MicroPython Introduction (cont.)

MCHE 201 – Spring 2018

Dr. Joshua Vaughan

Rougeou 225

joshua.vaughan@louisiana.edu

@Doc_Vaughan

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`

In-Class Exercise 5

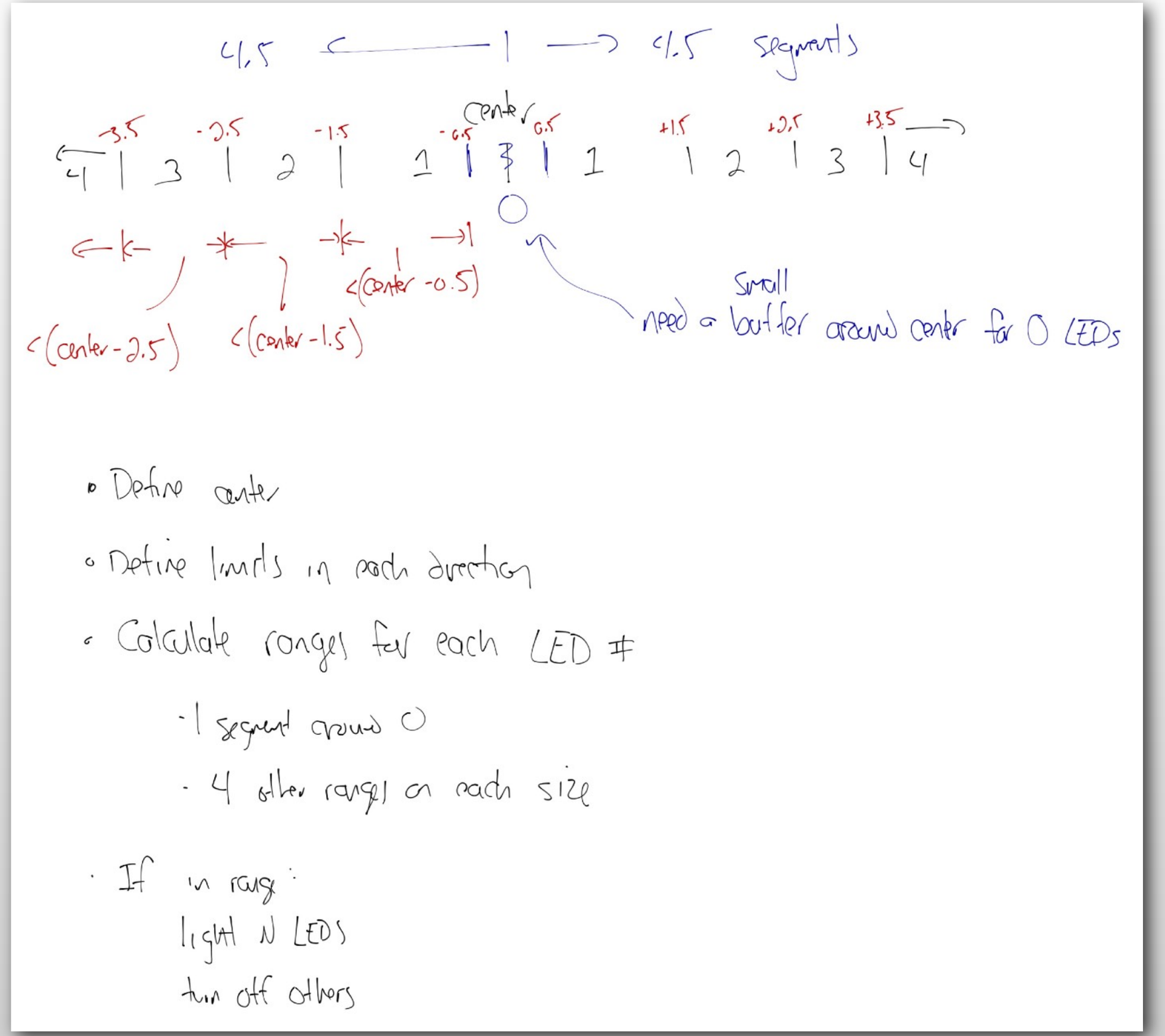
- Divide the flex sensor range into four
- Turn on the same number LEDs as the “range number” of the current state of the flex sensor.
- In other words, when the sensor is not bent, no LEDs should be on. When it’s bent a little, one LED should turn on. When it’s bent to its maximum, all 4 LEDs should be on.

