MicroPython Introduction (cont.)

MCHE 201 – Spring 2019

Dr. Joshua Vaughan

Rougeou 225

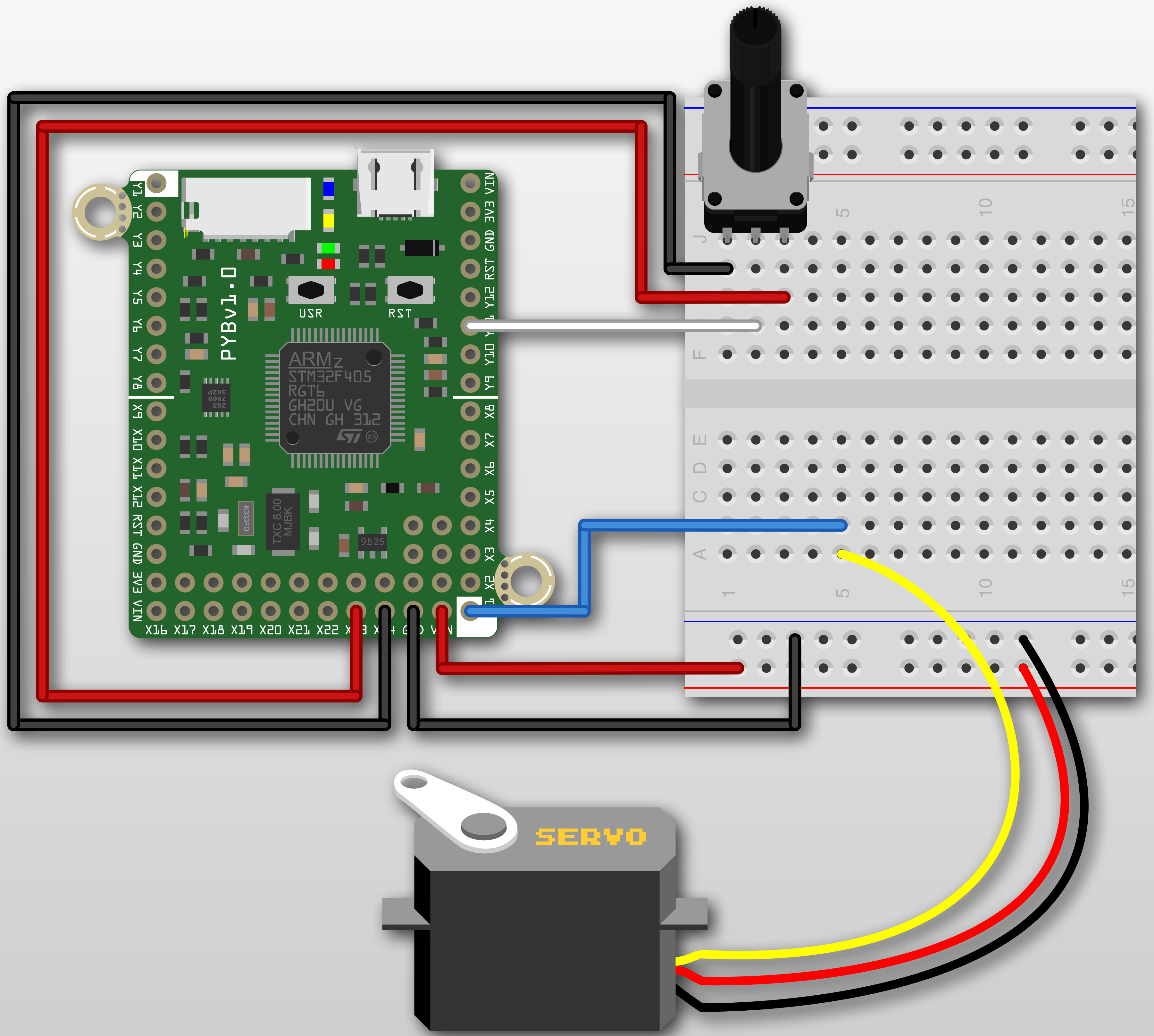
joshua.vaughan@louisiana.edu

@Doc_Vaughan

In-class Exercise 7

- Attach a potentiometer
- Have the servo angle track the angle of the potentiometer

In-class Exercise 7 Hardware



In-class Exercise 7 Setup

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

# Here, we will use the X1 position on the pyboard
servo1 = pyb.Servo(1)

# Define constants for the min and max servo angles
MAX_SERVO_ANGLE = 45
MIN_SERVO_ANGLE = -45

# Set up the ADC for the potentiometer
pot_adc = pyb.ADC(pyb.Pin("Y11"))
```

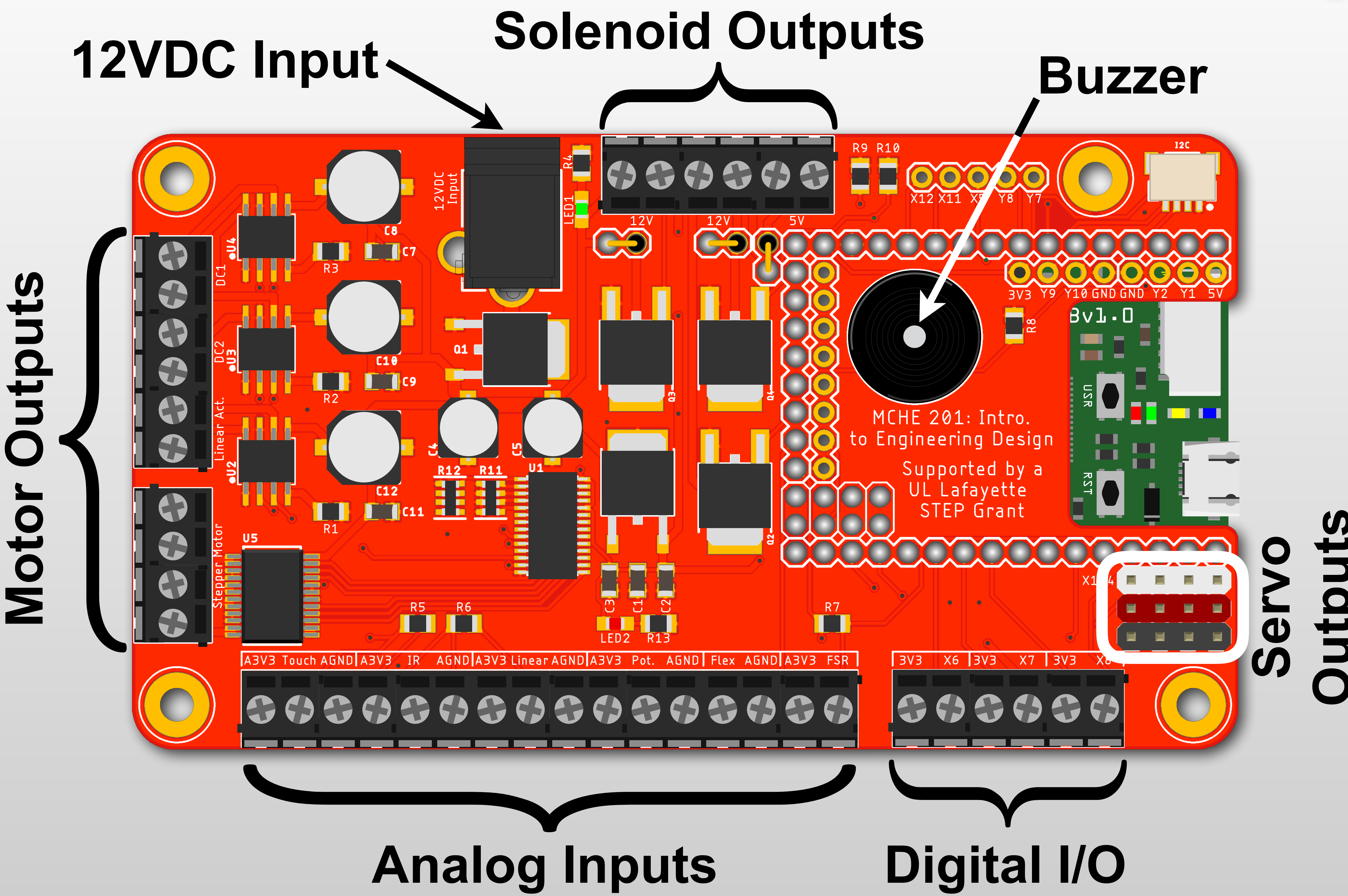
In-class Exercise 7 Angle Conversion

```python
def potADCtoServoAngle(ADC_value):
    """ This function converts a potentiometer reading of 0-4095 to an angle
    between MIN_SERVO_ANGLE and MAX_SERVO_ANGLE, using the global
    representation for those angle extremes

    The middle of the potentiometer range, 2048, should map to 0deg
    The max. of the range, 4095, should map to MAX_SERVO_ANGLE
    The min. of the range, 0, should map to MIN_SERVO_ANGLE

    Inputs:
      ADC_value : a number between 0 and 4095 representing a reading
                  from the potentiometer

    Returns:
      angle : The angle to move the servo to to match the potentiometer angle
    """

    # define the slope and intercept for the line mapping ADC_value to angle
    slope = (MAX_SERVO_ANGLE - MIN_SERVO_ANGLE) / 4095
    intercept = -slope * 2048

    # Now, calculate the angle output based on that linear function
    angle = slope * ADC_value + intercept

    return angle
```

In-class Exercise 7 Main Loop

```python
# Now read the pot and move the servo every 10ms, forever
while (True):
    # Read the value of the potentiometer.
    # It should be in the range 0-4095
    pot_value = pot_adc.read()

    desired_angle = potADCtoServoAngle(pot_value)

    # print out the values, nicely formatted
    print("The ADC value is {:d}.".format(pot_value))
    print("Moving to {:.2f} deg".format(desired_angle))

    servo1.angle(desired_angle)

