Jerry Chen
A13310365
February 09, 2018

ECE 16 Lab 2: Data Sampling & Characterization

1. Introduction

This lab introduced us to data sampling with CurieIMU and analogRead, timing sampling with polling and timers (particularly CurieTimerOne), reading and writing information from and to files, and displaying multiple data on a chart with SerialMonitor. The first two objectives provided insight on timing data sampling using either polling methods in loops or CurieTimerOne. Objective 3 helped show how acquiring information from the Arduino board analog pins and the CurieIMU gyroscope and accelerometer. Objective 4 demonstrated how to identify orientation and positioning of the based on the gyroscope, while objective 5 enabled sampling and displaying specific data at regular intervals, transferring this information to python, and writing it to files. Objectives 6 introduced us to displaying data on SerialPlotter, objective 7 demonstrated how to time and detect user responses via motion, and objective 8 showed how to process saved information using response data gathered from objectives 5 and 7.

2. Objective 1: Polling Sensors from Arduino

Objective 1 involves polling data from an analog pin via analogread on the Arduino board using a sketch named **SensorPolling**. In this case, we will start by defining the global variables, namely int(s) sampNum and analogPin, and unsigned longs prevTime and currTime. In the setup function, we set the analogPin we check to 0, set up Serial and record number of samples made in an int variable sampNum. Then we start recording the time with setting variable prevTime to micros() and print out the data heads at the end of setup and shift to loop. IT IS CRITICAL THAT THIS OCCURS HERE AS EACH PRECEDING STATEMENT TAKES TIME TO EXECUTE. In loop, we set an if block statement with the condition micros() - prevTime >= 10000, essentially meaning that each time the loop is triggered, if the current time occurs over 10000 microseconds after the last iteration of the if statement, all of the if statement's contents will be triggered. In the if block, we print out the current sample number and analogRead value from analogPin and time since the last analogRead.

Sample Number	Sensor Reading	Time Difference
1	920	10720
2	626	10709
3	428	10709
4	305	10708
5	229	10711
6	180	10710
7	152	10709
8	136	10711
9	128	10711
10	125	10777
11	122	10776
12	124	10778
13	124	10779
14	124	10777
15	123	10778
16	123	10778
17	122	10778
18	119	10779
19	122	10776
20	120	10776
21	121	10779
22	122	10778
23	122	10778
24	123	10778
25	121	10777
26	122	10779

Figure 1: A sample of the results from Objective 1's SensorPolling

As we can see from the above, the polling period is not necessarily consistent and can be prone to error due to the fact that the statements in each loop take time in microseconds to execute and requires recalculating the loop interval to account for the error produced by the execution of the loop contents. And even with recalculating time-based logic errors are still bound to occur as the time difference fluctuates. Ultimately, the need for recalculation and propensity for notable errors in time means that this is not an effective data collection strategy. With the timing issues in mind, that is where CurieTimerOne comes into play in the next objective.

3. **Objective 2: SensorTimer**

SensorTimer has the same global variables (sampNum, analogPin, prevTime and currTime), but requires including the CurieTimerOne.h directory. The setup is mostly the same as in SensorPolling, but with the key addition of CurieTimerOne.start(10000, &sample), which leads to a to be defined function sample. In addition, everything in the if statement in the to be cleared loop statement of SensorPolling will be shifted to sample. The CurieTimerOne acts independently timing of the contents inside of sample and will execute sample whenever 10000 microseconds has transpired since the last execution of sample, meaning that it will have much fewer and smaller timing errors.

Sample Number	Sensor Reading	Time Difference
1	922	10729
2	629	9988
3	431	9998
4	309	10000
5	234	10000
6	184	10000
7	158	10000
8	143	10000
9	135	10000
10	128	10068
11	127	10000
12	127	10000
13	127	10000
14	126	10000
15	124	10000
16	126	10000
17	125	10000
18	127	10000
19	127	10000
20	126	10000

Figure 2: A sample of the results from Objective 1's SensorPolling

As can be seen here, SensorTimer is more accurate and is preferable as a data collection strategy compared to SensorPolling's time difference.

