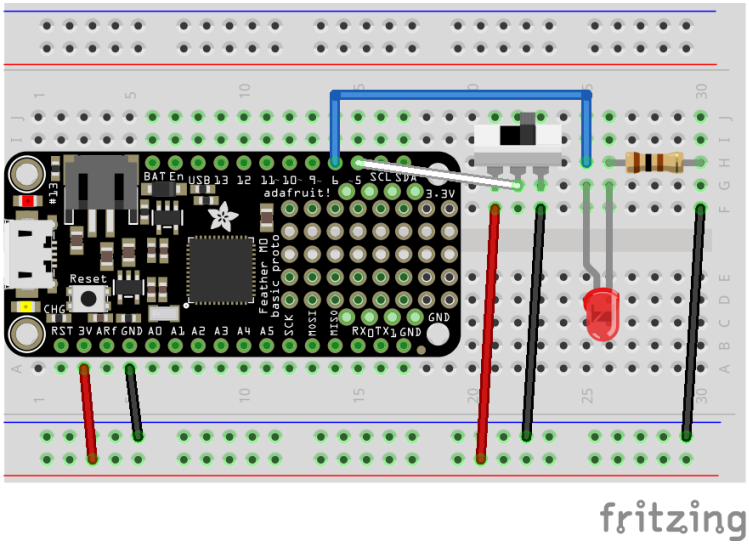
Introduction

Last week we learned about digital (0 or 1) output to make an LED turn off and on. This week we will learn about digital input (something the user or an object sends into the microcontroller). The input will be a button that we press. When the button is pressed, the microcontroller reads an input of 0 or LOW or False. When the button is not pressed, the microcontroller reads an input of 1 or HIGH or True. This seems backward, but it is the way the buttons are designed.

Procedure

Part 1: Operate an LED with a Button

First we need to set up our microcontroller to have both the LED from last week and the switch for this week. An image of the setup is below. The switch looks different from what you are using, but the wire colors are the same. We will use digital pin number 5 for the switch input.



In order for the switch to sense True (on) and False (off), we connect power from the micocontroller. The 3 Volt pin (red wire) is on and the GND pin (black wire, 0 Volt) is off. The switch is connecting on or off to pin 5 (white wire).

A basic code for indicating the button state is below. This code will turn on the LED when the button is pressed, and it will turn off the LED when the button is released (not pressed). Read the code carefully, and answer the questions below.

```
"""Lab 2 - Buttons
from CircuitPython Essentials Digital In Out example"""
import time
import board
from digitalio import DigitalInOut, Direction, Pull

# LED setup on digital pin 6.
led = DigitalInOut(board.D6)
led.direction = Direction.OUTPUT

# Button setup on digital pin 5.
switch = DigitalInOut(board.D5)
switch.direction = Direction.INPUT

while True:
# We could also if switch.value == True
if not switch.value:
led.value = True
else:
led.value = False

time.sleep(0.01) # debounce delay
```

We are using if statements to determine the state of the button. The code looks for a statement to be true. If the statement is true, the indented code after the if statement is executed. If the statement is false, the program moves on to the else or whatever code follows beyond the if statement. You can add print statements, e.g.,

```
print("LED on")
```

inside of the indented if statement code. What happens when you use

```
if switch.value == False
```

instead of

```
if not switch.value
```

1. **Why are we looking for the switch to be False?**
2. **The LED is connected to digital pin 6. What digital pin is the button connected to?**
3. **What are the variable names of the digital connections LED and button?**
4. **How is the setup of the the button digital pin different than the setup of the LED digital pin?**
5. **Explain how you send the SOS message using the button instead of code.**

Part 2: Reaction Timing

Let's turn it around. Now, we'll use code that randomly turns on the LED to test your reaction time. The code is below.

```
##### Lab 2 - Reaction Time
##### from CircuitPython Essentials Digital In Out example"""
import random #we need another library to generate a random timing
import time
import board
from digitalio import DigitalInOut, Direction, Pull

# LED setup on digital pin 6.
led = DigitalInOut(board.D6)
led.direction = Direction.OUTPUT

# Button setup on digital pin 5.
switch = DigitalInOut(board.D5)
switch.direction = Direction.INPUT

# variables to store information
ranDelay = 0.0 # random delay time in seconds
startTime = 0.0
rxnTime = 0.0 # start time and reaction time variables

while True:
    print("Push the button to start game")

    while switch.value == True:
        continue
        # wait for the user to push the button for start

    print("Get Ready!")
    time.sleep(1.0)
    print("Get Set!")
    time.sleep(1.0)
    ranDelay = random.random() * 5.0 # generate a random delay time 0 to 5 seconds
    time.sleep(ranDelay) # wait the random amount of time
    print("Go!")
    startTime = time.monotonic() # get the time
    led.value = True # turn on the LED

    while switch.value == True:
        continue
        # wait for the reaction button press

    led.value = False
    rxnTime = time.monotonic() - startTime
    print("Your time is ", rxnTime, " seconds.")
    time.sleep(2.0)
```

Q: Can you see in the code where the LED is being turned on at a random time after you press the button to start the program? Explain how the program works. As always, it's okay to ask your TA or instructor questions.