    # Wait 10ms before looping again
    time.sleep_ms(10)
```

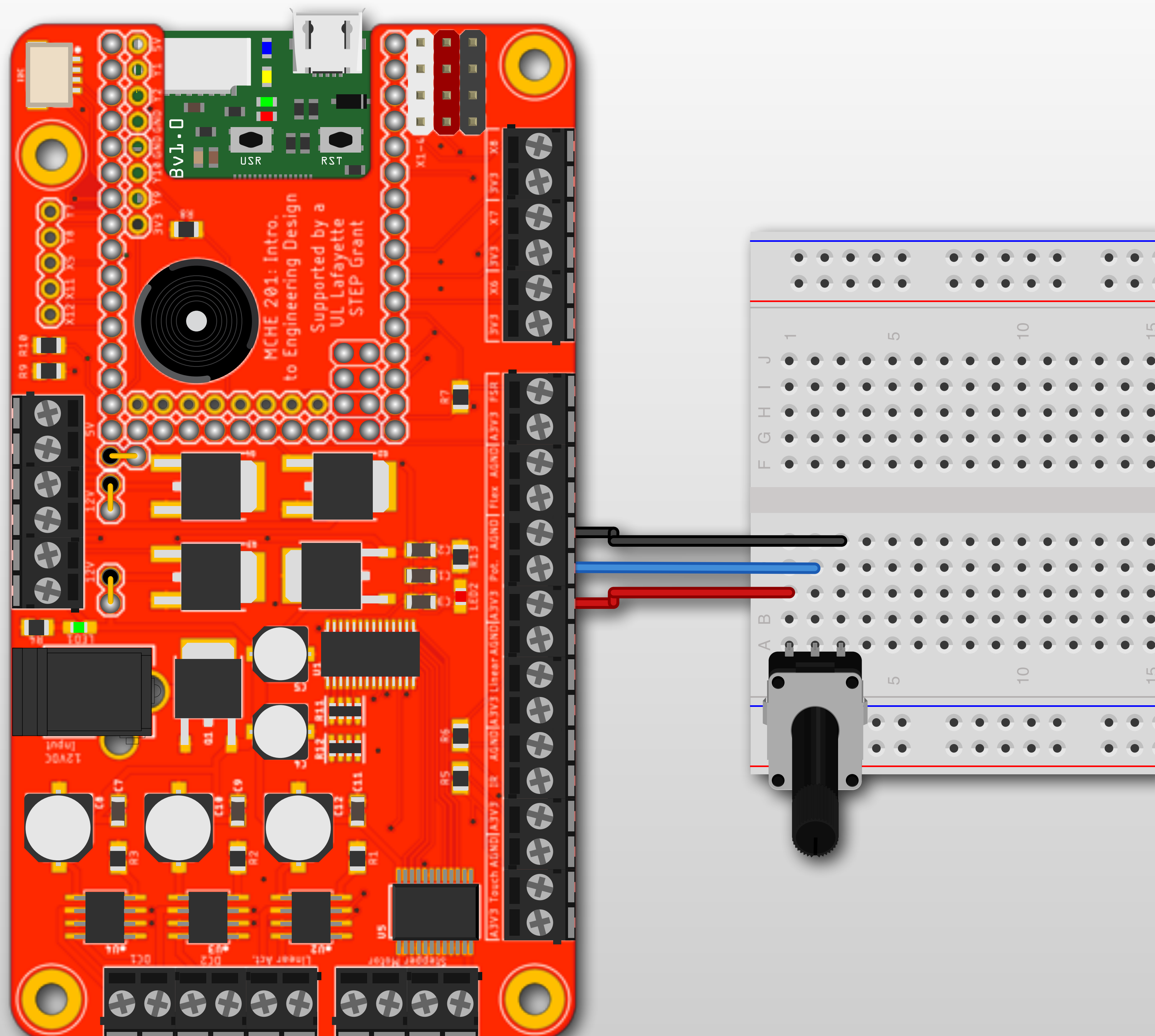
The MCHE201 Board



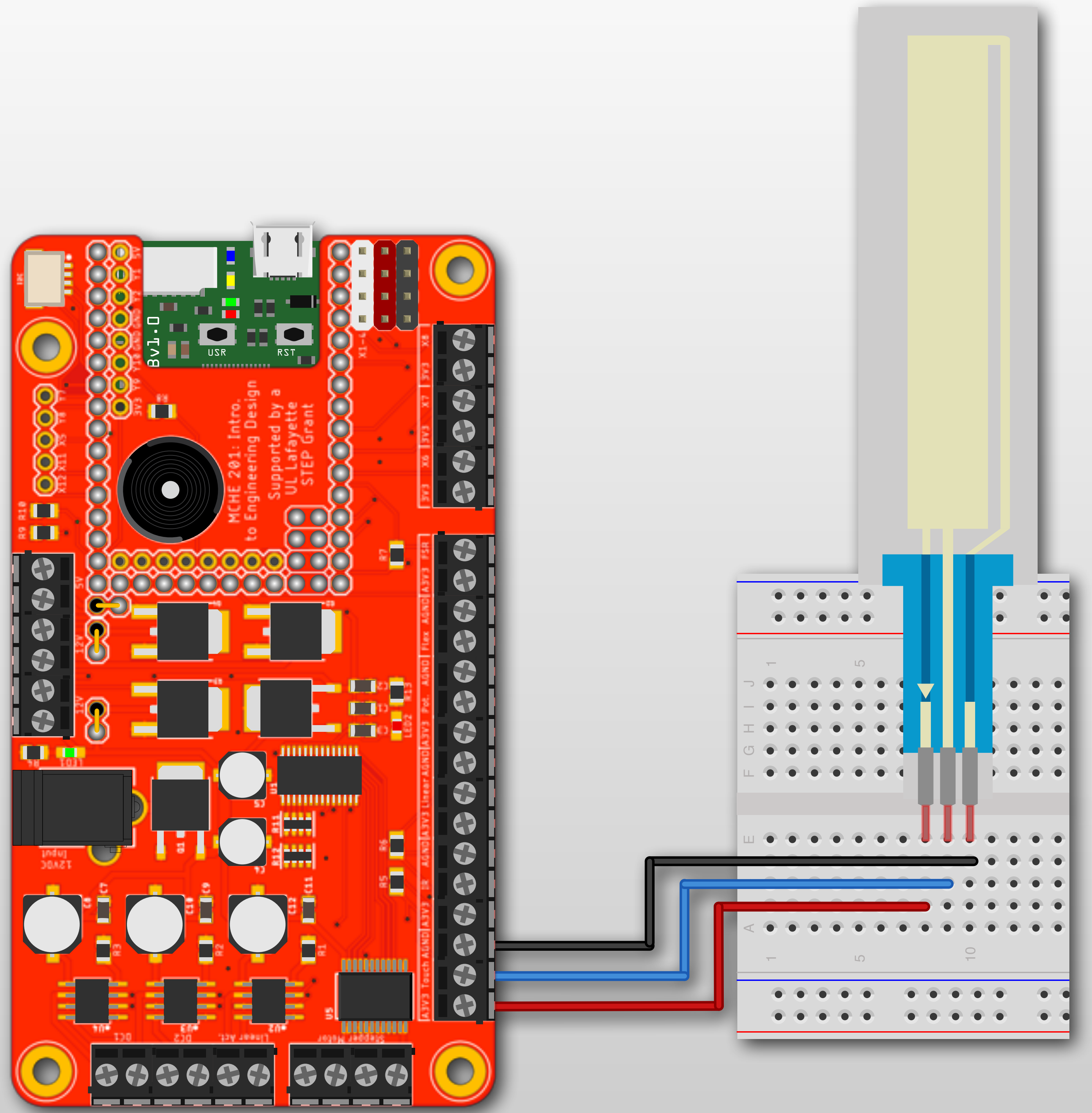
MCHE201 Board Analog Inputs

- Pin assignments match code you've learned already
- All resistors are included on-board… Just connect the sensor itself
- Wiring diagrams are included in GitHub repository for each sensor

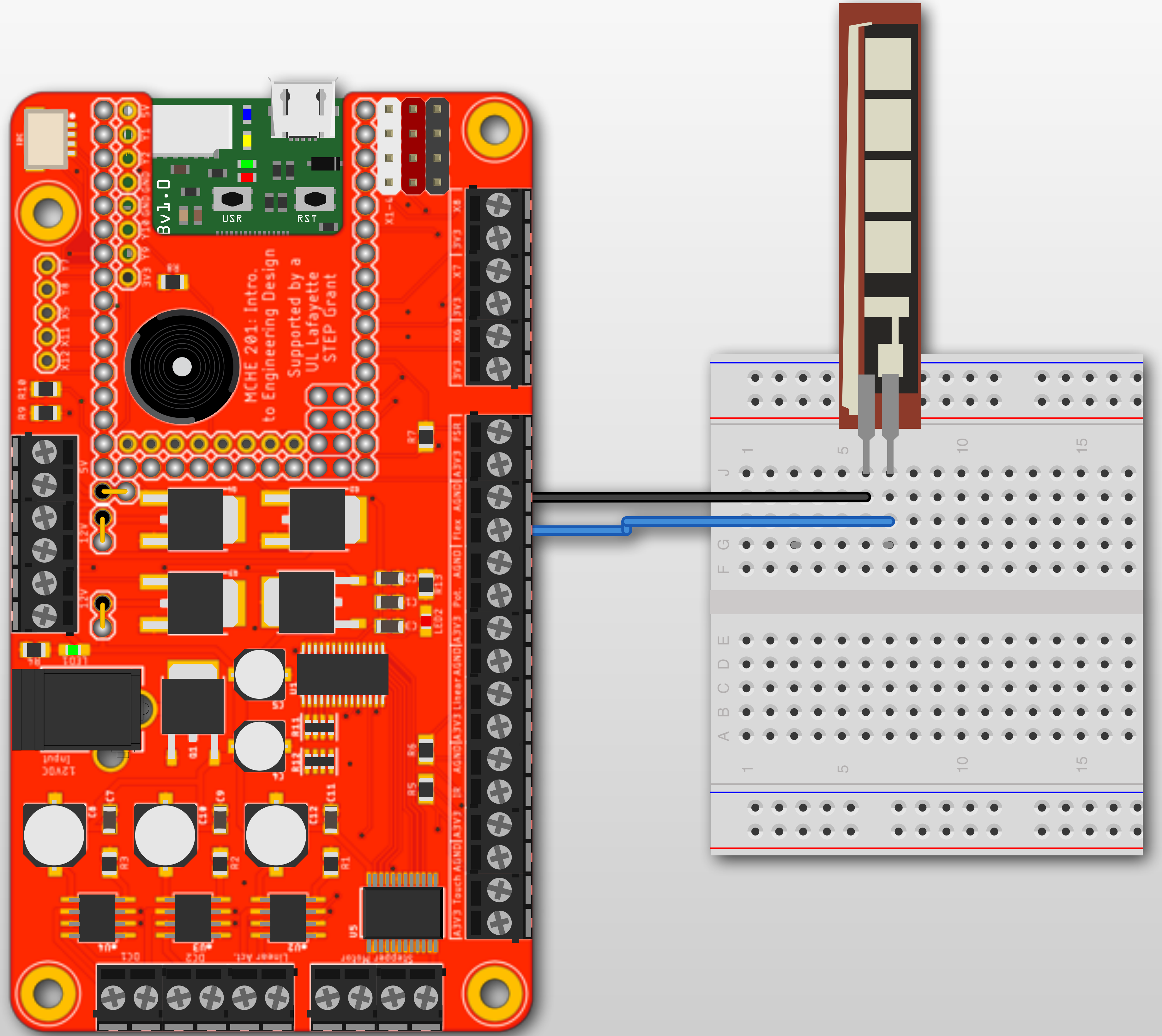
MCHE201 Board – Potentiometer



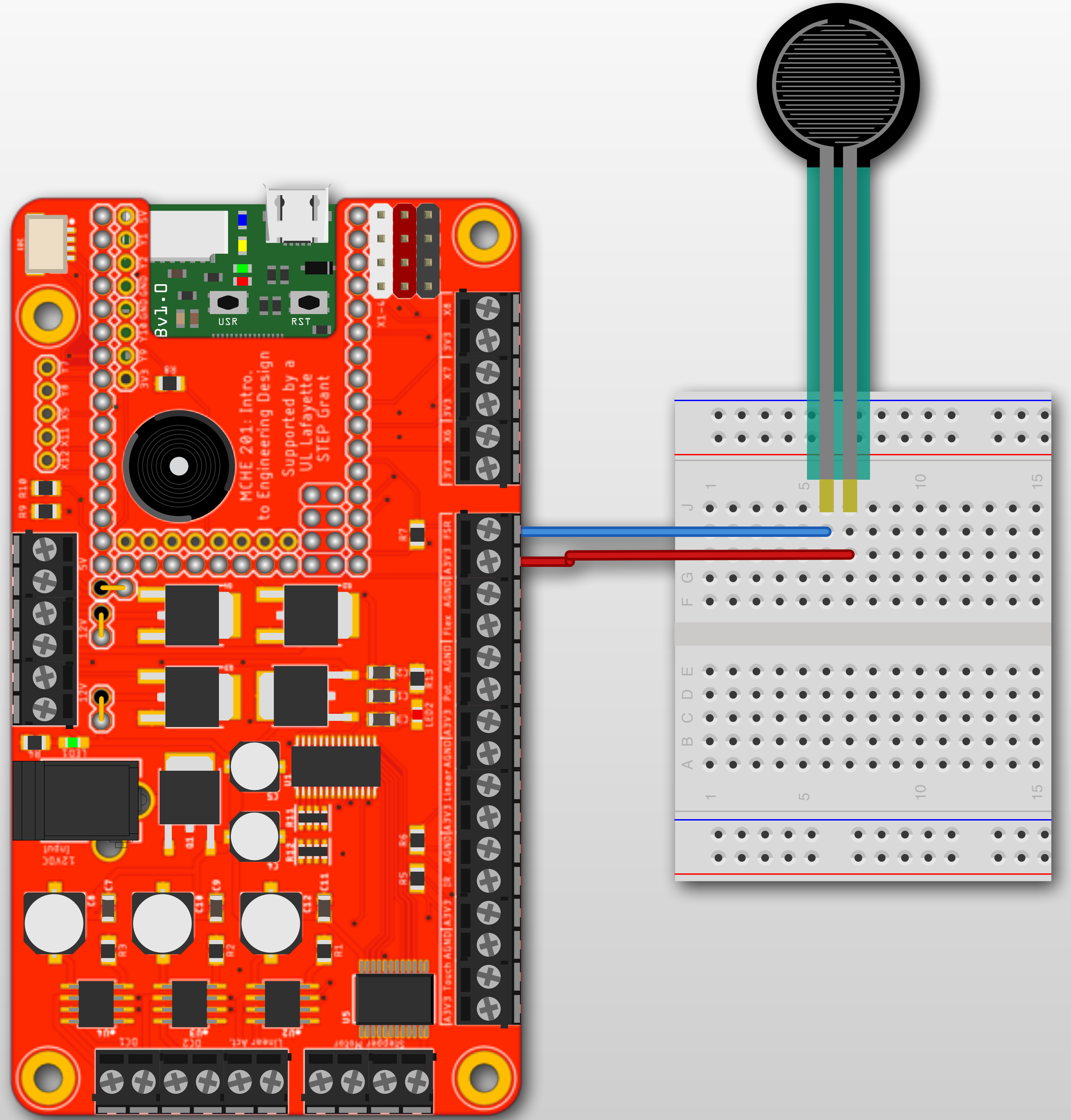
