MicroPython Introduction (cont.)

MCHE 201 – Spring 2018

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 Setup

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

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

# Define constants for the min and max servo angles
MAX_SERVO_ANGLE = 75
MIN_SERVO_ANGLE = -75

# Set up the ADC for the potentiometer
pot_adc = pyb.ADC(pyb.Pin("X22"))
```

In-class Exercise 7 Angle Conversion

```
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 potentiomter 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 Example 7 Main Loop

```
# 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)
```

Controlling Timing

```
# Import time module
import time

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

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

```
# sleep for 500 milliseconds
time.sleep_ms(500)

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

Time Comparison

- Get the current time (to the ms or *μ*s) 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`

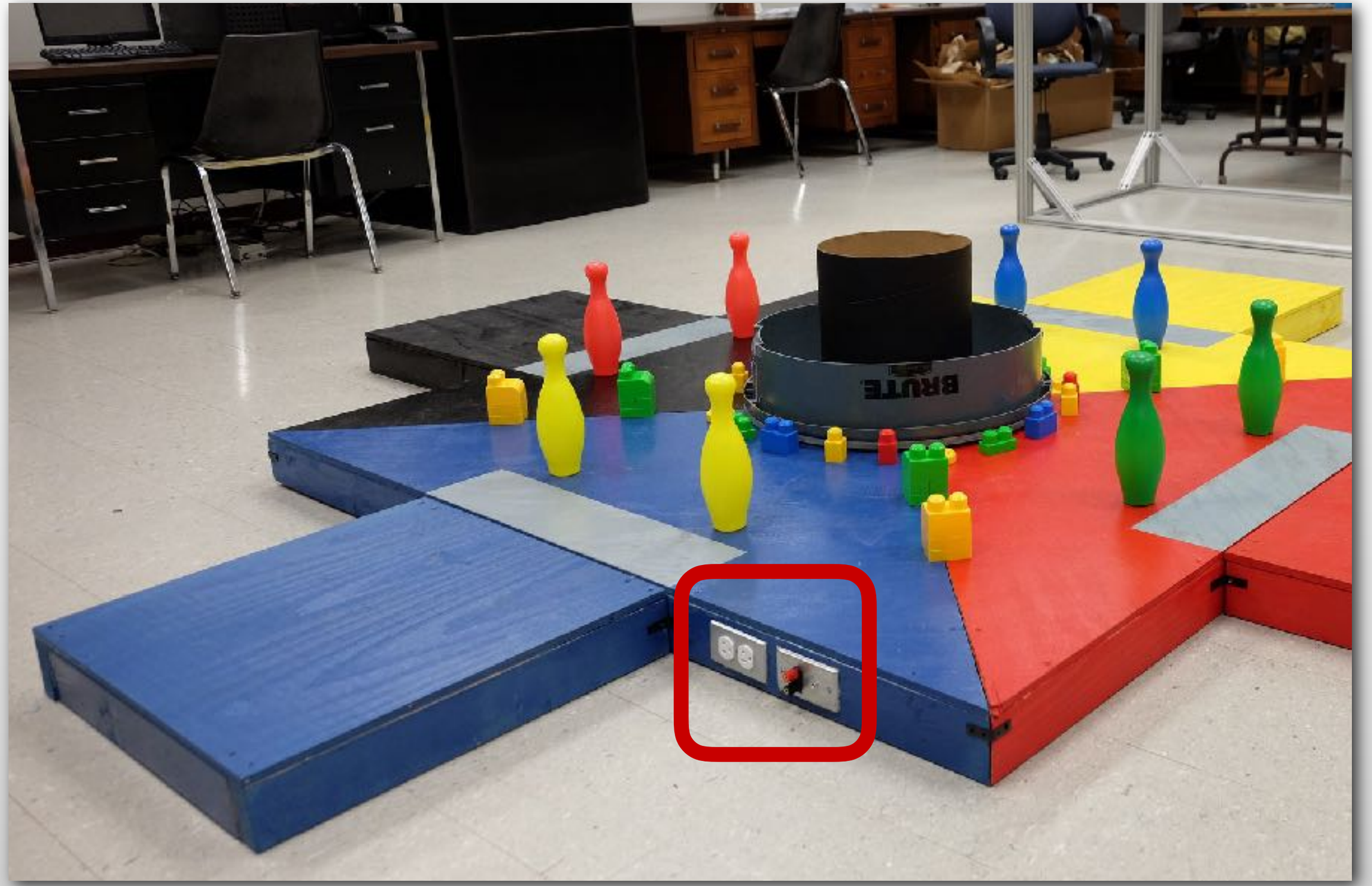
In-class Exercise 8

- 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 pressed to the REPL

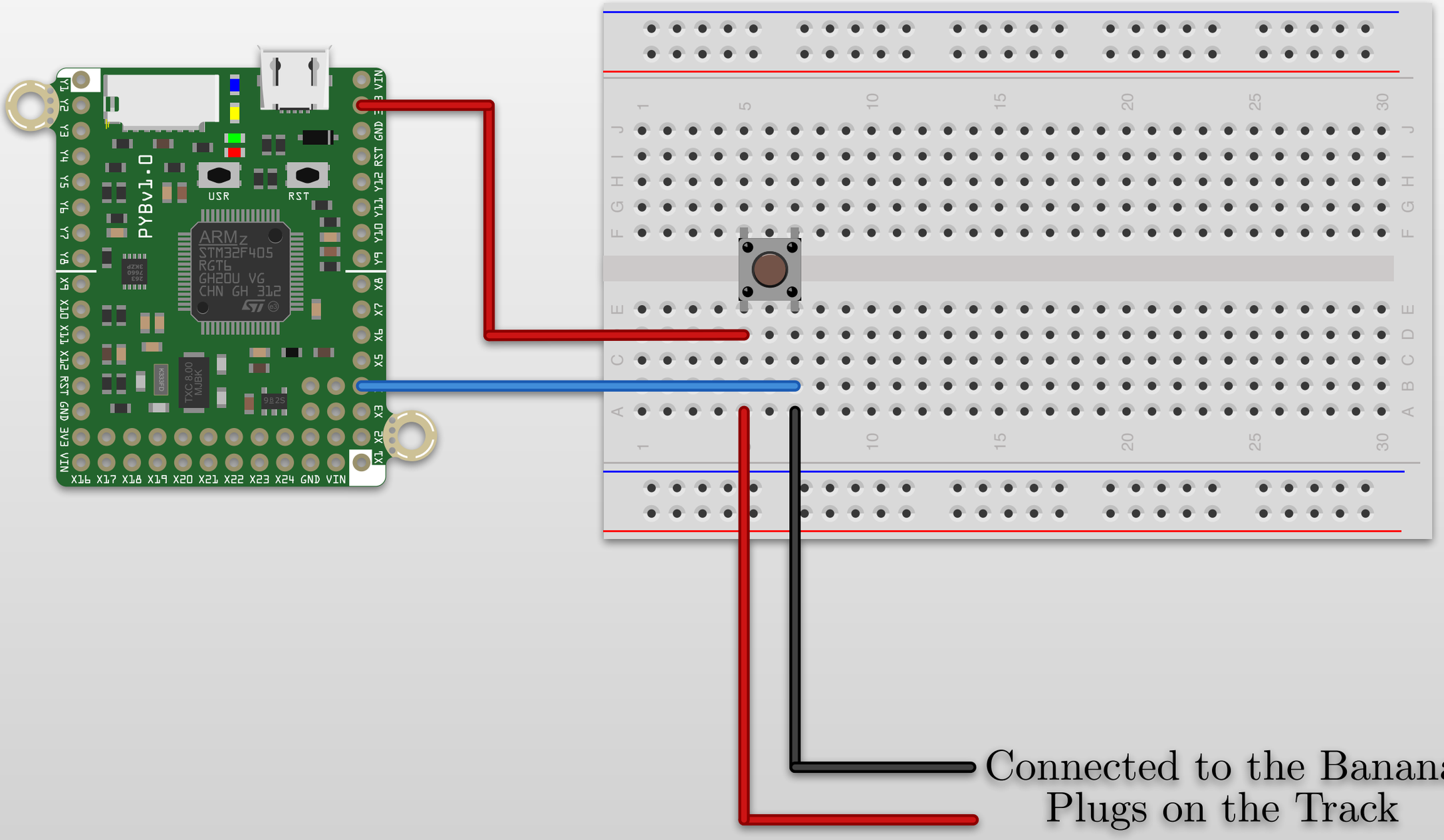
In-class Exercise 9

- 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 10s 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

MCHE201 Track Connections



Reading the MCHE201 Start Signal



MCHE201 Track Start Signal

- Will be closed for the 30-second trial time, open otherwise

- Works just like holding down that pushbutton for 30 seconds.

- 120VAC outlet is *always* 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

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