You will need to open the "Serial Monitor" because the code gives you instructions as it runs. There is a button on the left side menu that says "Monitor". This will let you see the information being sent from the Arduino to the computer. Write a brief explanation about how the timing game is played.

- Test your reaction timing 20 times, recording your reaction times in Excel.
- Repeat another 20 times, recording your reaction times as a separate set of data.

Analysis

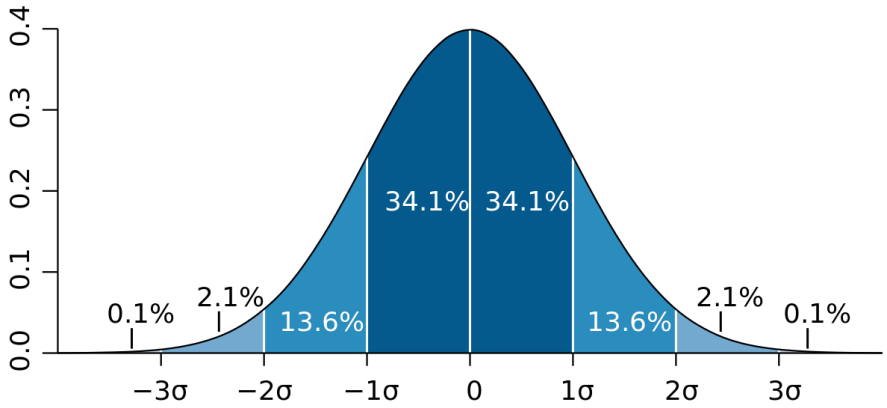
We want to know how reproducible your reaction times are. That is, how much statistical uncertainty is in your reaction timing and is one set of reaction times statistically similar to another set. We do this by calculating the average and the standard error (also known as the standard deviation of the mean).

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N-1}}$$

$$SDM = \frac{\sigma}{\sqrt{N}}$$

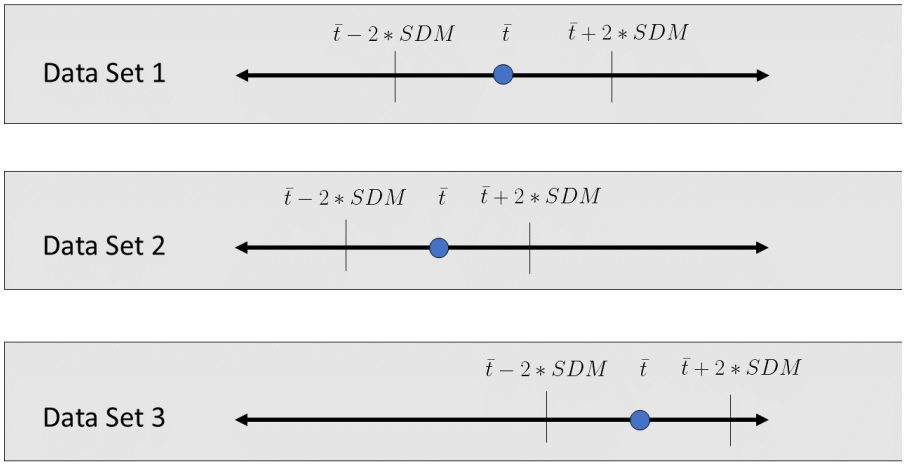
The first equation is the average $\bar{x}$. The second equation is the standard deviation of the set of x_i data when the data consists of a number N measurements. The third equation is the standard error of N measurements. One can see that as the number of measurements increases, the standard deviation and standard error go to zero. For an infinite number of measurements, we expect to know the average exactly, with no uncertainty. Since we cannot do infinite numbers of measurements, we use statistics such as these. These statistics under ideal circumstances tell us that 68% of our data will fall between one standard deviation below and one standard deviation above the average or mean. We expect 95% of our data to fall within two standard deviations of the mean. The following figure shows what an ideal measurement distribution would look like. The graph is the probability of measuring on the y-axis and the measured value on the x-axis.