MCHE201 Board – Soft Pot.



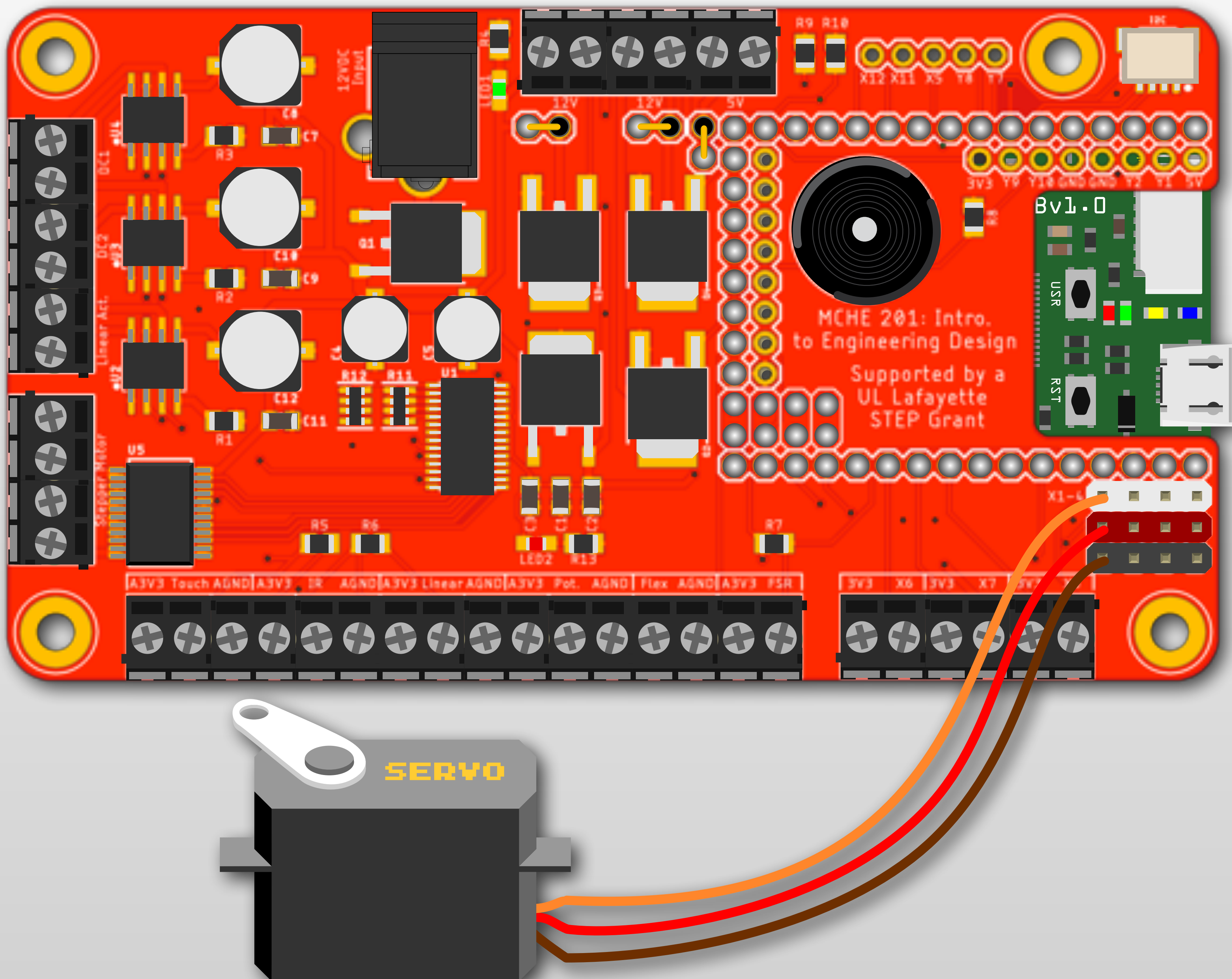
MCHE201 Board – Flex Sensor



MCHE201 Board – FSR

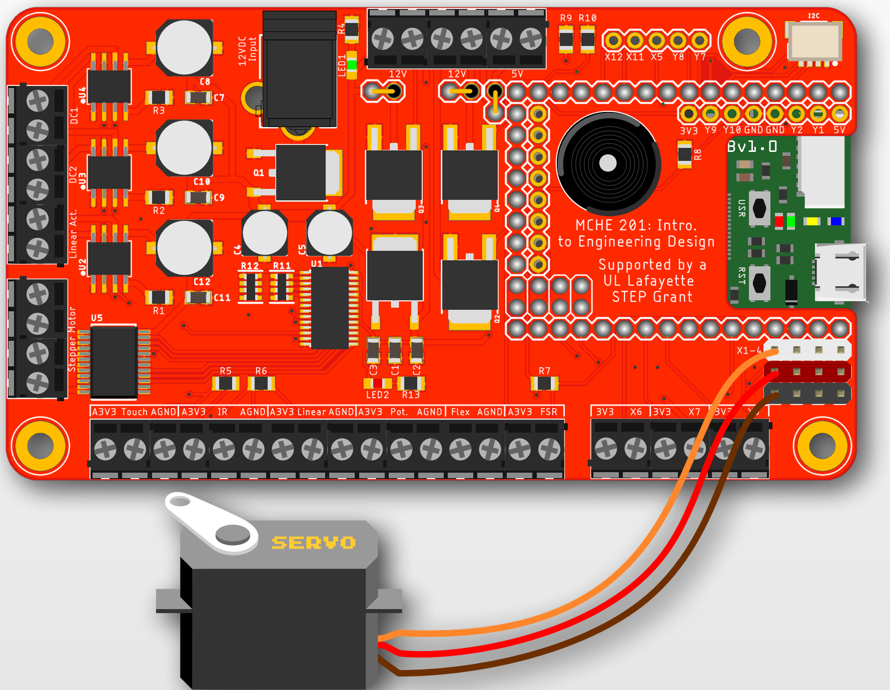


MCHE201 Board – Servomotors

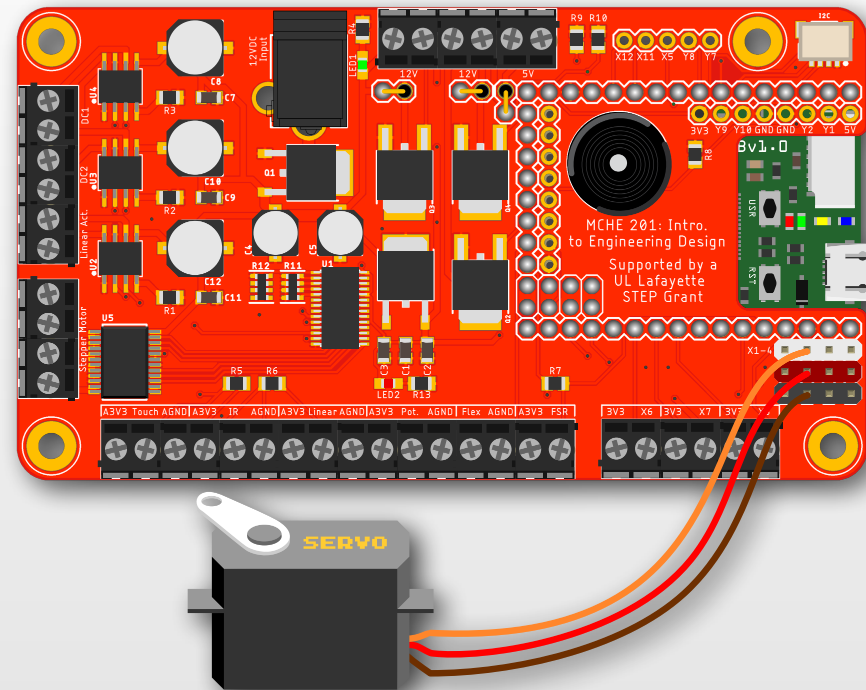


Wired as Servo 1

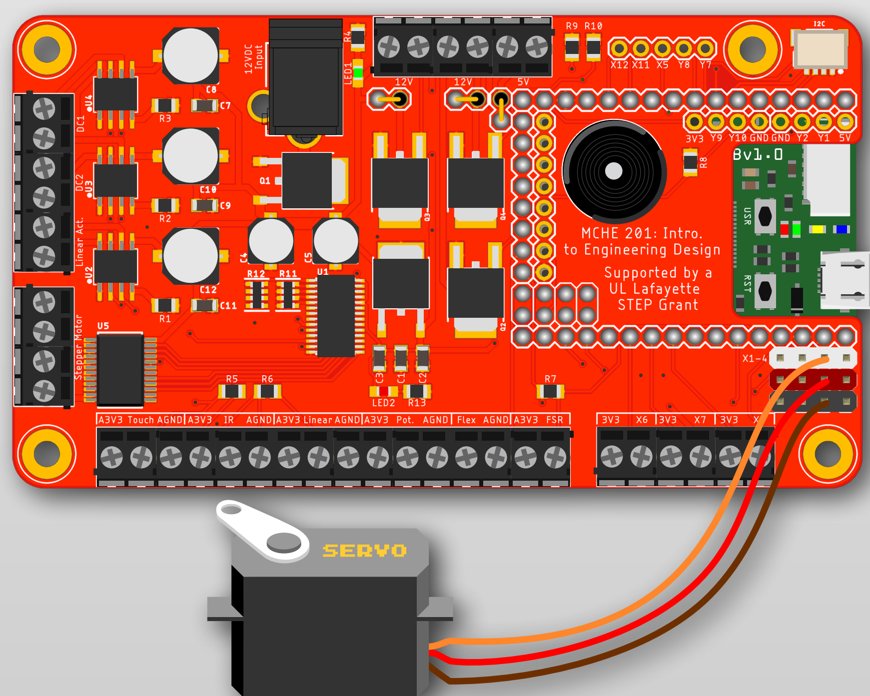
MCHE201 Board – Servomotors



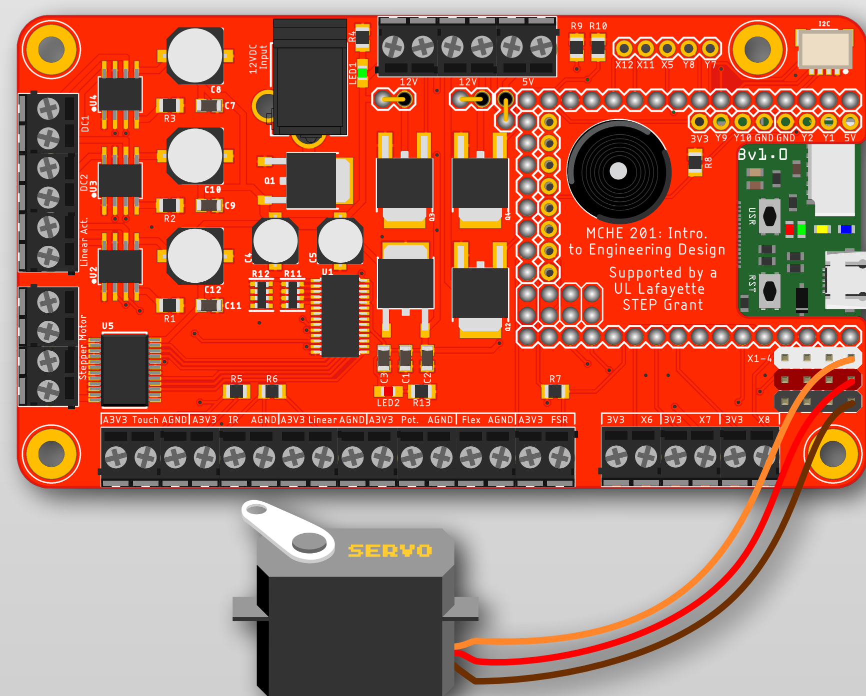
Wired as Servo 1



Wired as Servo 2



Wired as Servo 3



Wired as Servo 4

Controlling Timing