In-class Exercise 5 Setup

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

# Set up the analog-to-digital converter
flex_adc = pyb.ADC(pyb.Pin("X22"))
```

```
# 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)
```

Wait… What's the *algorithm*?



- Define center
- Define limits in each direction
- Calculate ranges for each LED #
  - 1 segment around 0
  - 4 other ranges on each side
- If in range:
  light N LEDs
  turn off others

Setting up the Ranges

```
# These numbers will likely vary for your particular system.
# So, they should be determined experimentally.
MIN_ADC = 2875
CENTER = 3275
MAX_ADC = 3850

# Using the analysis above, we can define the size of each division
LOW_ADC_DIVIDER = (CENTER - MIN_ADC) / 4.5
HIGH_ADC_DIVIDER = (MAX_ADC - CENTER) / 4.5
```

```
# We'll create ranges both above and below the center
# This will account for the flex sensor being bent in either direction
ONE_ZONE_LOW = CENTER - LOW_ADC_DIVIDER * 0.5
TWO_LED_LOW = CENTER - LOW_ADC_DIVIDER * 1.5
THREE_LED_LOW = CENTER - LOW_ADC_DIVIDER * 2.5
FOUR_LED_LOW = CENTER - LOW_ADC_DIVIDER * 3.5

ONE_ZONE_HIGH = CENTER + HIGH_ADC_DIVIDER * 0.5
TWO_LED_HIGH = CENTER + HIGH_ADC_DIVIDER * 1.5
THREE_LED_HIGH = CENTER + HIGH_ADC_DIVIDER * 2.5
FOUR_LED_HIGH = CENTER + HIGH_ADC_DIVIDER * 3.5
```

The Reading and Check

```
# Now read the pot every 500ms, forever
while (True):
    # Read the value of the flex sensor. Should be in the range 0-4095
    flex_value = flex_adc.read()

    # print out the values, nicely formatted
    print("\nADC:     {:5d}".format(flex_value))

    # Check ADC value to determine to which of the ranges it belongs
    if flex_value < FOUR_LED_LOW or flex_value > FOUR_LED_HIGH:
        print("All LEDs on.")
        RED_LED.on()
        GREEN_LED.on()
        YELLOW_LED.on()
        BLUE_LED.on()
(several elif statements)
    else:
        print("No LEDs on.")
        RED_LED.off()
        GREEN_LED.off()
        YELLOW_LED.off()
        BLUE_LED.off()