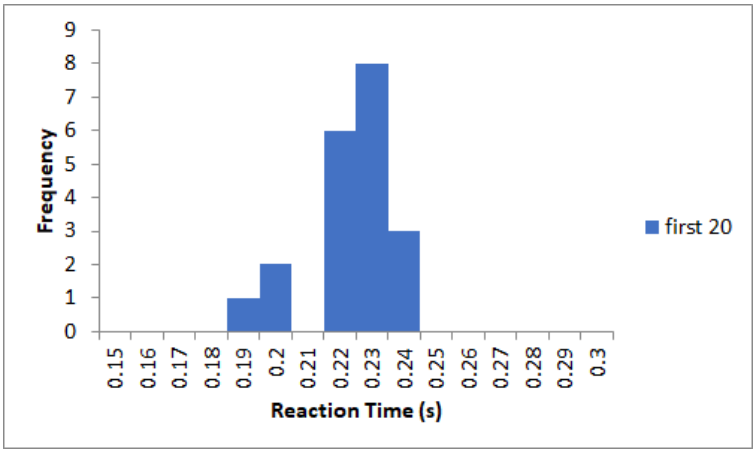
Thus, we might say that our reaction time is

$$t_{rxn} = \bar{t} \pm 2 * SDM$$

with 95% confidence. The following image shows a comparison of three different data sets. **Which data sets agree with one another with 95% confidence? How do you determine this?**



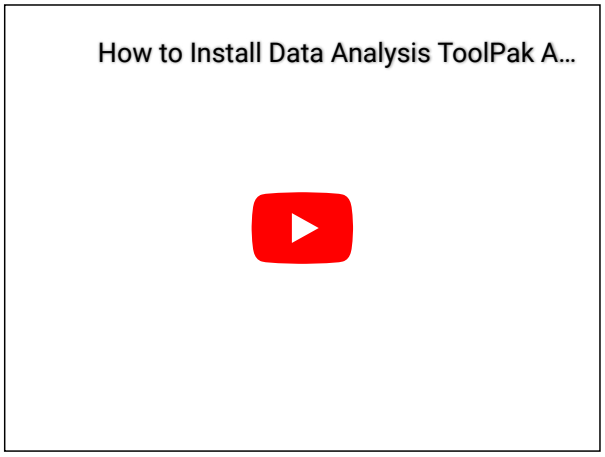
After 20 measurements, our distribution won't look quite as smooth as the ideal distribution above. It might look like the following graph. You will see how to make this graph in the next section.



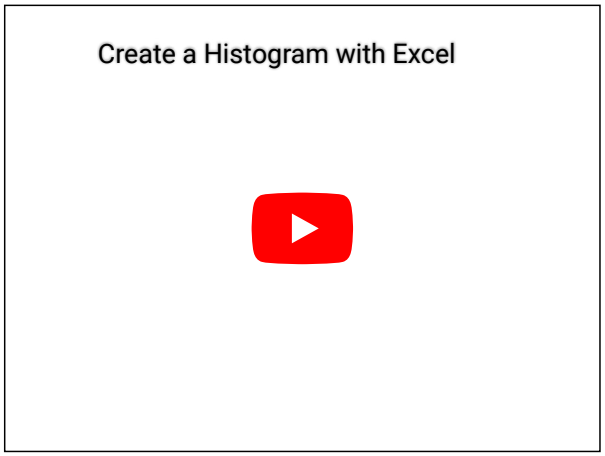
With your data, do the following analysis. Calculate the average. In Excel this is done by typing in a blank cell `=average(A2:A21)`, assuming your data is in cells A2, A3, ... A21. Calculate twice the standard error by entering into a blank cell `=2*stdev(A2:A21)/sqrt(20)`. Repeat for your second set of 20 measurements. Do your two sets of measurements statistically agree? The answer to this is determined by whether the ranges of t_{rxn} overlap.

Creating a histogram

To make a graph of your data, we'll use the data analysis tools in Excel. You may need to add these tools. Here is a video to add the tools.



Here is a video describing how to make a histogram.



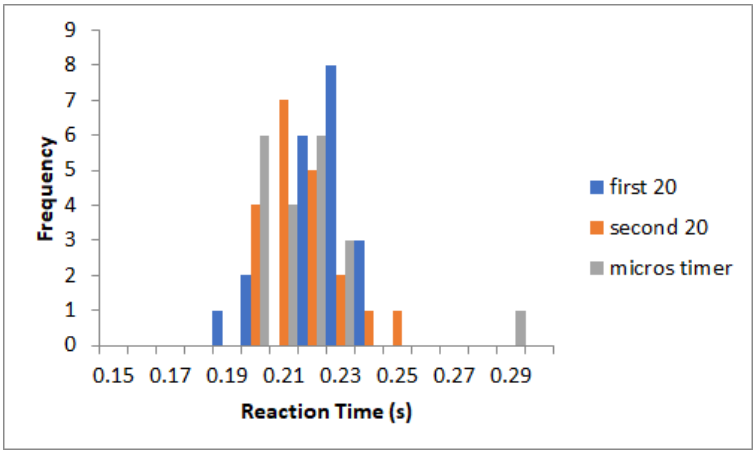
Create a histogram of your first 20 and your last 20 measurements. Put them on the same graph so you can see how they compare. Comment on the graph and the average and standard error analysis completed in the previous section.

