ECE16: Rapid hardware and software design for interfacing with the world

Lab 2: Data Sampling & Characterization
Due Date: May 5, 2017 11:59pm

The goal of this lab is to introduce you to sensors, microcontroller based strategies for data sampling and Python based methods for data analysis. You will consider different strategies for data acquisition and communication to the PC, run experiments to test these strategies, and analyze/visualize the data from the experiments. The main consideration we will make in this laboratory is timing stability and communications.

Objective One - Implement sampling

Note: for now, **set the serial baud rate to 115200**.

In this first objective, as described in class, you will implement the first of two different methods that we will explore for sampling sensor data, polling. You will have the Arduino compute timing characteristics and report them back via the serial monitor.

Part 1:

Before trying this approach, write a small Arduino program to measure the amount of time it takes to read data with the microcontroller from three types of inputs to the Arduino: analog inputs, accelerometer data, gyroscope data. For the analog input, sample from three different analog input pins (they don't need to be connected to anything). For the accelerometer and gyroscope, measure the time for raw value reads. (Hint: use microsecond timing to test how long sensor reads take, if you'd like to check for consistency you could loop with a delay of your choice to repeat this measurement over and over again. Example output:

```
Microseconds for Accel read
41
Microseconds for analogRead
20
Microseconds for Gyro read
38
Microseconds for Accel read
41
Microseconds for analogRead
19
Microseconds for Gyro read
39
Microseconds for Accel read
41
Microseconds for analogRead
19
```

Related to this measurement:

1) Include the Arduino code in your lab writeup (only as part of Objective One, you do not need to repeat this program in the appendix).

2) In your write-up describe the results of this testing (i.e. the time delay measurement).
3) Describe the maximum rate (in hertz) at which you can read each of these sensor values with an Arduino based upon your testing. Describe why this maximum rate might not be achievable (Given the time it takes for the microcontroller to read a value, what is the maximum sample rate possible and what might keep your program from achieving this maximum rate. Note for the accelerometer and gyroscope, the IMU sensor also has an internal data rate, as described in class.)

Part 2:

Now that you have a measure of sampling time, write a program that uses a polling approach to read values from one of the three sensors (analog inputs from three pins, accelerometer, gyroscope). Have your program wait for a serial port connection and then provide the user with a prompt to enter a character. Wait for this character before initiating polling and use this character to select which of the three sensor types to poll from. Be sure to provide the user with clear instructions!

Each time a sensor is sampled, calculate the amount of time (in microseconds) since the last value was sampled. Write this elapsed time and the sampled sensor values to the serial port in a form that you can directly read from both the serial monitor (human readable) and into Python (as ASCII using `Serial.print()`). Keep all values in their raw (integer) format. Also include a sample counter that starts at 1 and is incremented each time a new analog value is sampled. You will need to add some additional formatting text to be able to separate each of the five pieces of data that are being written (sample number, three sample values, time elapsed since last sample). However, keep in mind that each character transmission takes time! So you may want to limit the number of additional formatting characters used.
Example output:

```
Type 1 to sample from the analog input, type 2, Gyro, 3, Accelerometer
1       914     784     770     1147
2       627     471     406     1132
3       398     292     231     1131
4       255     193     151     1131
5       170     140     116     1131
6       128     116     106     1131
7       105     107     102     1131
8       93      99      101     986
9       82      91      97      913
10      73      85      94      985
11      67      80      93      985
12      62      77      91      985
```

Specification for the Part 2 Arduino Program:

1) Require user input to begin polling and use this input to select the sensor type
2) Write five numbers -- Sample number (using your own counter variable), three sampled sensor values and the time difference in microseconds since the last read -- into one line. These numbers should be human readable and each sample should be on its own line (as shown in the example above).

When you first run the program, measure the maximum sampling rate using this polling strategy (Note: the loop time will include the time it takes for writing the data to the serial port). Include this measured maximum sampling rate in your lab report.

At this point you should be able to move the device around to get a sense for how the accelerometer and gyroscope work! Vary the value from the accelerometer by placing the Arduino in different orientations. Vary the value of the gyroscope by rotating the Arduino around different axes. (Nothing to turn in related playing with the sensors for this objective; however, building intuition for these readings will help later in the lab!)

Now, target a sample rate of 100 Hz. Calibrate your code to get as close as possible to this sampling rate (note that you might not be able to get the timing to be exact). Describe how you designed this program in your lab report and include the code in the appendix.

Objective Two - Implement interrupt based sampling

Note: for now, **set the serial baud rate to 115200**.

In this second objective, as described in class, you will implement interrupt based sampling using the microcontroller's built-in timer. Write a new Arduino program to accomplish this task. Have your program wait for user input to select the sensor type (as you did for Objective one) and once the character is entered enable the interrupt based sampling. Again, target 100 Hz sampling. As with Objective One, each time a value is sampled, calculate the amount of time (in microseconds) since the last analog value was sampled. Write this value, the sampled analog value, and a sample number to the serial port in a form that you can directly read from both the serial monitor and into Python. Use the same format that you used in Objective One.

Describe how you designed this program in your lab report and include the code in the appendix.