4. **Objective 3: Timing sensor readings from the Arduino**

   As can be seen from objective 1's inconsistent results, each statement requires time to fully execute. In objective 3, we observe the execution time of gathering data from an analog pin, and the CurieIMU accelerometer and gyroscope in an Arduino sketch we will call **SensorIntervals**. To minimize the effect of Serial interfering with the timing, we set it to begin with a high bit transfer rate (115200). In addition, we will set the IMU with an accelerometer range of 2 and a gyro range of 250 and, as usual, set the analogPin to zero. In loop we repeatedly print out the microseconds for accelerometer read, analogRead, and gyroscope read by recording the start time immediately before and the end time immediately after each read and then print it out. To summarize we read the inputs starting with the accelerometer, then analog pin, then gyroscope. Each of the three inputs has us type a five statement block: the labeling print statement to clarify what is being read, a recorded start time in micros immediately before the read statement, the actual read statement, the recorded end time immediately following, and finally the print statement with the time difference between start and end times.

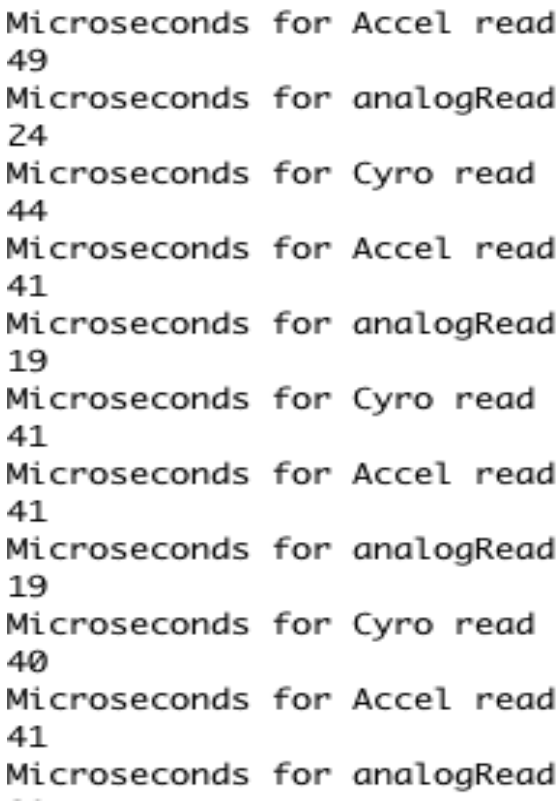

Figure 3: A portion of the resulting output of SensorIntervals

From the above data we can now see the approximate time required for each value read.

5. **Objective 4: Arduino Orientation Applet**

   A major feature about accelerometers and gyroscopes is that they tend to be used in products that require orientation identification, including tablets and smartphones. It is here in objective 4 that we identify the orientation of the Arduino board. Now the accelerometer works by observing shifts in its internal components as a result of its external body's exposure to forces and its internal components' inertia/reaction. These include normal forces and gravity, which ultimately result in some accelerometer values representing portions of gravity instead. In other words, we can determine the orientation of the board using gravity.

   With this information, we can now write the orientation applet OrientationApp. First we will need to time the interval between each orientation check and check the accelerometer, so we will need to include CurieTimerOne.h and CurieIMU.h. Also, since we will be comparing previous and current orientations, we will need a last orientation int variable to represent the last orientation. In setup(), we begin the

CurieIMU, set the accelerometer to 2, and start up serial with a high bit transfer rate (115200), give it time to get ready then set the timer to a preferably large number (in this case 100000 microseconds or 0.1 second), set the lastOrientation to -1 (which represents no previous recorded orientation) and attach the ISR checkOrient. Like previous timer based functions, we will leave loop empty. In checkOrient, we will first need to identify which axis of the board is experiencing the most force/gravity. In other words, we will need to identify the axis with the largest absolute value of force acting on it and then determine the direction. To do this, we first must readAccelerometer and save them in local variables x, y, and z. Next we take the abs of each of them and save them as absX, absY, and absZ and compare their magnitudes. If absZ has the largest magnitude, the board is likely facing up or down, depending on whether or not z is positive or negative. If absY has the largest magnitude, the board is likely in landscape position, depending on whether it is positive or negative. If absX has the largest magnitude, the board is likely in portait position. The images below display the output and board position.