There are two Arduino timers. The one used in the code is `millis()`, which measures milliseconds that have elapsed. The other counter is a microsecond counter called `micros()`. Replace the two instances of `millis()` with `micros()` and adjust the print statement to properly scale `rxnTime` to seconds. Make 20 measurements and determine if this timer results in

- statistically different reaction times.
- different standard error, i.e., different precision.

If not, then the Arduino apparatus is not likely affecting your data, and you are recording true reaction times.

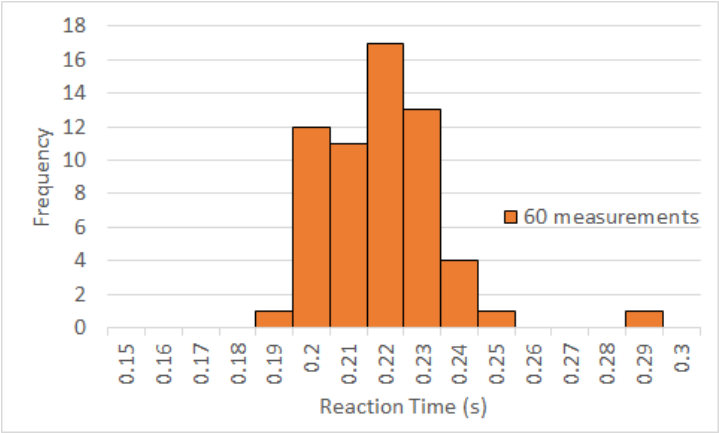
Here is the data I collected in graphical and numerical forms.



$$t_1 = 0.218 \pm 0.006\text{s}$$
$$t_2 = 0.210 \pm 0.006\text{s}$$
$$t_{micros} = 0.212 \pm 0.010\text{s}$$

Create a histogram of all 60 measurements and calculate the mean and standard error.

- **How do the 20 measurement and the 60 measurement data sets compare to one another?**
  - **Are the means the same?**
  - **Are the standard errors the same?**
  - **Would you expect the standard error to be the same for more measurements? Explain in terms of the standard error formula.**



Turn in a report with answers to all questions and your analysis here. Be sure to save your analysis. We will use it in two weeks to compare to results we collect then.

Make prints of your graphs to put into your notebook.

EXTRA

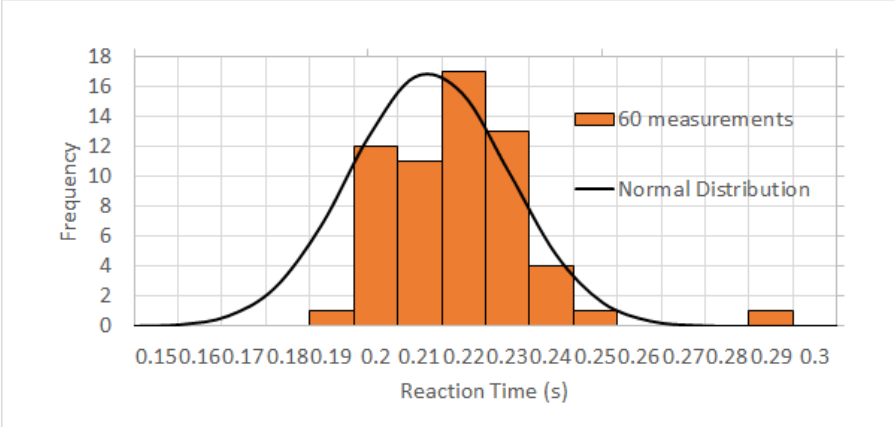
The equation that describes the normal distribution is called a Gaussian Function. It has the form

$$f(x) = Ae^{-(x-\bar{x})^2/2\sigma^2}$$

The variables are

- A is the amplitude. You can enter the maximum from your histogram frequencies.
- x are the reaction time bins used in making your histogram.
- $\bar{x}$ is the average of the 60 measurements.
- σ is the standard deviation.

You can go above and beyond my expectations if you make a graph like the following and comment on the agreement between your data and the normal distribution. **What would you need to do to make the data exactly match the normal distribution?**



Last modified: Wednesday, August 16, 2023, 10:45 AM