```
# Import time module
import time

# sleep for 1 second
time.sleep(1)

# sleep for 500 milliseconds
time.sleep_ms(500)

# sleep for 10 microseconds
time.sleep_us(10)
```

Controlling Timing

```
# Import time module
import time

# sleep for 1 second
time.sleep(1)

# sleep for 500 milliseconds
time.sleep_ms(500)

# sleep for 10 microseconds
time.sleep_us(10)
```

The time.sleep family of functions *sleep* the processor.

Time Comparison

- Get the current time (to the ms or μs) using `time.ticks_ms()` or `time.ticks_us()`
- Do time math using `time.ticks_add()` and `time.ticks_diff()`
  - `time.ticks_add(ticks, delta)` calculates `ticks + delta # Units must match`
  - `time.ticks_diff(ticks1, ticks2)` calculates `ticks1 - ticks2`
- More info at: `http://docs.micropython.org/en/latest/pyboard/library/utime.html`

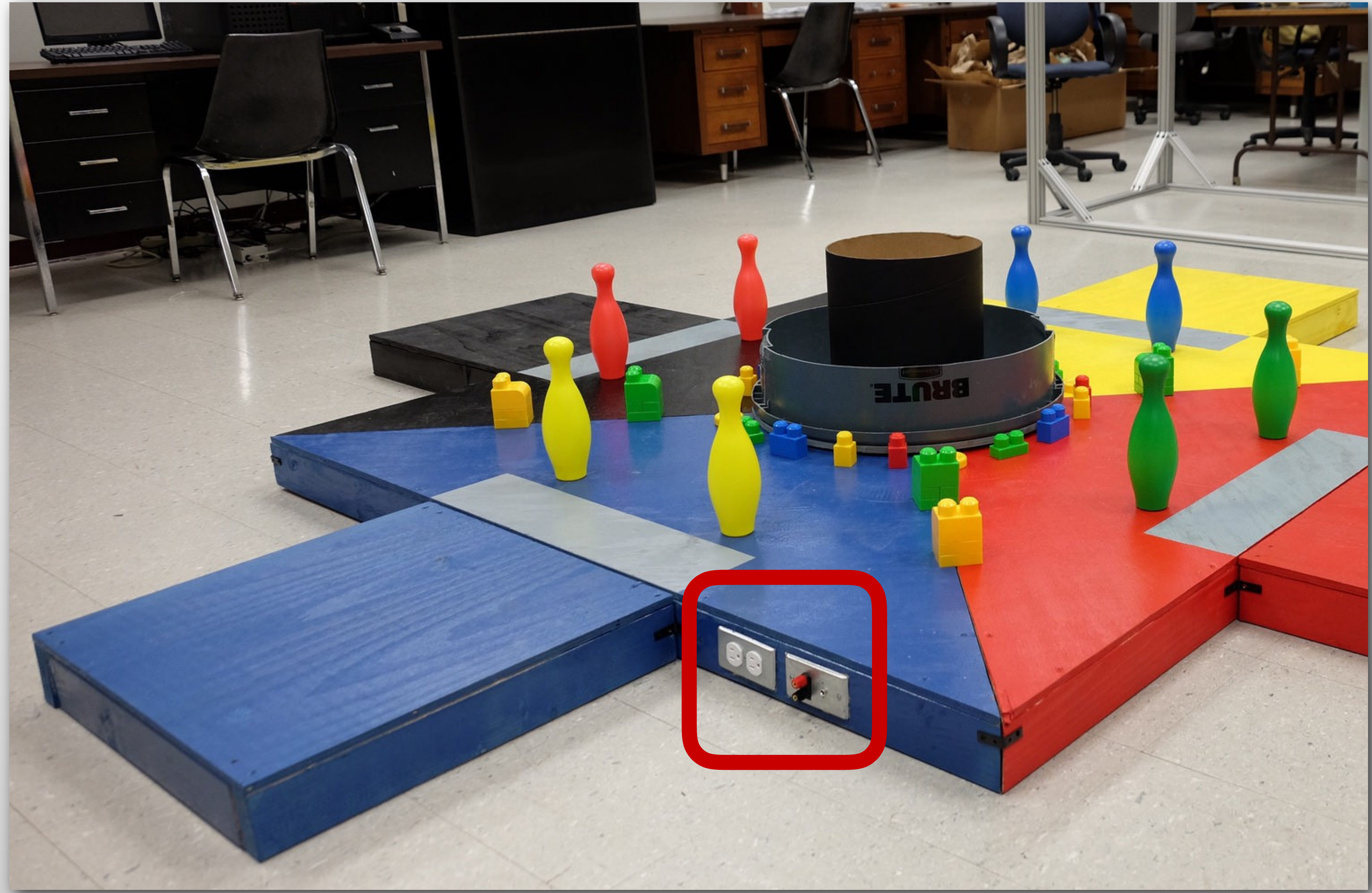
In-class Exercise 8

- Connect a pushbutton
- Turn on the green LED
- When the pushbutton is pressed
  - Turn on the red LED
  - Turn off the green LED
- When the button is pressed again
  - Turn off the red LED
  - Turn on the green LED
  - Print the time elapsed between button presses to the REPL

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 presses to the REPL
- If more than 5s elapses:
  - Print "You took too long!!!" to the REPL
  - Turn on only the green LED again

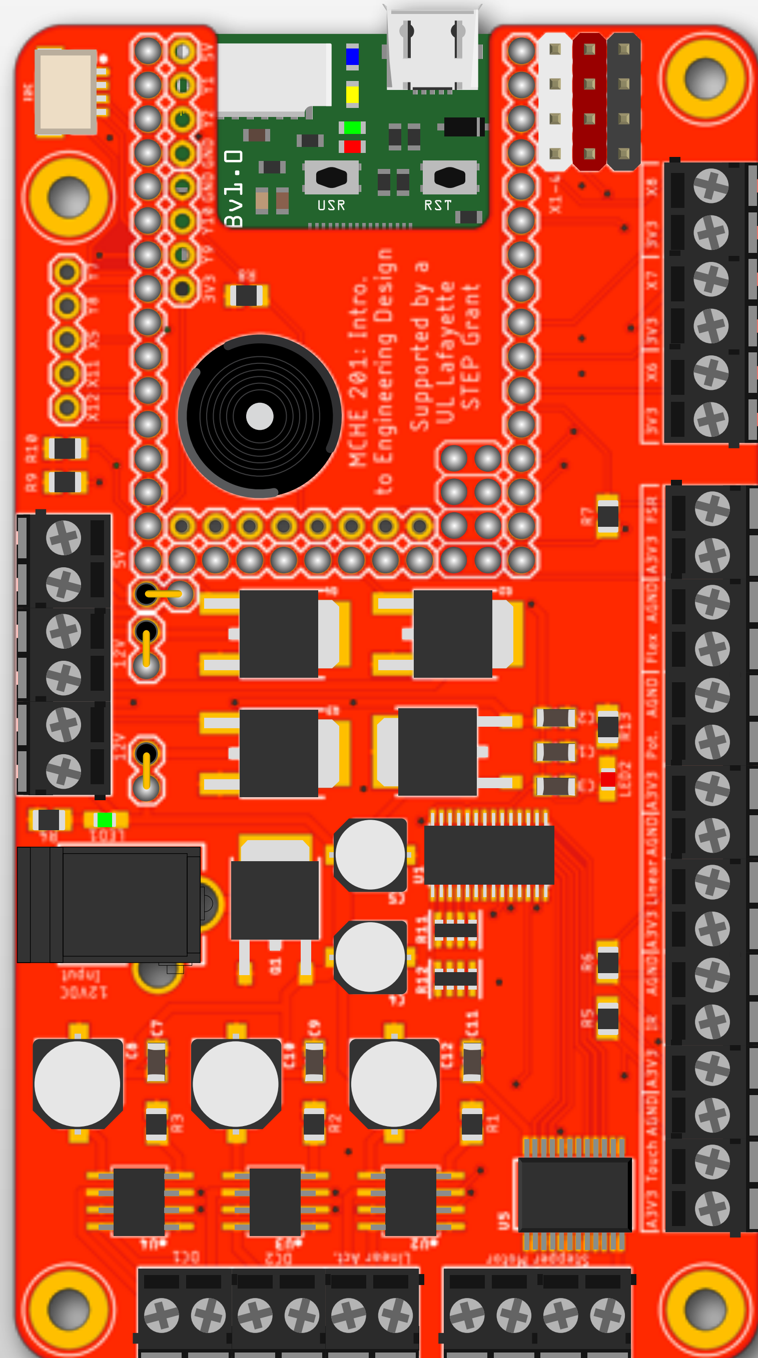
MCHE201 Track Connections

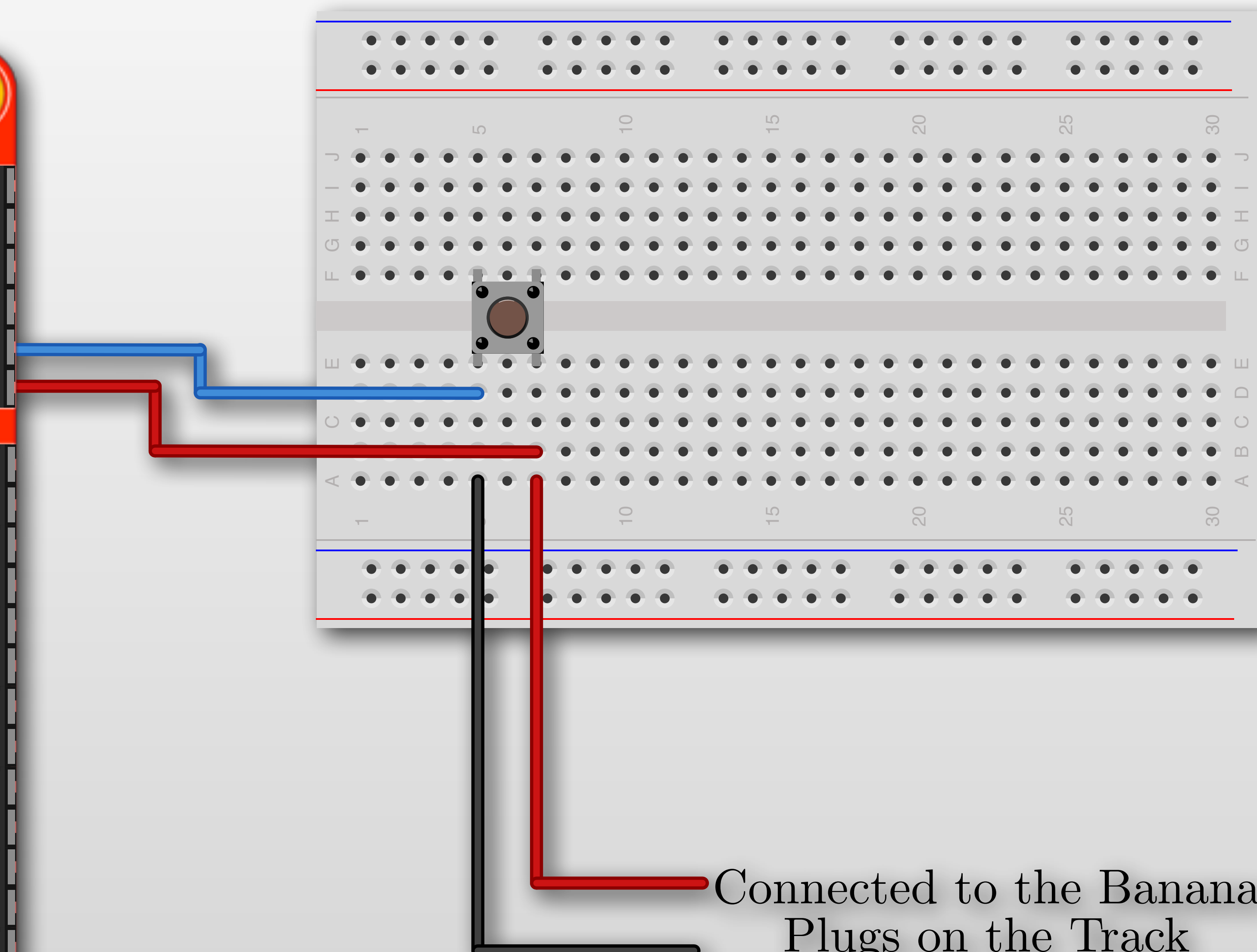


MCHE201 Track Start Signal

- Will be closed for the 30-second trial time, open otherwise
- Works just like holding down a pushbutton for 30 seconds.
- The 120VAC outlet is *always* on

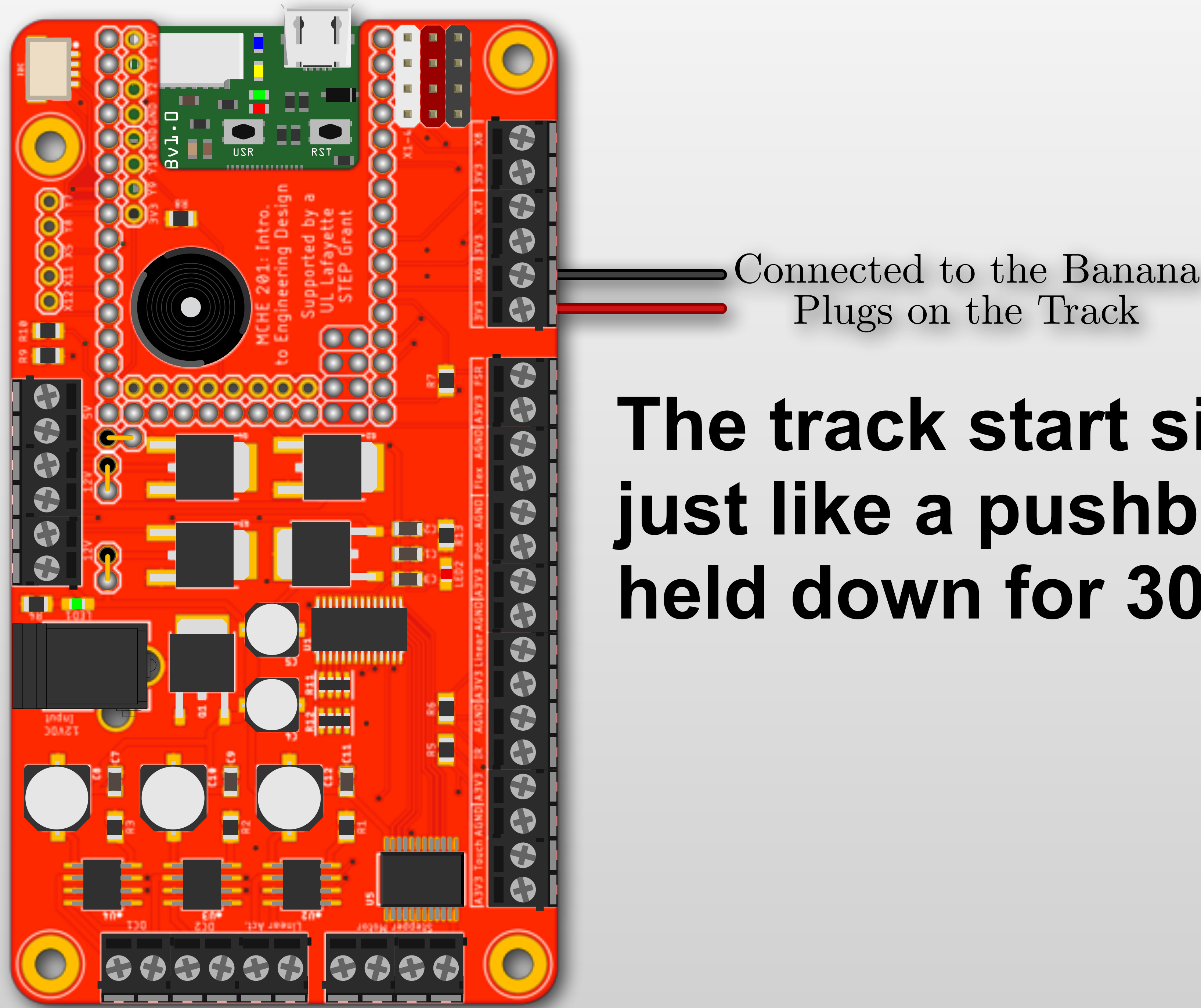
Reading the MCHE201 Start Signal





The track start signal behaves just like a pushbutton being held down for 30 seconds.

Reading the MCHE201 Start Signal



The track start signal behaves just like a pushbutton being held down for 30 seconds.

One way to Sense Start

```
# Assign the input pin to variable input_pin
# We set it up as an input with a pulldown resistor
input_pin = pyb.Pin("X6", pyb.Pin.IN, pull=pyb.Pin.PULL_DOWN)

# This will loop forever, checking the button every 10ms
while (True):
    input_state = input_pin.value()   # read the state of the input

    if (input_state):
        print("The start button is pressed.)
        # Main code could be here

        # If what runs here is less than 30 sec. long, you'll need to
        # account for that condition. If not, then the start signal
        # will still be on when this part of your code finishes. So, it
        # will still be True and therefore start running again.

    else:
        print("The start button is not pressed.")

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

A More "Professional" Way

- Use Interrupts:
  - "Run this function immediately when X happens"
  - Functions need to:
    - be short/fast, and
    - create no new objects

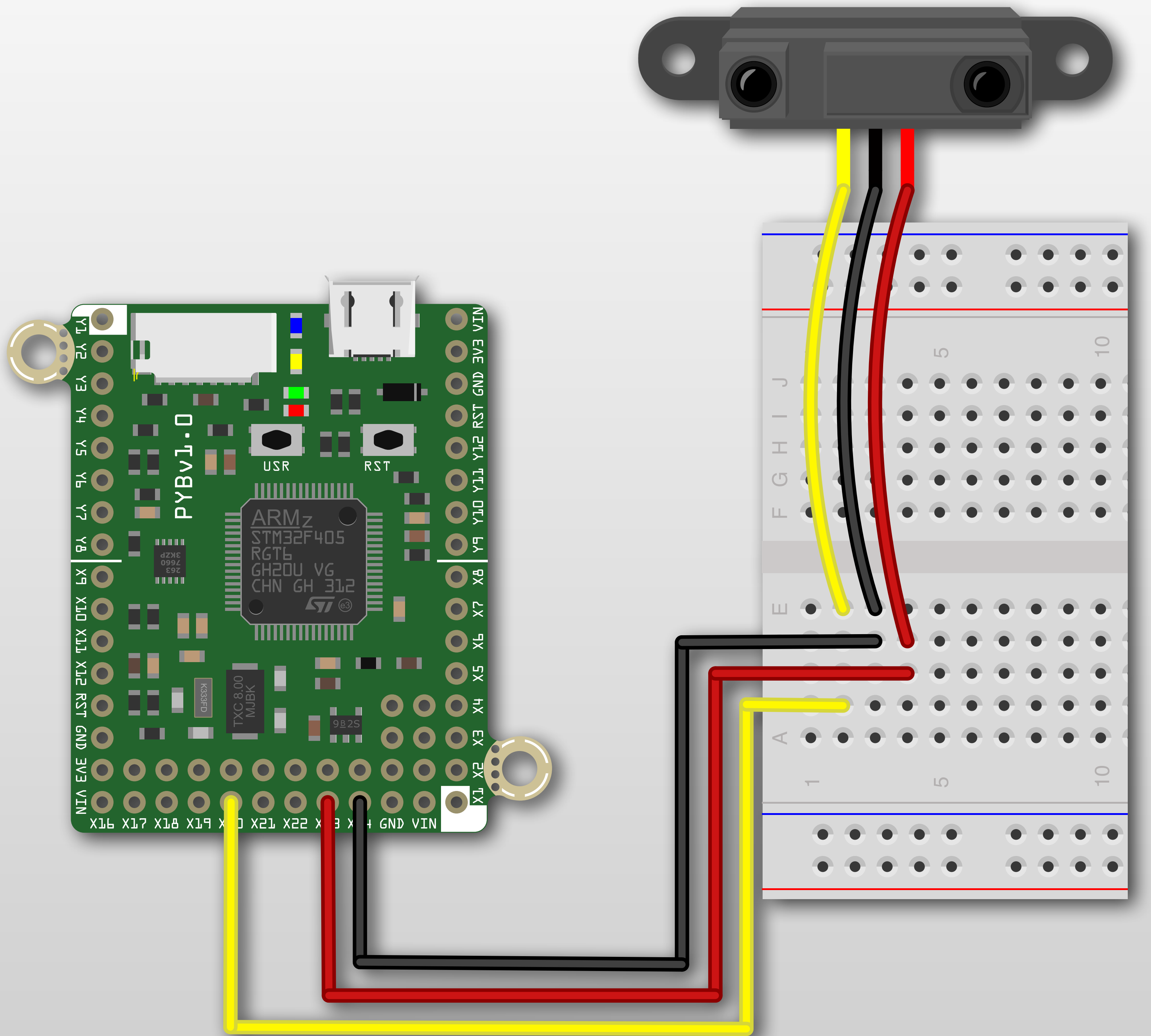
- https://github.com/DocVaughan/MCHE201---Intro-to-Eng-Design/tree/Spring-2019/MicroPython/pyboard%20start%20button%20interrupt