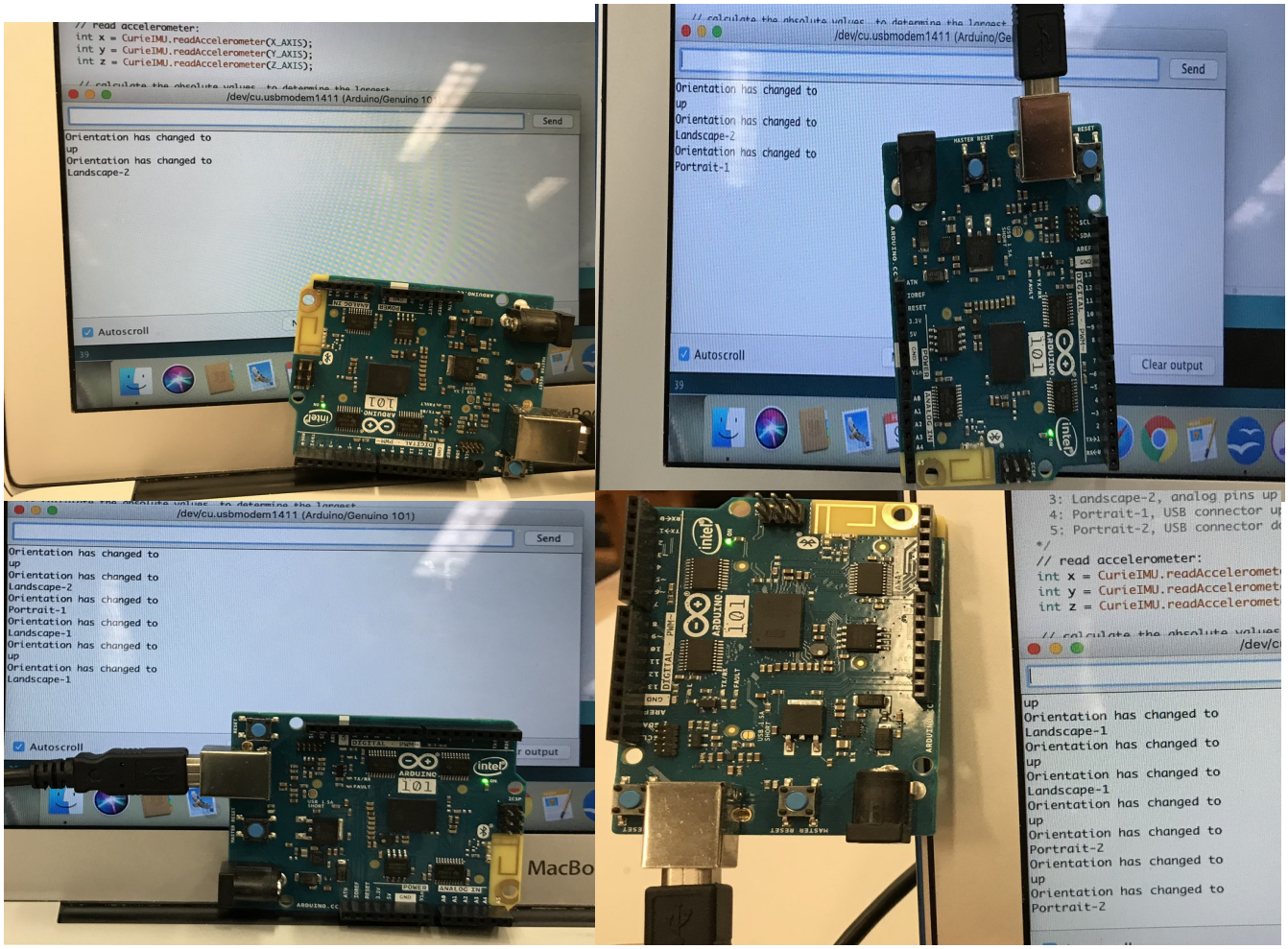

Figure 4: the position of the board and the resulting output each time.