    time.sleep_ms(500)
```

In-class Exercise 6

- Vary the intensity of the onboard blue LED based on how hard you are pressing on the FSR

- Pressing harder should make the light brighter

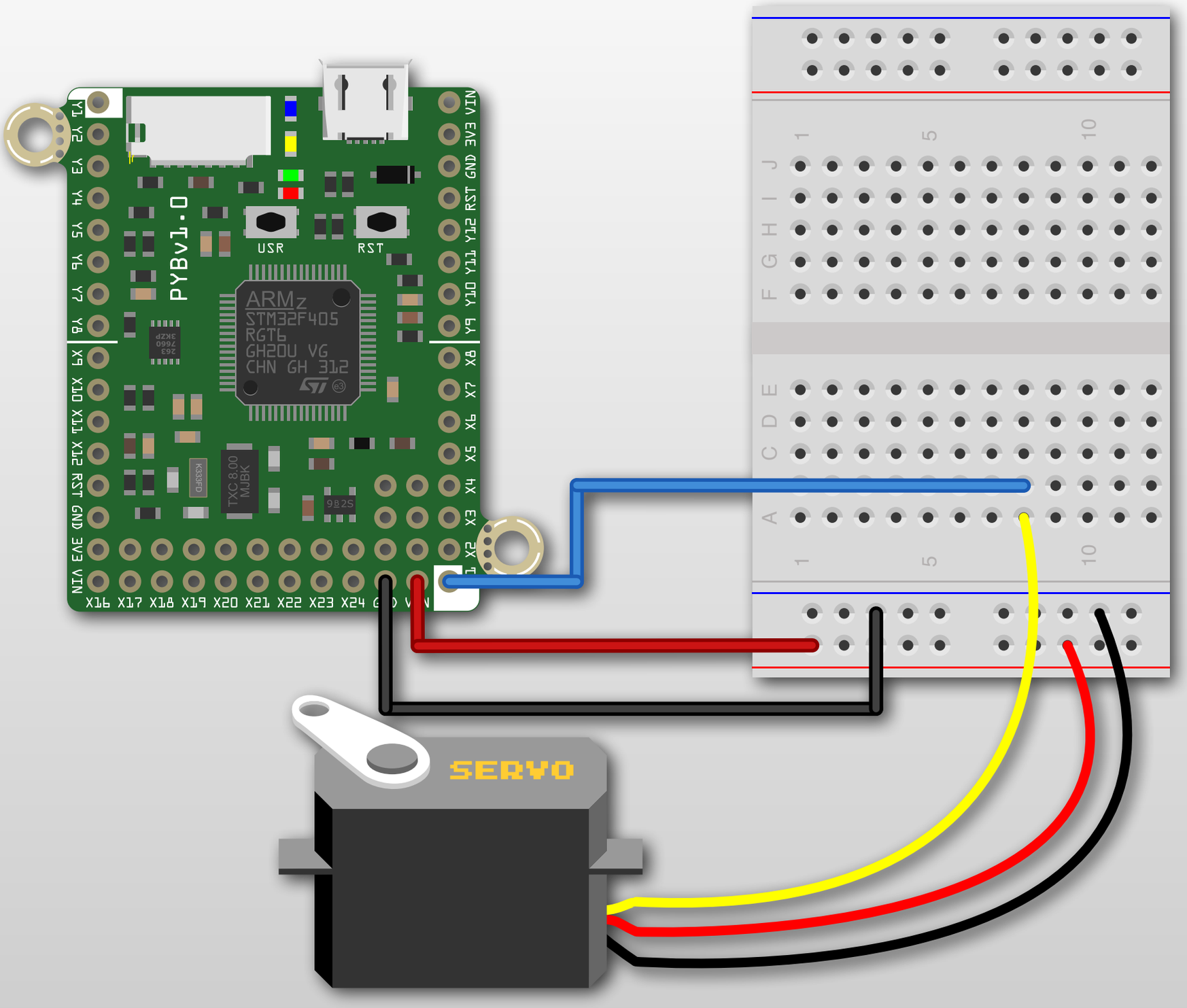
In-class Exercise 5 Setup

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

# Assign the 4th LED to variable BLUE_LED
BLUE_LED = pyb.LED(4)

# Set up the analog-to-digital converter
fsr_adc = pyb.ADC(pyb.Pin("X22"))
```

Wait… what's the *algorithm*?

- Have linear range of ADC in ~0-4095
- LED.intensity() expects integer from 0-255
- Define a function to map
  - Linear is good place to start (y=mx + b)
  - *Note:* Our eyes don't process light this way
- Based on that mapping, set LED intensity

Servomotor Hardware Setup



Servomotor Core Functions

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

# Now, we can control the angle of the servo
# The range of possible angles is -90 < angle < 90,
# but many servos can not move over that entire range. -60 to 60 is safer
servo1.angle(45)

# Sleep 1s to let it move to that angle
time.sleep(1)

# Move to -60degrees
servo1.angle(-45)

# To get the angle, call the .angle() method without an argument
current_angle = servo1.angle()

# Move to 45 degrees, taking 2seconds to get there
servo1.angle(45, 2000)
```

Reading User Input

- We can ask for user input from the REPL using `input()`

```
# Now, we'll ask the user for their input
print("Enter the desired angle in
degrees, then press return.")
desired_angle_input = input()
```

No guarantee the user will input a reasonable number… or a number at all.

MUST Check Input

Is it a number?

```
# We can us a try...except block to make sure
# the user actually input a number. If not,
# we'll use the current angle as the desired.

try:
    # convert to an integer
    desired_angle = int(desired_angle_input)
except ValueError:
    print("Please enter a valid number.")
    print("Remaining at current angle.")
    desired_angle = current_angle
```

MUST Check Inputs

Is is an *acceptable* number?