- More info:
  - https://micropython.org/resources/docs/en/latest/library/machine.Pin.html#machine.Pin.irq
  - https://micropython.org/resources/docs/en/latest/reference/isr_rules.html

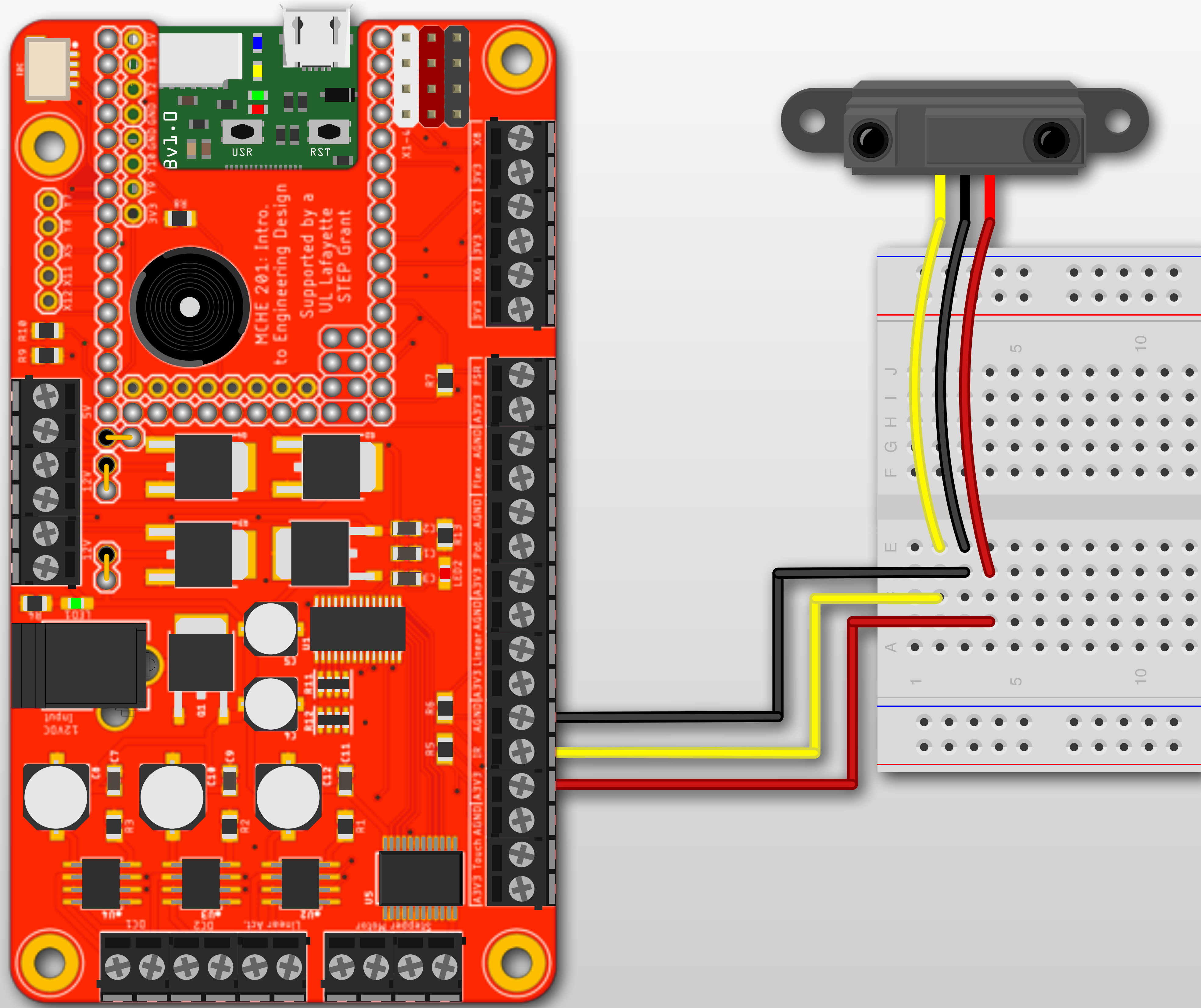
In-class Exercise 10

- Connect
  - a pushbutton
  - the servomotor
- Start the servo at 0 degrees
- When the pushbutton is pressed:
  - move the servo to 30 degrees
  - pause 1 second
  - move the servo back to 0 degrees
- Only allow this to happen once per 30 seconds

IR Sensor Hardware Setup



MCHE201 Board – IR Sensor



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

What will happen?

```python
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…

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
```

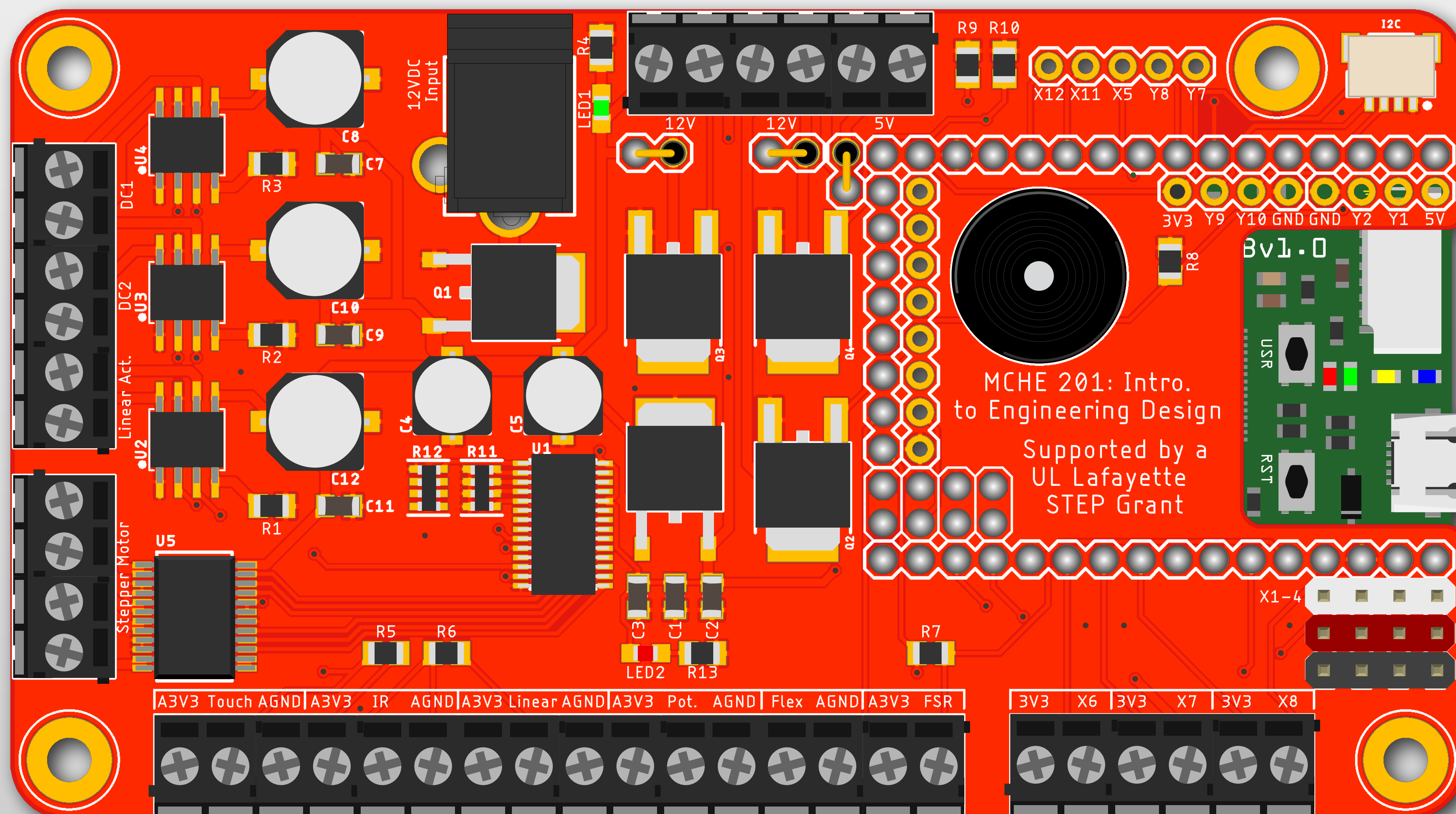
KEY POINT!!!

- 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

MCHE201 Board – Motor Control

- Separate microcontroller handles low-level motor control
- pyboard and it communicate over i²c



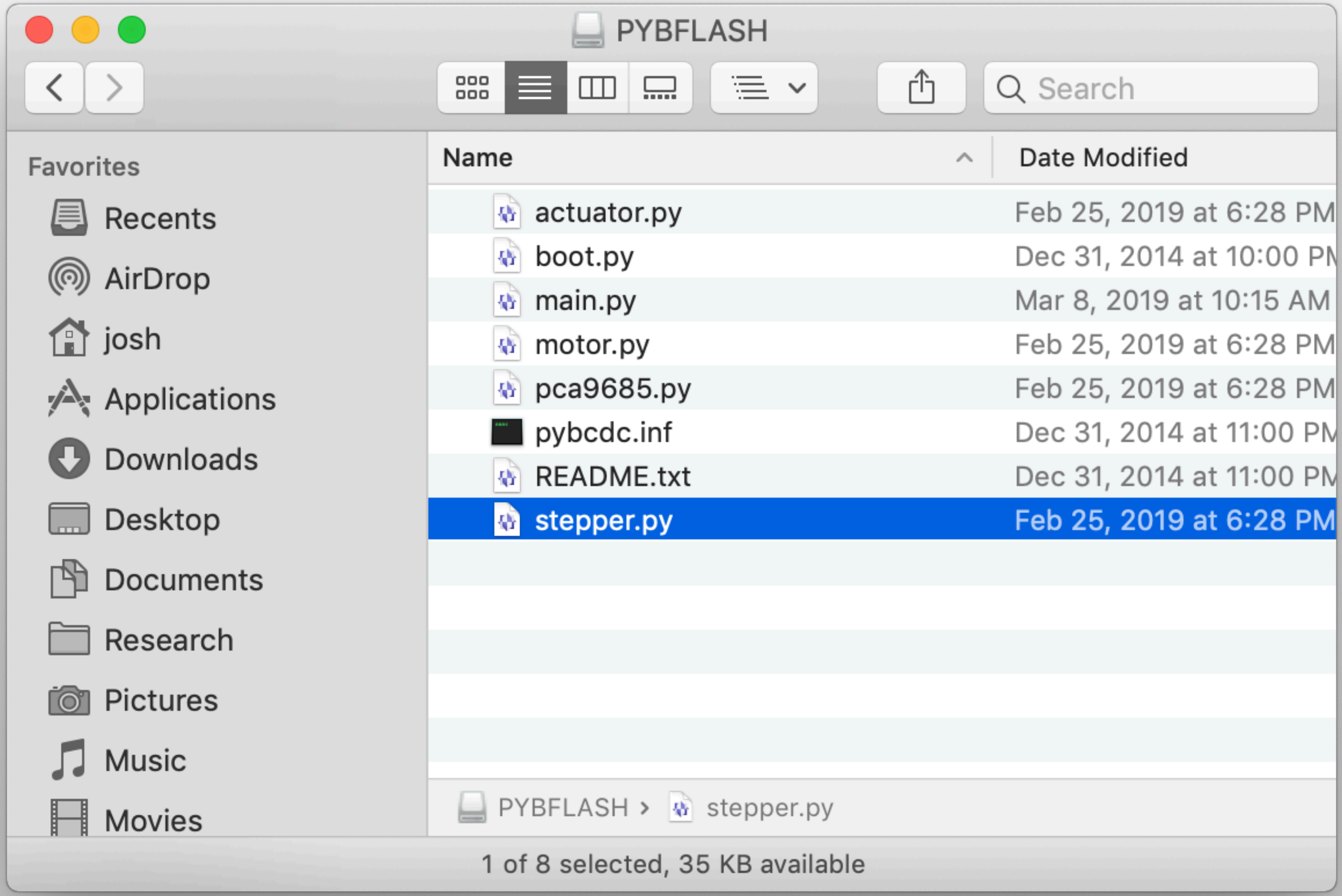
Installation of MCHE201 Libraries

- Go to: `https://github.com/DocVaughan/MCHE201_Controller`

- Download all the `.py` files from there

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

PYBLASH after install



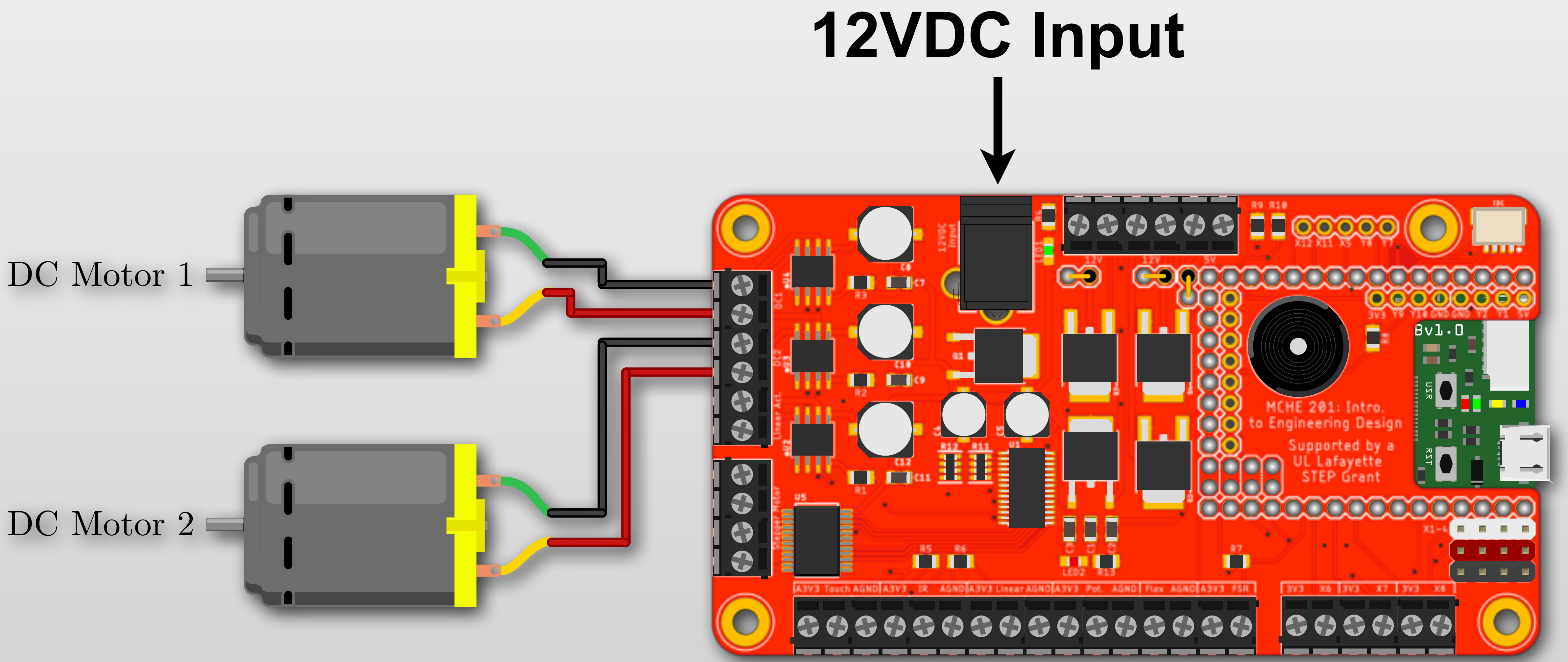
Initialization in MicroPython

```python
# We'll use the machine module i2c implementation.
import machine

# Initialize communication with the motor driver
i2c = machine.I2C(scl=machine.Pin("X9"),
                  sda=machine.Pin("X10"))
```

This is needed for all MCHE201 controller board scripts and should never need to be changed.