6. Objective 5: Logging Data to Python

This objective reintroduces pySerial, demonstrates the usage of I/O between files and enables us to record information in .txt files. Here we will write an Arduino sketch named **DataCollection** and a python script named **data_collection.py** to write data from the accelerometer and gyroscope to a .txt file

In the Arduino sketch, we will need to include CurieTimerOne to sample periodically and CurieIMU to sample from the board. We will also need variables for saving unsigned longs starting time (the time we start the program) and current time, and x-y-z and pinVal (measured voltage of the pin) variables to save motion measured by the IMU and a boolean newRead to indicate each time a new value has been read such that the loop's contents (separate from the ISR) are triggered separately from the ISR so as not to interfere with its execution. We set up Serial in setup() with a bit rate of 115200 and begin the CurieIMU and set the gyro and accelerometer ranges and wait for serial to get ready and for the user to respond from serial with an input number. Once the input is selected and saved, we set newRead to false and set the start time in miilis and start the CurieTimerOne to trigger every 5000micros or 5 milliseconds to attach to a function samplingISR. In the samplingISR function, we sample from the selected input (accelerometer, gyroscope, or analog pin) and flag a boolean each time an input is retrieved. Now in loop, we record the current time in each iteration, and if a newRead has been retrieved from samplingISR, it prints out the time elapsed since the input was selected and the gathered data of each execution of samplingISR before and sets newRead back to false.

In the python script, we import serial to enable communication via pySerial and set the serial port and rate to communicate with the Arduino board and exchange bits at a large rate (115200) and set printCount (the number of lines that have been printed and saved) to zero. We then form a serial connection between the board and provide encoded input to the Arduino board to determine which source to gather data from and write the data to the appropriate file (once we open it with fo = open(file_name, 'w')), which has a string name determined by the user's selected source. It then starts reading lines and reads until all the needed lines have been read. In this case we need to read 5000 samples. And since in my sketch each sample prints out two lines, printCount should stop one it reaches 5000 * 2 (printCount < 5000 * 2). Once this is done we close the serial connection and the .txt file. The resulting files can be found

attached with sampling occurring regularly as expected.

7. Objective 6: Arduino Serial Plotter

Here we now start using Serial Plotter in an Arduino sketch called **SerialPlotter** to print and plot data on the Serial Monitor and Plotter respectively. We will include CurieIMU and CurieTimerOne for sampling and start setting up in the setup function by setting Serial.begin(115200) and starting up and setting the IMU gyro and accelerometer. We then allow Serial to set up before printing out a request to select what to sample from. Once the user has provided the answer, we indicate that the program is initializing and set up the timer at 5000 micros based on the user input to either trigger sampleAccel or sampleGyro, both of which sample and print the magnitude of the accelerometer/gyroscopic readings from the selected source.

Now in Serial monitor we will select the input and then close it before we open the Serial Plotter and toy around with the board, rotating and moving it around and checking the plotter.

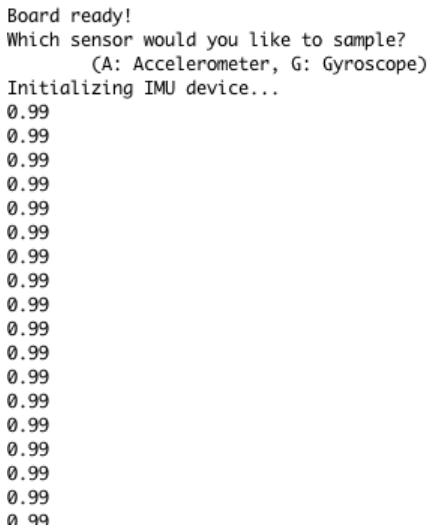

Figure 4: The accelerometer data printed to Serial monitor. No motion placed on accelerometer.

