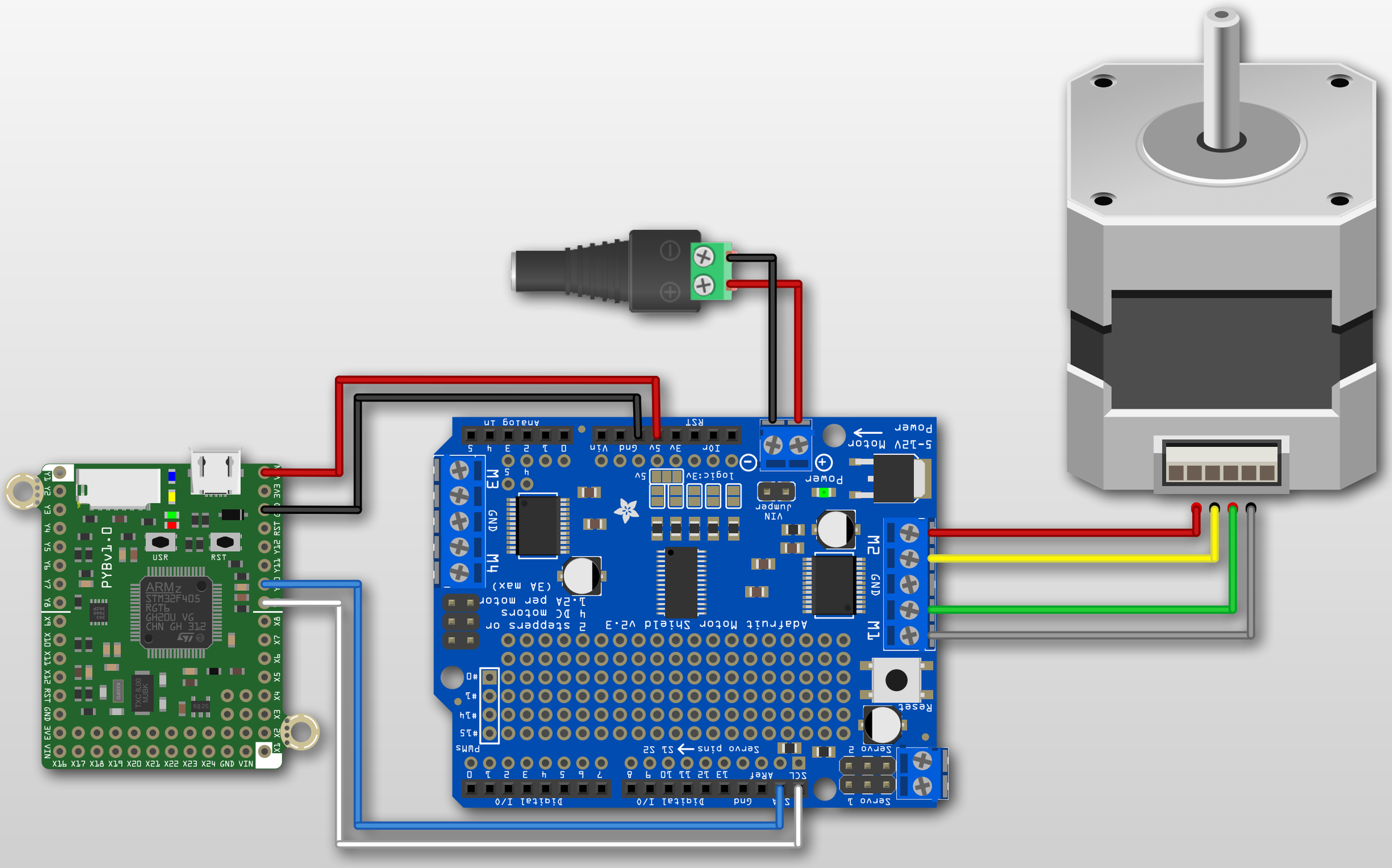


Image via https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/using-stepper-motors