DC Motor Hardware Setup

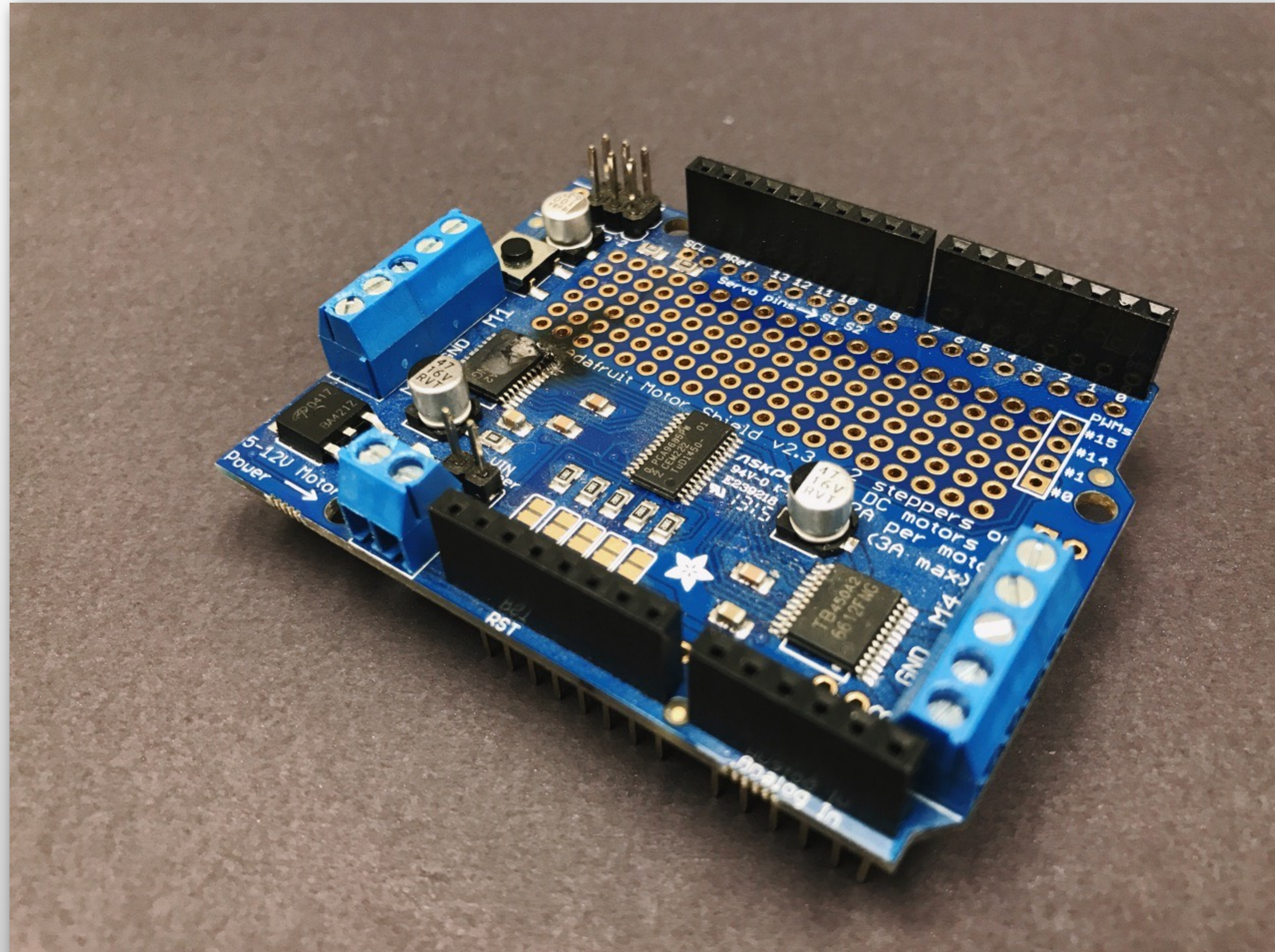
- Motor can be plugged into DC1 or DC2
- Do ***NOT*** let conductors on the leads touch



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

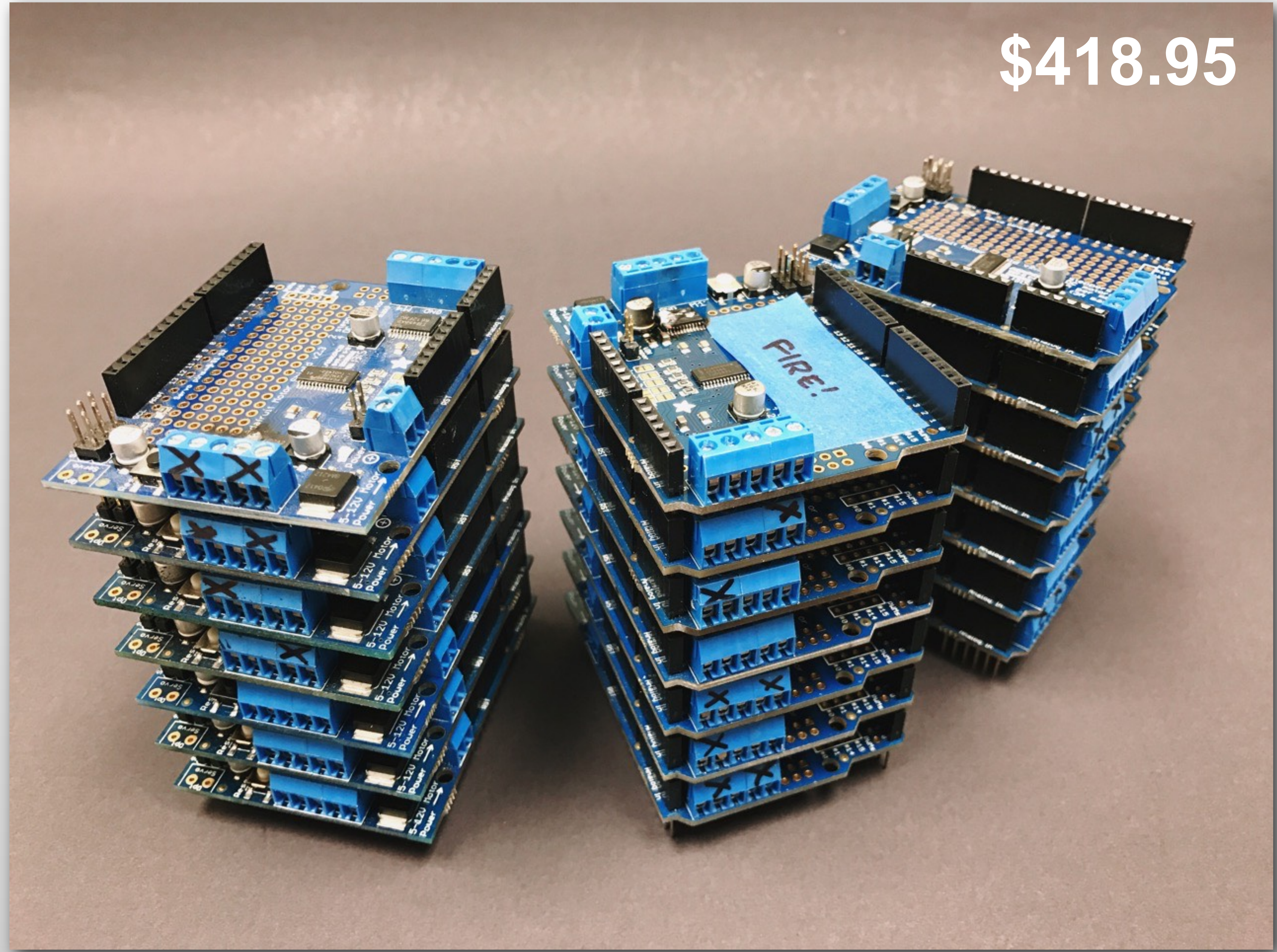
- Stripping too much wire from the motor connections
- Keeping stalled motors powered
- Reversing a motor without stopping it first

Entirely-avoidable Carnage



Entirely-avoidable Carnage

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
# i2c must already be set up as before
motors = motor.DCMotors(i2c)

# DC1 on the board is motor 1, DC2 is motor 2
MOTOR_NUMBER = 1 # DC1

# To control the motor, give it a speed between -100 and 100
motors.set_speed(MOTOR_NUMBER, 50)   # Go ~1/2 speed forward

# To stop, issue a speed of 0
# NOTE: ALWAYS STOP BEFORE SWITCH DIRECTIONS!!!
#       sleep() FOR A SHORT TIME TO LET THE MOTOR ACTUALLY STOP!!!
motors.set_speed(MOTOR_NUMBER, 0)