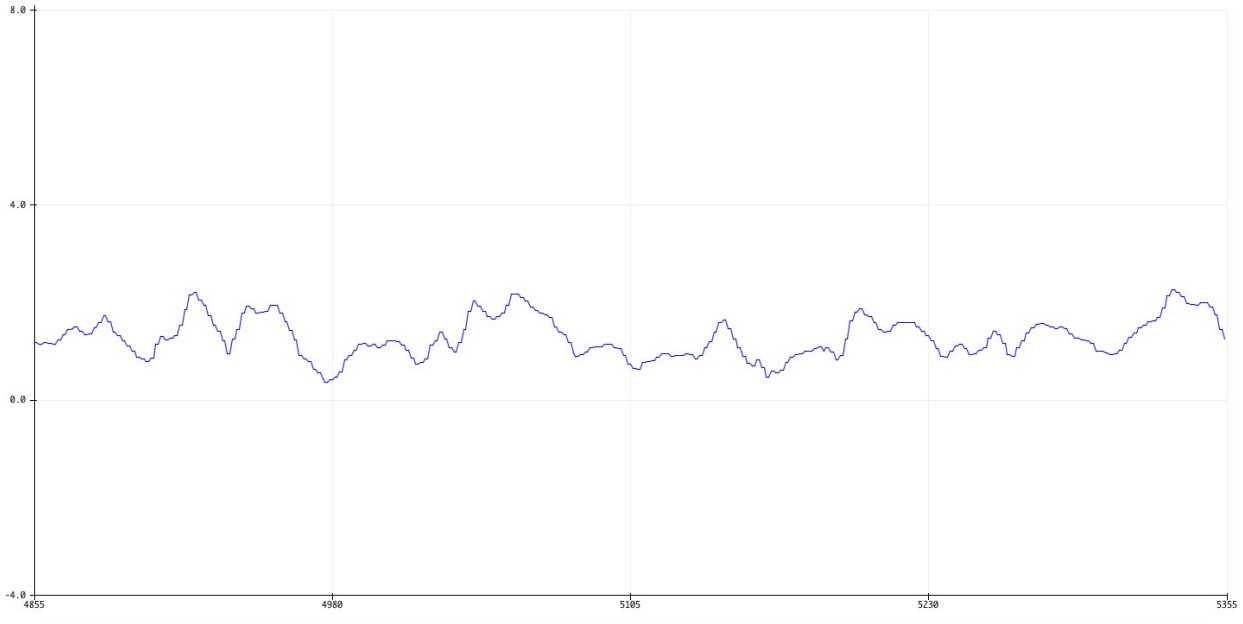

Figure 5: Accelerometer data plotted in Serial Plotter while being exposed to motion.

```
Board ready!
Which sensor would you like to sample?
       (A: Accelerometer, G: Gyroscope)
Initializing IMU device...
250.13
250.13
248.55
248.55
285.31
285.31
329.10
329.10
343.17
343.17
340.21
340.21
328.98
328.98
306.71
306.71
244.38
244.38
162.91
162.91
76.68
76.68
77.88
77.88
133.26
133.26
109.14
109.14
48.06
48.06
49.45
49.45
```

Figure 6: Gyroscope data printed in the Serial monitor while board experiences rotation.