```
# Check that desired length is within the bounds of the actuator
if desired_angle > SERVO_MAX_ANGLE:
    desired_angle = SERVO_MAX_ANGLE
    print("The servo cannot move to that angle.")
    print("Moving to max. angle instead\n".format(desired_angle))
elif desired_angle < SERVO_MIN_ANGLE:
    desired_angle = SERVO_MIN_ANGLE
    print("The servo cannot move to that angle.")
    print("Moving to min. angle instead\n".format(desired_angle))
else:
    print("Moving to desired angle".format(desired_angle))

servo1.angle(desired_angle)
```

In-class Exercise 7

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

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

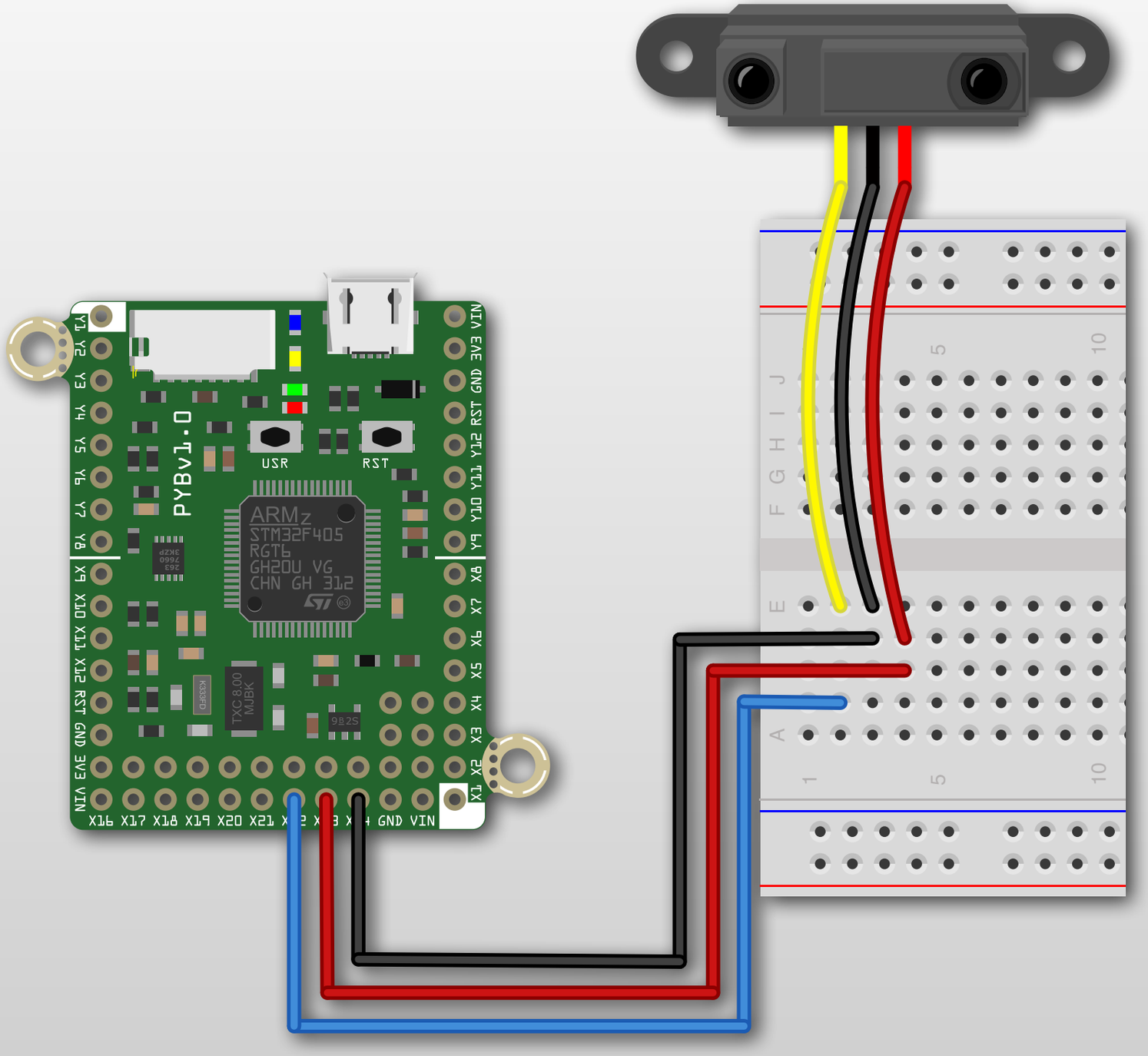
```
try:
    # Stuff to do if all is well

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