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

Stepper Motors

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

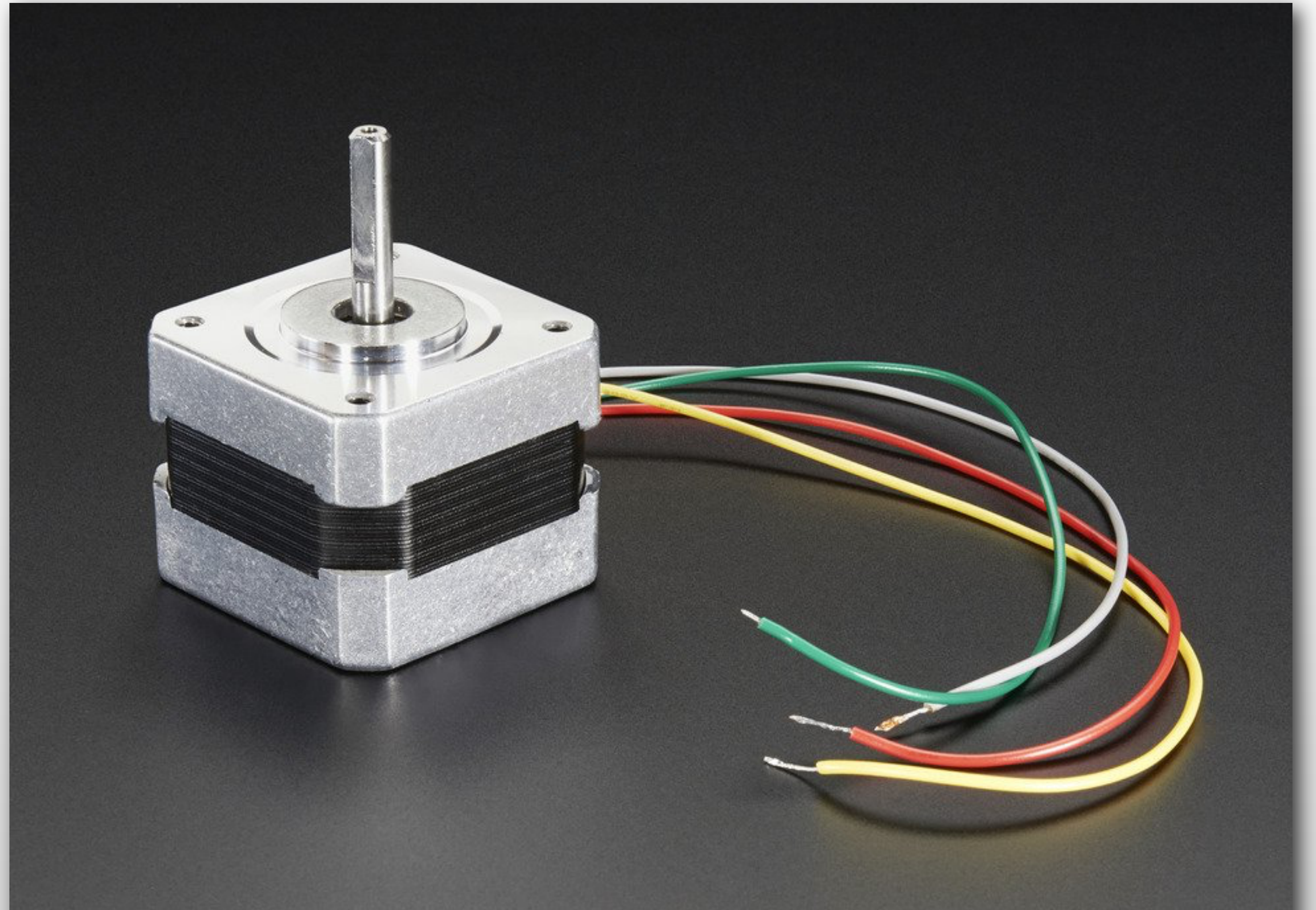
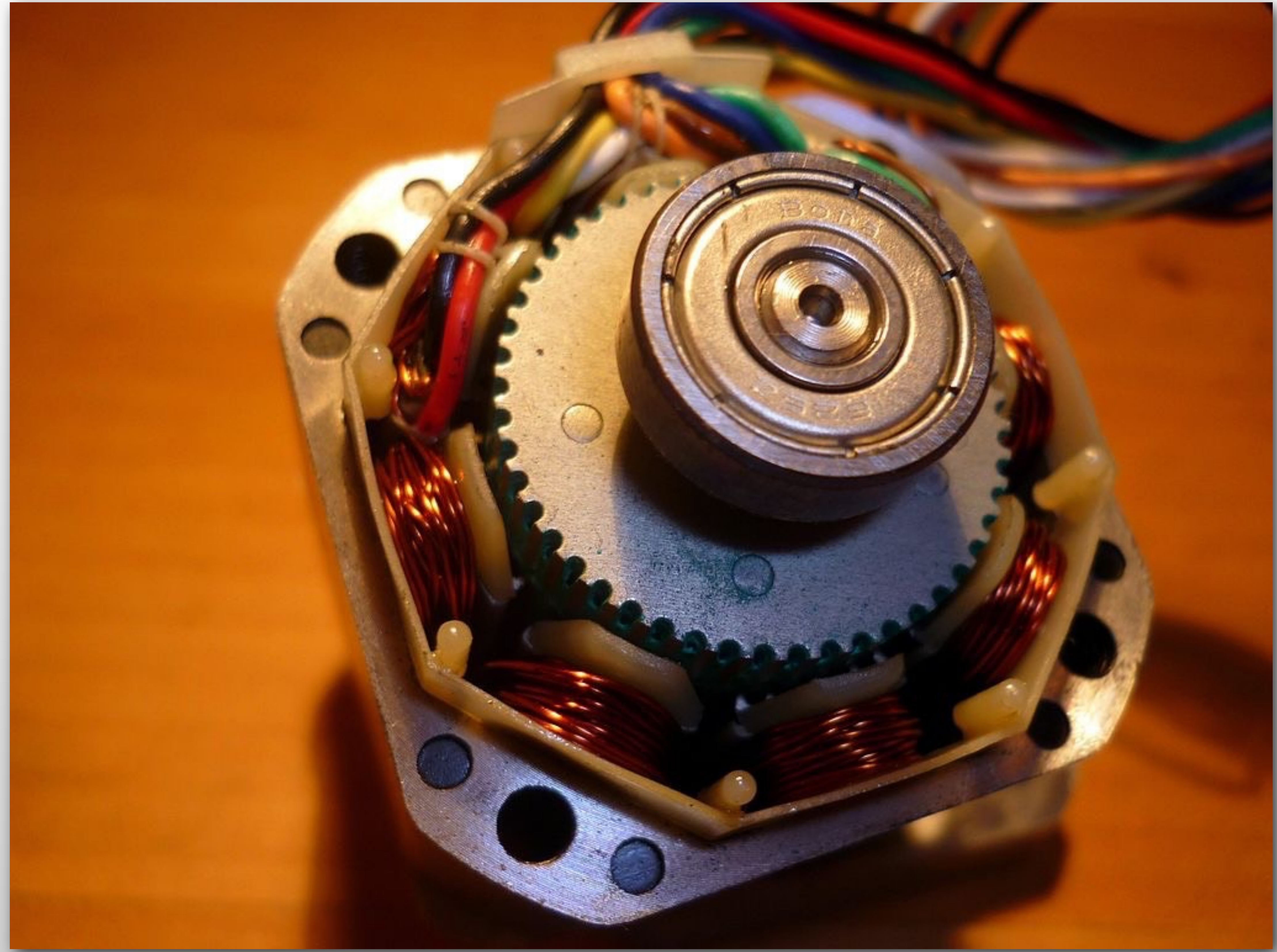
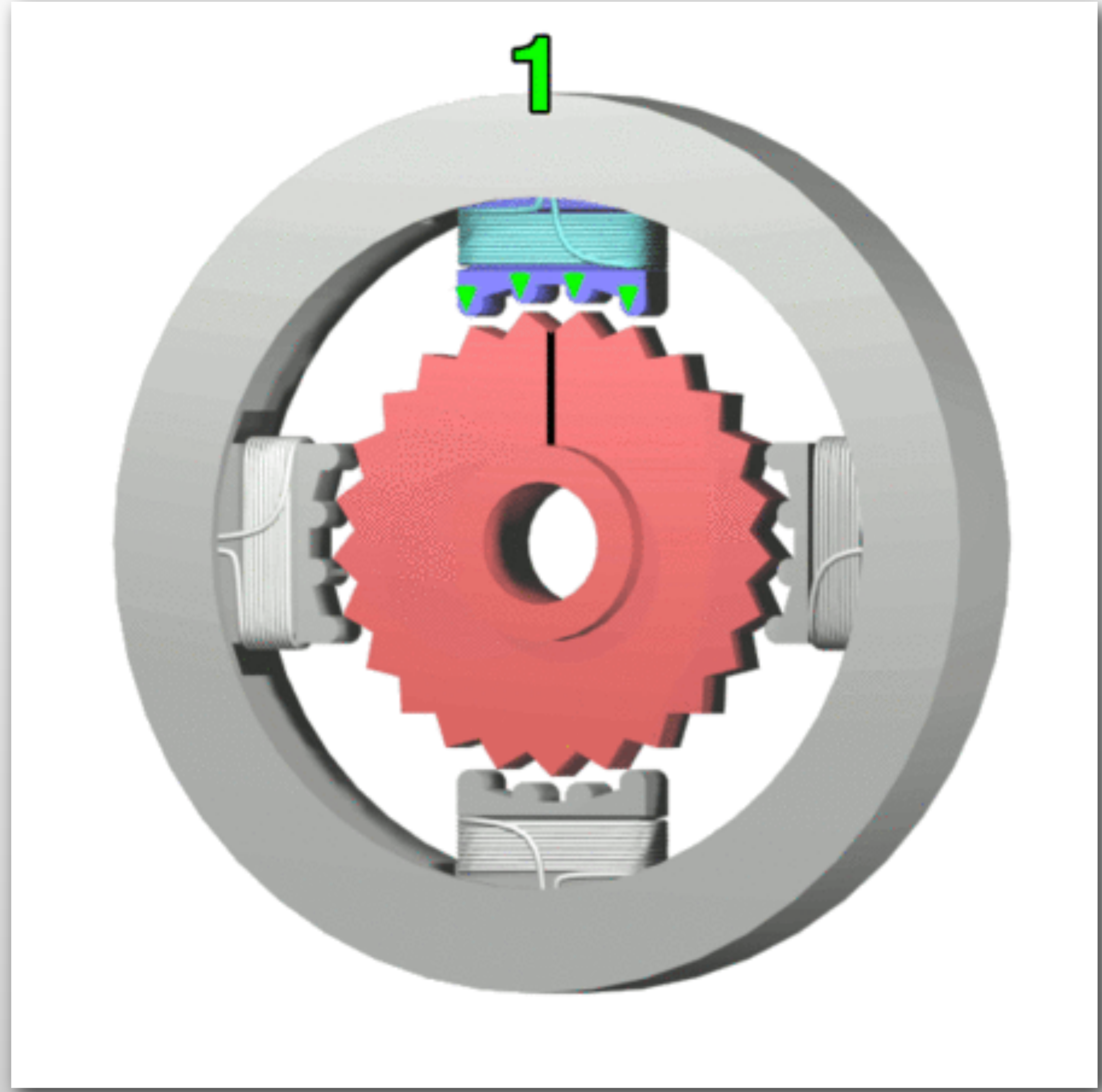


Image via https://www.adafruit.com/products/324

Stepper Motors

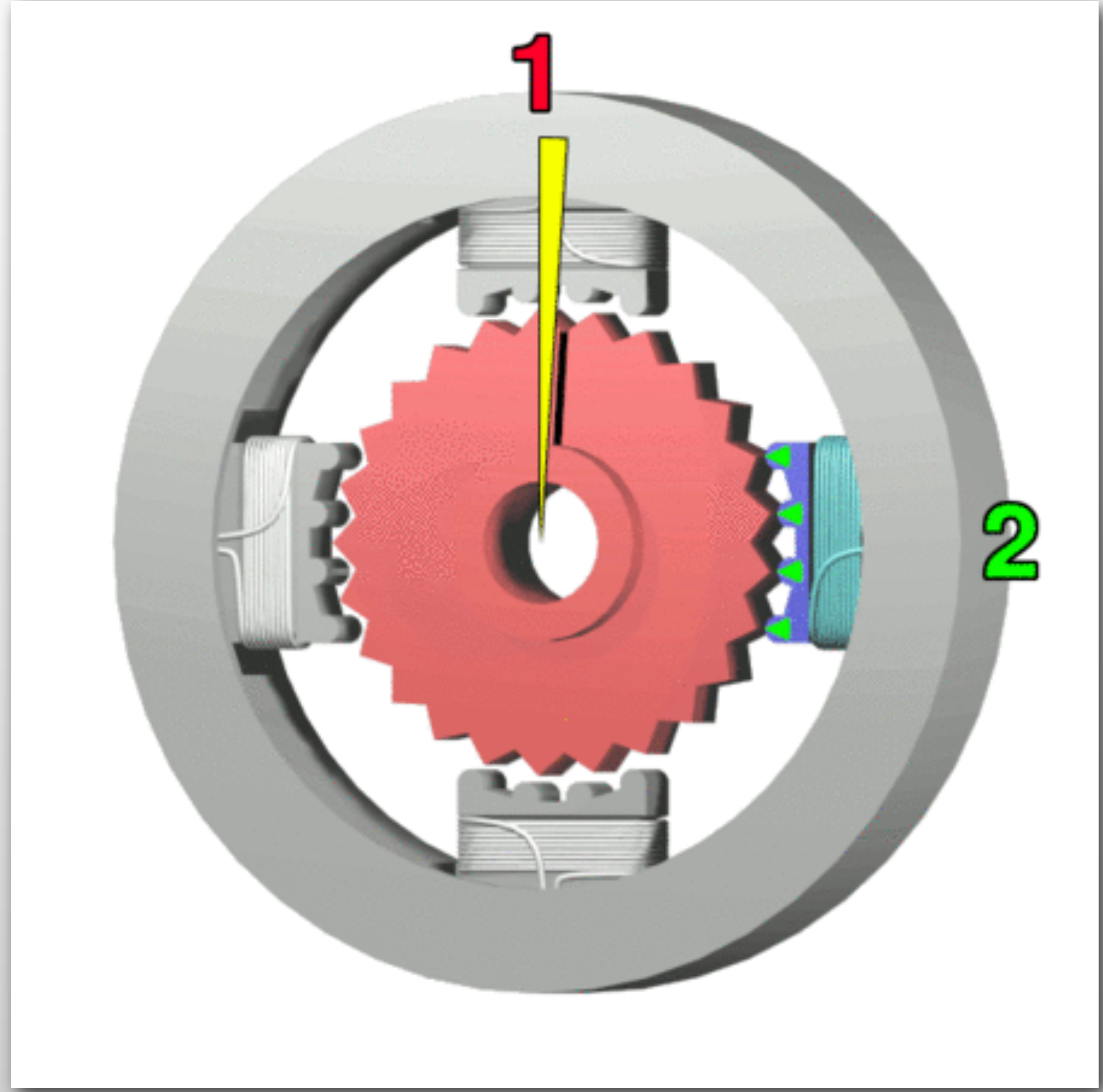


Stepper Motors



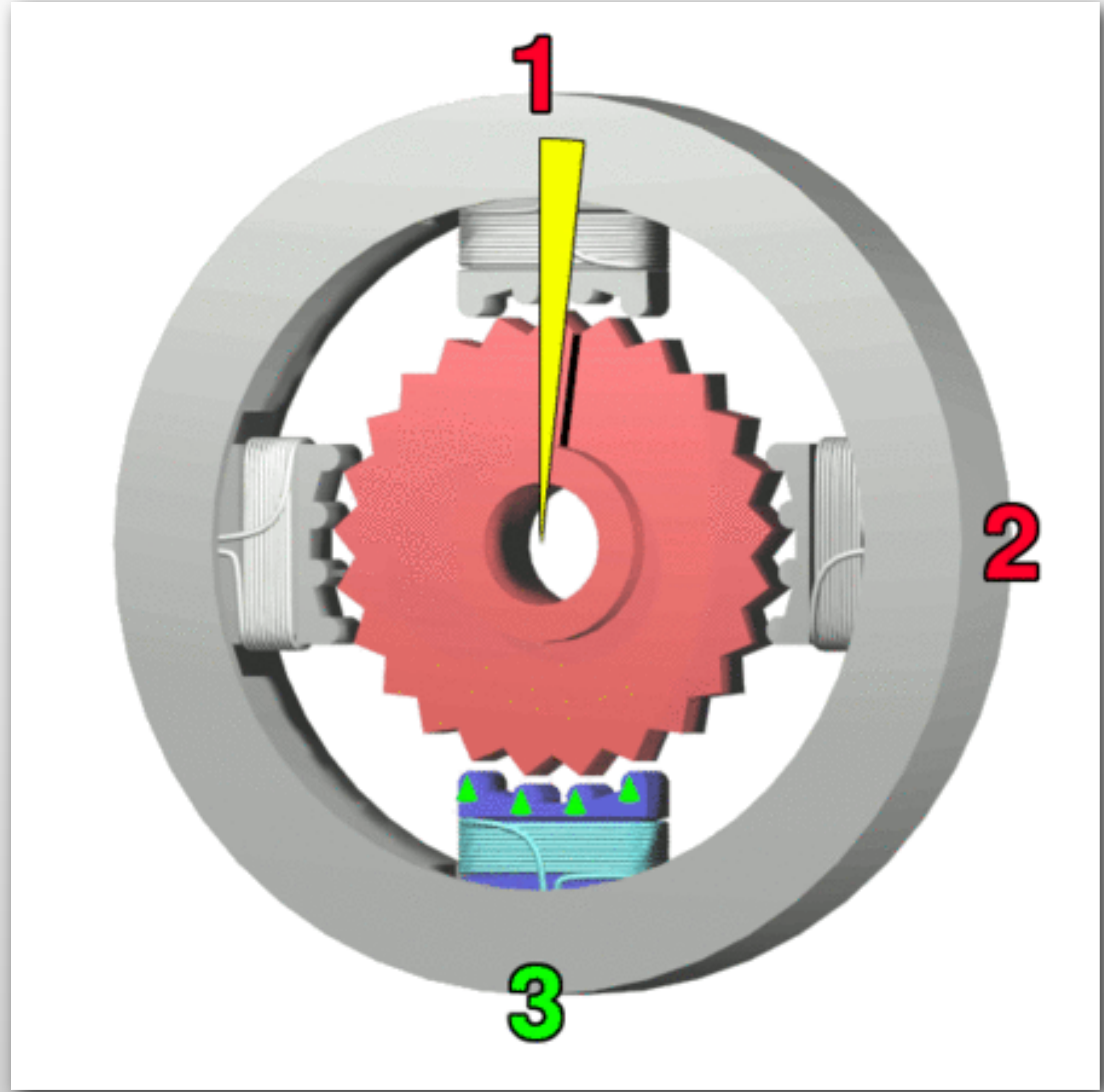
Adapted from "StepperMotor" by Wapcaplet; Teravolt.