Stepper Motor Hardware Setup



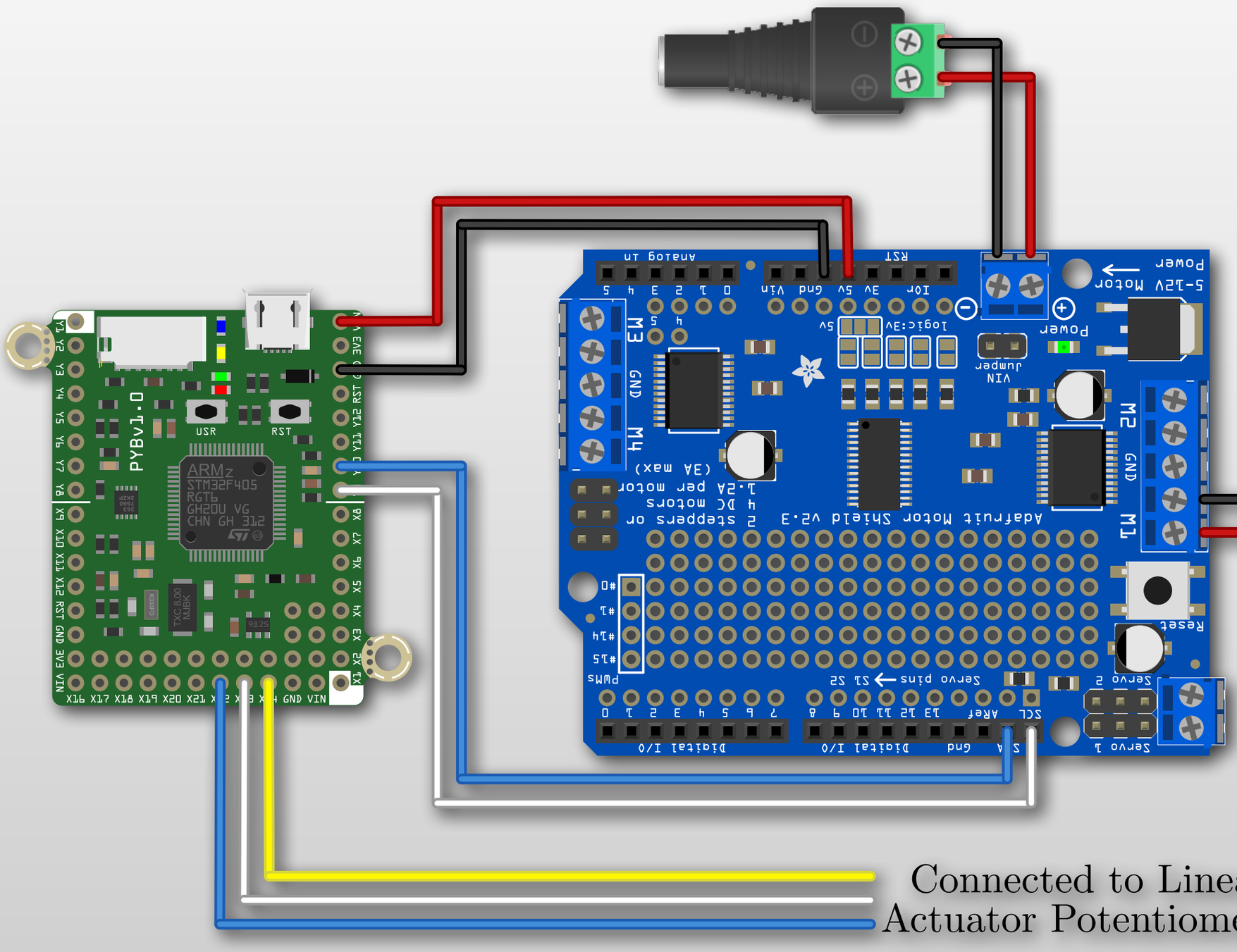
Stepper Motor Initialization

```
# We need to import the stepper motor code from the library
import stepper

# Now, we can initialize the stepper motor object
steppers = stepper.Steppers(i2c)

# Then, create an instance for the connected stepper motor
# Use index 0 if the motor is connected to M1 & M2
# Use index 1 if the motor is connected to M3 & M4
STEPPER_MOTOR_NUM = 0

stepper0 = steppers.get_stepper(STEPPER_MOTOR_NUM)
```

Stepper Motor Core Functions

```
# Now, we can control the motor. To make it move one step in
# SINGLE step mode. Note that the onestep() function is blocking.
# Nothing else will run while the step is being performed
stepper0.onestep(stepper.FORWARD, stepper.SINGLE)

# We can also move in DOUBLE step mode. This time in reverse
stepper0.onestep(stepper.BACKWARD, stepper.DOUBLE)

# We can also move in MICROSTEP step mode.
# It will move 1/16 of a step each time.
stepper0.onestep(stepper.FORWARD, stepper.MICROSTEP)

# To make the motor move more than one step, we need to
# repeatedly call the one-step function. The motors in the
# MCHE201 kit have 200 step/rev so the for loop below should
# cause the motor to turn one full revolution
for index in range(200):
    stepper0.onestep(stepper.FORWARD, stepper.SINGLE)
```

Linear Actuator Hardware Setup



Linear Actuator Coding

- It is a DC motor, setup and control just like one

- The feedback is just a potentiometer whose value is proportional to length of actuator.