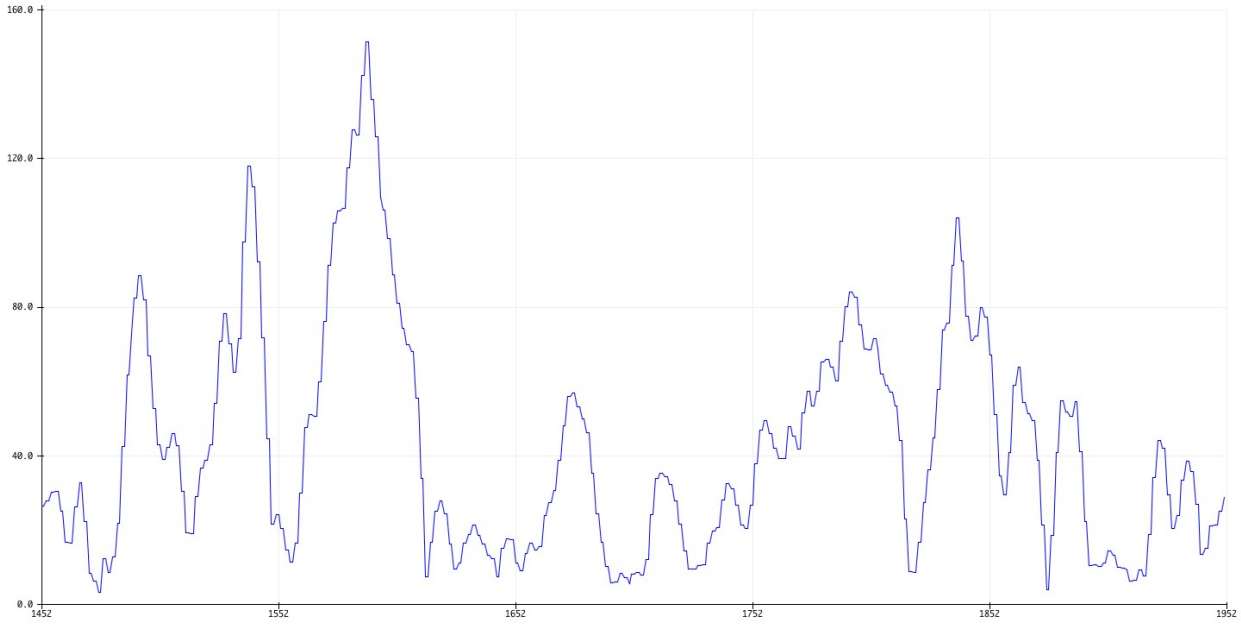

Figure 7: Gyroscope data plotted in the Serial plotter while board experiences rotation.

8. Objective 7: User Response Time

This objective is very similar to objective 5. However, unlike objective 5 which involves saving multiple samples and the time each sample was taken, objective 7 involves saving only one piece of information: a response time.

In the Arduino sketch we write (**ResponseTime**), we will need global variables startTime (unsigned long) to record the time when the sketch starts looking for a response, randNum (int), the randomly determined number of seconds until the board seeks a user response, a boolean checkGyro to determine whether to check the gyroscope or accelerometer, and three variables to record initial board position and three variables to record current board position to determine whether there has been movement. We leave loop() empty.

In setup(), we set the LED light to light up, begin Serial and set randomSeed to respond to analogRead(0) (in other words, randomly return numbers based on the voltage in analog pin 0). Next, we begin CurieIMU and set the accelerometer and gyro ranges (my example sets them to 3 and 250 respectively). Then, once Serial is ready, we prompt the user to input which sensor to sample: gyroscope or accelerometer and receive it. Once the input is received the randNum value shall be determined between 1 and 11, inclusively and exclusively, and the startTime shall be set with millis(). Immediately, 4 digitalWrite and 4 delay functions are added to the sketch to make it flash twice within 1 second. After that the setup function determines whether to read the initial values from the accelerometer or gyroscope and goes into a while loop that serves as the countdown to randNum where the conditional (millis() - startTime < randNum * 1000). In this loop, if the to be defined function detectMovement returns true the startTime is reset using millis(), restarting the countdown. Once the loop exits, the timer for periodically checking for a response starts with a period of 100 microseconds and attaching ISR function checkResponse. A new startTime is set to time the user and the LED turns on.

detectMovement determines whether to check the gyroscope or accelerometer based on user input by comparing (subtracting) the current position values to the original position values recorded in setup and determining if the difference is significant enough to return true.

checkResponse will use detectMovement (which returns boolean) to determine the response time and print it to Serial. Once the response time is printed, the timer is paused and the LED is turned off, effectively ending the program's processes.

Now the python script begins with importing serial and defining SERIAL_PORT

and SERIAL_RATE, as usual and opening the serial connection in the main function, reading and printing the prompt from the board, and writing ascii input to the serial connection. Depending on the user input, the fileName written to will be one of two names: “gyro_response_time.txt” or “accel_response_time.txt”. Upon reading the line that states the response time, as determined by a loop that seeks out a keyword that identifies the presence of a printed response time, the output from the board is printed and appended (written at the end) into the file, which is momentarily accessed with “with – as”. The serial connection then closes, ending the main function.

Now the text files attached to the lab was the result of running the Arduino sketch and Python script 20 times for each file, as the sketch and script only seek one response. However, if rewritten, the sketch and script can automatically loop to check for multiple responses.

9. **Objective 8: Data Analysis**

   Now here in the last objective, we will take the files saved and produced from objectives 5 and 7 and analyze the time periods.

10. Problems and Issues

In Objective 5, some issues occurred with the clock cycles when the printed strings indicated were too long. The time to print the strings initially proved to be too long, so the strings had to be reduced from their initial length.

In Objective 8,

Other than some logic error fixed in debugging, particularly some timing issues that were involved in the while loop of objective 7's Arduino sketch during its development, this programmer experienced no other issues in this lab.

11. Conclusion

This lab ultimately enabled us to use Arduino and Python, and taught us how to provide input and output to the two and between each other and how to process information between them.