Objective Three - Log data to a file

Write a Python program to pull values from the serial port and write them to a file on disk. We will use this program to log data from the Arduino running code from Objectives 1 and 2. When the Python program is run, ask the user to input a filename and ask the user which sensor type they want to record from. Make sure the Python program is configured to provide the necessary input to the Arduino to start testing and file logging. Reformat the data being read by Python so that it is easier for a human to read:

1) Scale the units of the analog readings so that they are in volts measured by the microcontroller. (Note: Arduino 101 only read analog input from 0V to 3.3V instead of 0V to 5V on Arduino.cc reference. Make sure to map it correctly)
   Scale the units of the gyroscope measurements so that they are in °/second.

Scale the units of the accelerometer measurements so that they are in g.
2) Scale the units of elapsed time to milliseconds.
3) Improve the text formatting so that it is clear what is being measured for each sample number and so that the units of each measurement are apparent. Use your best judgment, as if you were to hand these files to a colleague and wanted them to be able to understand the files with minimal explanation. Also, do not throw away precision (e.g. although using units of volts, signal changes may be smaller than a volt). However, you may want to keep Objective Four in mind when setting this format!

To make sure your scaling for the accelerometer and gyroscope are correct, you can run a few sanity checks -- we'll describe these in lab session.

In writing up your lab report, describe this Python program in the objective section and include a legible screenshot of the sample of the file output when this program is used with either of the Arduino samplers, interrupt or polling (only the first 10 samples please!). Please provide sample output for all 3 sensor types:

- For the accelerometer readings, make sure your Arduino is flat on a table.
- For the gyroscope readings, rotate your Arduino around the axes of the cable connection. Include the full program in the appendix.
- For the analog sensor readings, please see the TA to get checked off for recording EMG data.

Objective Four - Summarize Performance

In this objective, you will measure average sampling rate and standard deviation by loading data from logged data files (using the program from objective three). To do this, write a program that loads logged data from a file and reports back the following:

1) Were any samples missing in the data?
2) How many samples were analyzed?
3) The mean time difference between samples.
4) The standard deviation of time difference between samples.
5) The maximum time difference between samples.
6) The minimum time difference between samples.

Again, use your best judgement in formatting what is written to screen, assuming that a colleague familiar with this project should be able to understand the result.

You should have run each sampling method for at least 5,000 samples. When writing your lab report, describe your Python program in this objective section and provide the output for data files from each of the two sampling methods. Are there any differences in the statistics of polling vs. interrupt driven sampling? If so, why?
Include the Python program source code in the appendix.

Objective Five - Maximum Sampling Rate & Baud Rate (Bonus Problem! Up to 10% of the lab points!)

Experiment with each of your samplers to find the maximum sampling rate achievable while maintaining a relatively consistent sample rate variability. Do not change any of the structure of the code or the outputs, just the timing constants (which will change the sampling rate). What happens if the interrupt is called for the second time, before the first one is finished? Describe the changes you made to each Arduino program, your testing strategy, answers and results.
Next drop the baud Rate to 9600 and repeat this test. Describe your results and any differences noted relative to your testing with a baud rate of 115200.

Objective Six – Finding Device Orientation by using IMU of Arduino Board

Mobile devices automatically determine whether the user is holding the device in portrait vs. landscape mode, and change the orientation of the display for better user experience. In this objective, you will write a sketch that will determine the orientation of your Arduino device by using IMU which contains three axis accelerometer and three axis gyroscope. You will need to determine four orientations as shown in Figure-1. Portrait-1 refers to USB connector being up and Portrait-2 refers to USB connector being down. Landscape-1 refers to USB connector pointing to left as you face the top part of Arduino, and Landscape-2 refers to USB connector pointing to right as you face the top part of the Arduino. *In this objective, you will implement interrupt based sampling to read from the relevant sensor, identify the orientation and print out a message in serial monitor every time the orientation changes*. For this objective, please provide the following:

- List the reasons why you have selected to use a certain sensor type for solving this problem
- Describe the strategy you have developed for determining the orientation
- Attach a snapshot of your output (you may take a picture of the Arduino in a certain position and the corresponding serial monitor output) for all 4 orientations.
- Attach the source code in the appendix

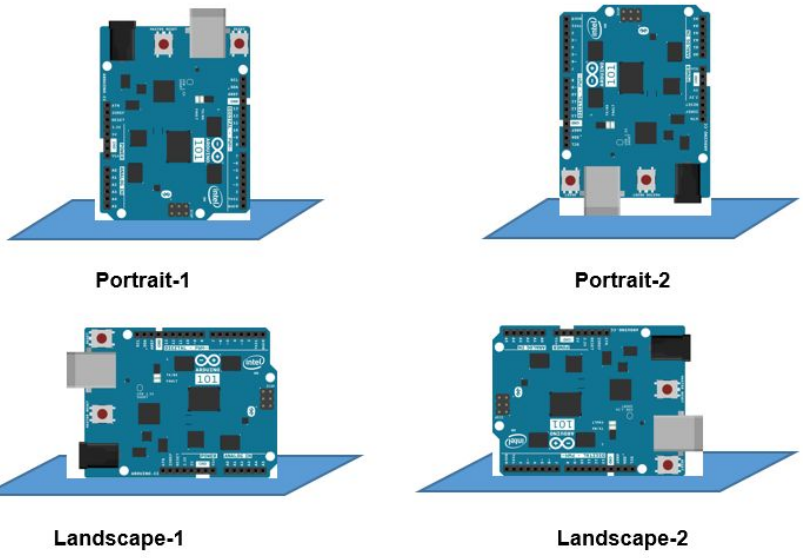

Figure-1 Portrait and Landscape Mode Examples