Stepper Motors



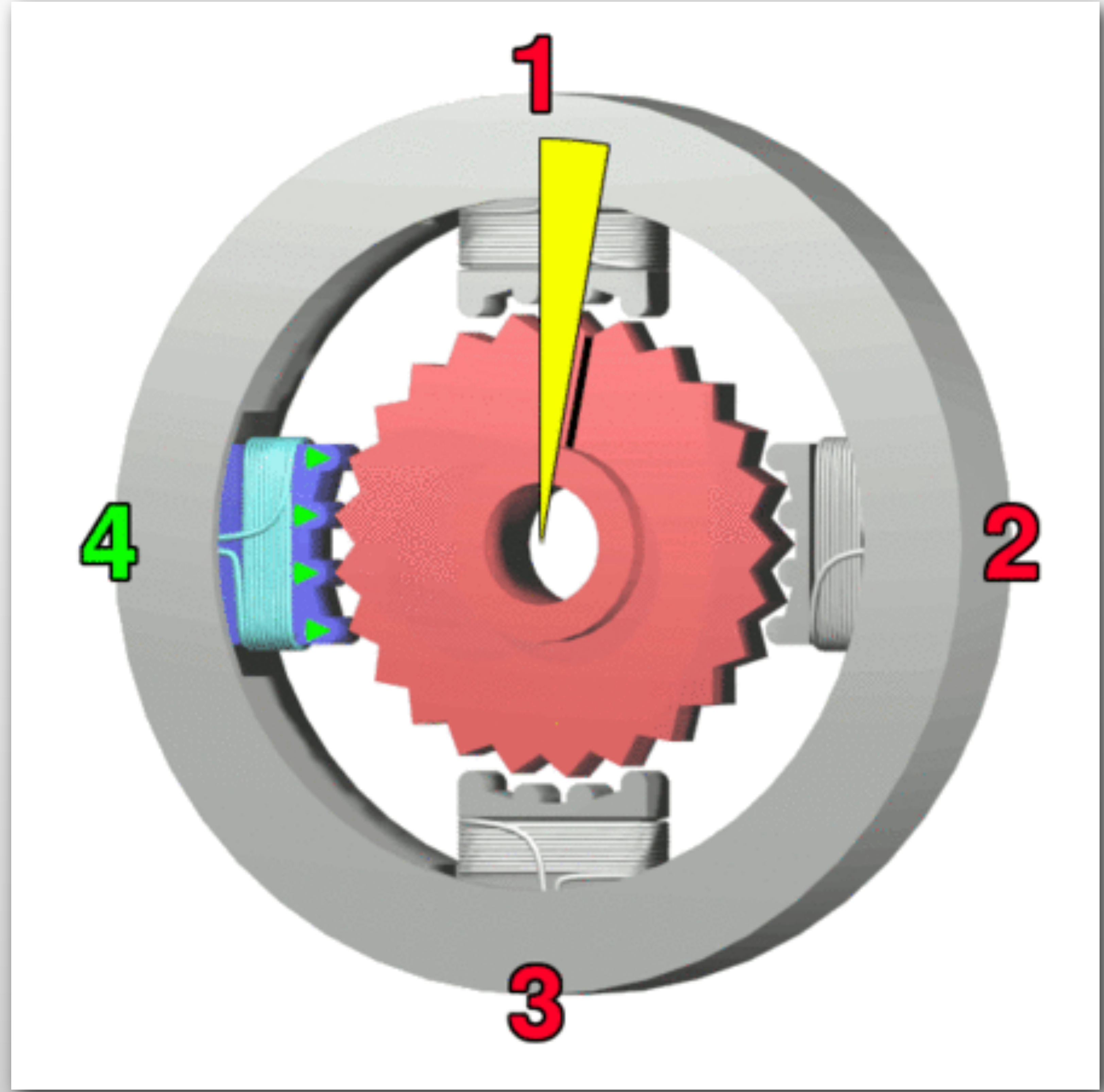
Adapted from "StepperMotor" by Wapcaplet; Teravolt.

Stepper Motors



Adapted from "StepperMotor" by Wapcaplet; Teravolt.

Stepper Motors



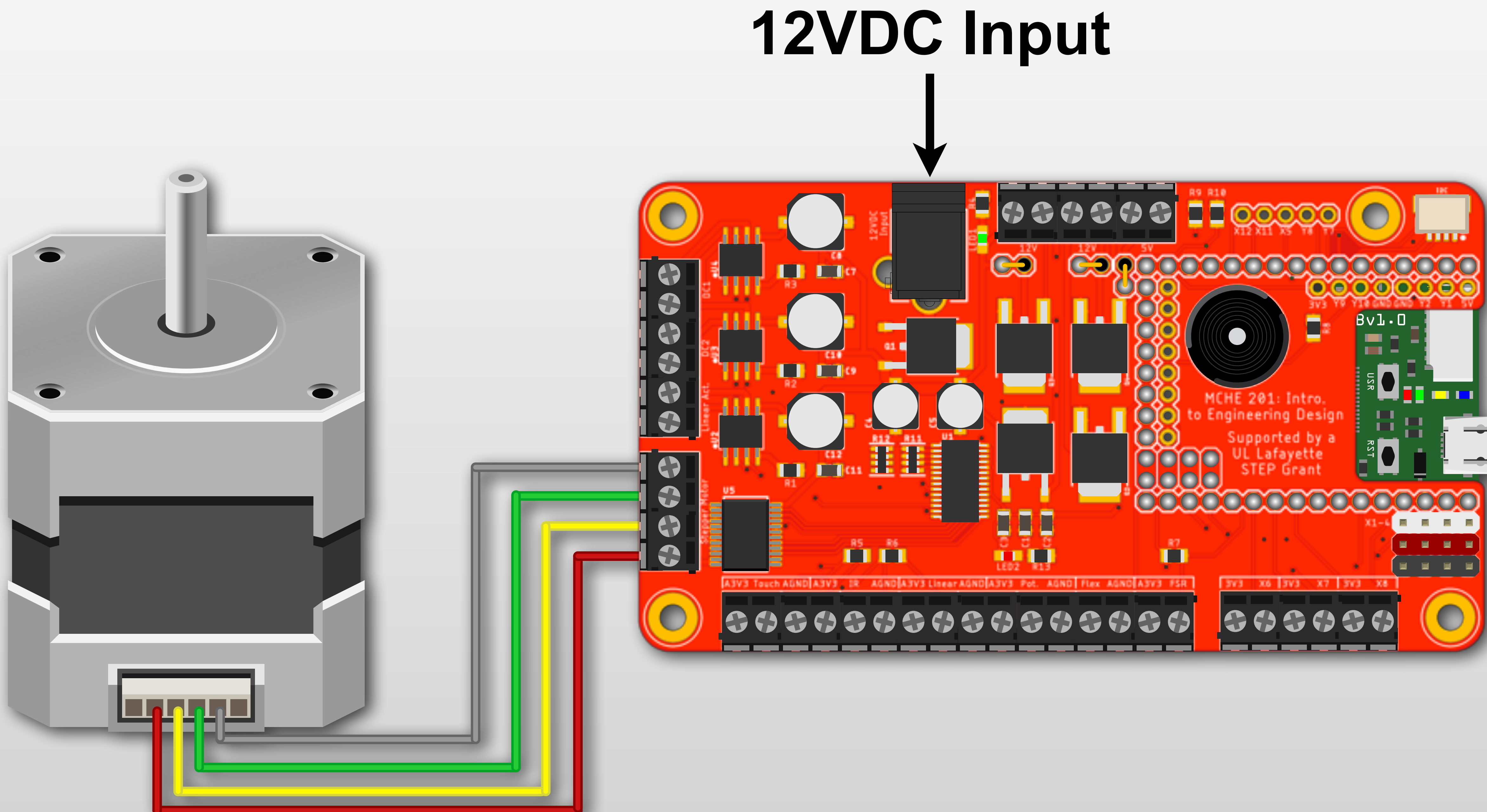
Adapted from "StepperMotor" by Wapcaplet; Teravolt.

Stepper Motor – Pros/Cons

- Pros
  - Precise
  - Quiet
  - Low Electromagnetic Interference (EMI)
  - Can be fully enclosed
  - Great for positioning tasks (can sometimes avoid sensors)

- Cons
  - Needs controller
  - Higher Initial Cost
  - Low torque

Stepper Motor Hardware Setup



Stepper Motor Initialization

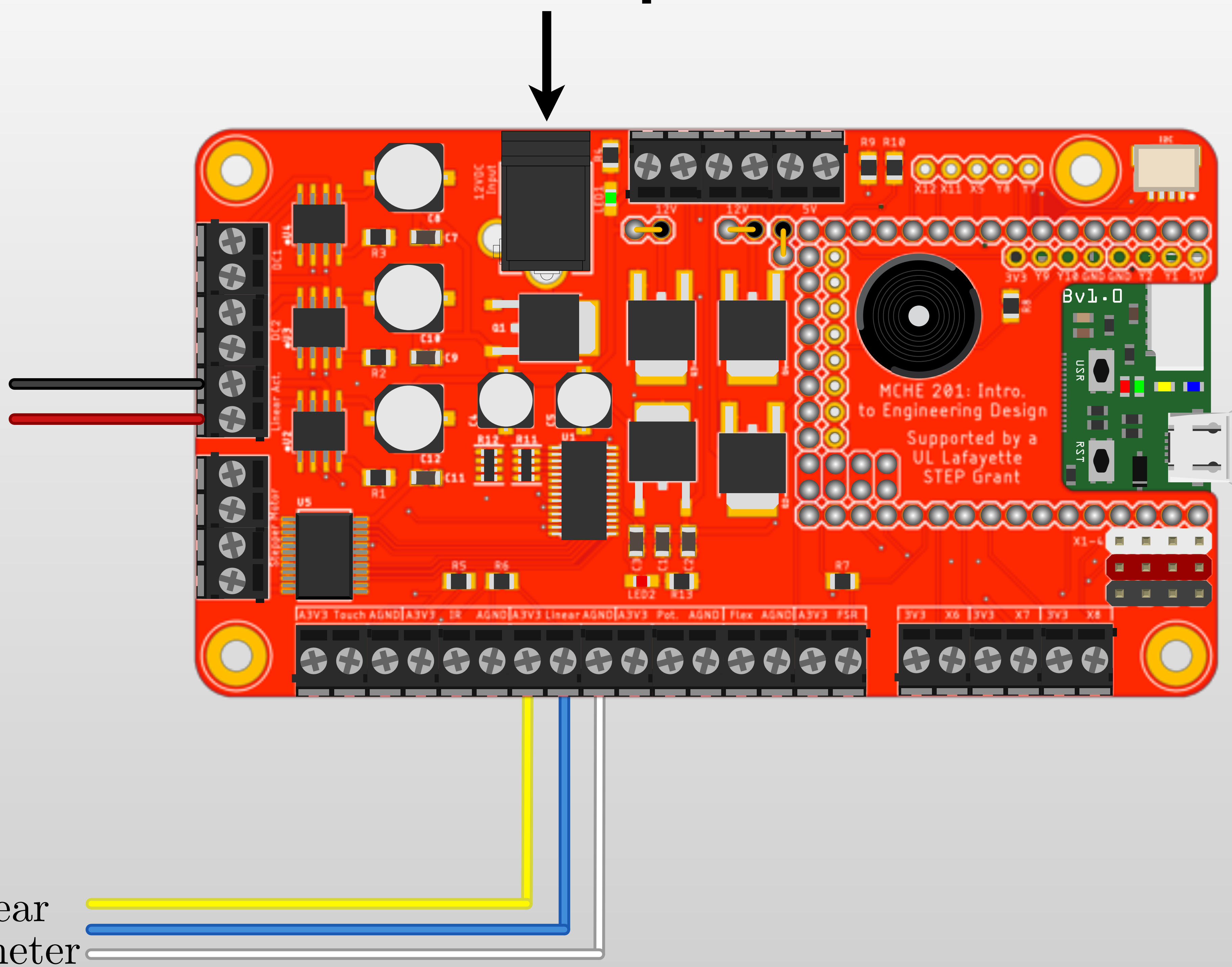
```python
# i2c must be defined as before.

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

# Now, we can initialize the stepper motor object
stepper_motor = stepper.StepperMotor(i2c)
```

Stepper Motor Core Functions

```python
# 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
stepper_motor.onestep(stepper.FORWARD, stepper.SINGLE)

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

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

Linear Actuator Hardware Setup

12VDC Input



Connected to Linear Actuator Motor

Connected to Linear Actuator Potentiometer

Linear Actuator Coding

- It is a DC motor. At the low-level, it's controlled like one

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

Linear Actuator Initialization

```
# i2c must be defined as before.

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

# Now, we can initialize the stepper motor object
linear_actuator = actuator.LinearActuator(i2c)

# Optional: Set up the analog-to-digital converter to read
# the linear actuator potentiometer that gives us
# information on its current length
linear_adc = pyb.ADC(pyb.Pin("X21"))
```

Linear Actuator Basic Control

```
# To control the actuator, give it a speed between -100 and 100
print("Moving at 1/2 speed in one direction")
linear_actuator.set_speed(50)     # Go 1/2 speed in one direction
time.sleep(0.5)                   # Continue at this speed for 0.5s

# ALWAYS STOP THE actuator BEFORE SWITCHING DIRECTIONS!!!!
# To stop, issue a speed of 0
print("Stopping.")
linear_actuator.set_speed(0)
time.sleep(1) # pause briefly to let the motor stop - 1s here

# To move in the opposite direction, give a negative speed
print("Moving at 1/2 speed in the other direction")
linear_actuator.set_speed(-50)    # Go 1/2 speed the other way
time.sleep(0.5)                   # Continue at this speed for 0.5s

# To stop, issue a speed of 0
print("Stopping.")
linear_actuator.set_speed(0)
```

GitHub

All of the code contained in this lecture is available at the MCHE201 Class Repository on GitHub:

`https://github.com/DocVaughan/MCHE201---Intro-to-Eng-Design`