MECH 423 – Measurement and Instrumentation – 2024W
Lab 1: Data Acquisition using C#

Objectives

The goal of this lab is to create a data acquisition program using C#. The program will interface with a microprocessor on the MSP430EXP Board. The microprocessor is continuously sampling data from a 3-axis accelerometer at a rate of 100 Hz on each axis, and then transmits this data to the PC via a USB-serial port. Learning objectives of this lab include the following:

- Learn to write event-driven programs using Visual C#
- Become self-sufficient in learning new commands in C#
- Write code to acquire data from a serial port
- Write code to buffer and parse incoming data stream
- Develop a user interface to display and storing incoming data
- Develop a state machine for recognizing gestures from accelerometer data
- Showoff your creativity using C#

Logistics and Evaluation

This lab is done individually. Students are expected to learn to program in C# and complete the lab activities on your own using their own computers. Students begin by searching for Visual Studio 2022 and download the installer for the free "Community" version. Run the installer and select only the ".NET Desktop Development" option. After installing, start learning about Visual Studio, programming in C#, and Windows Forms starting from the embedded links. Please also follow these two video tutorial series: C# 101 and .NET Core 101. The video tutorials are for console programs rather than Windows Forms, but the materials are still highly relevant for this course. Additional tutorials can be found on MSDN, Google, or YouTube. Finally, there is the MECH 368 C# Tutorial (separate document available on Canvas) developed a few years ago. We have provided it as a reference. You do not need to hand-in the mini-projects for grades, but you can do them as extra practice.

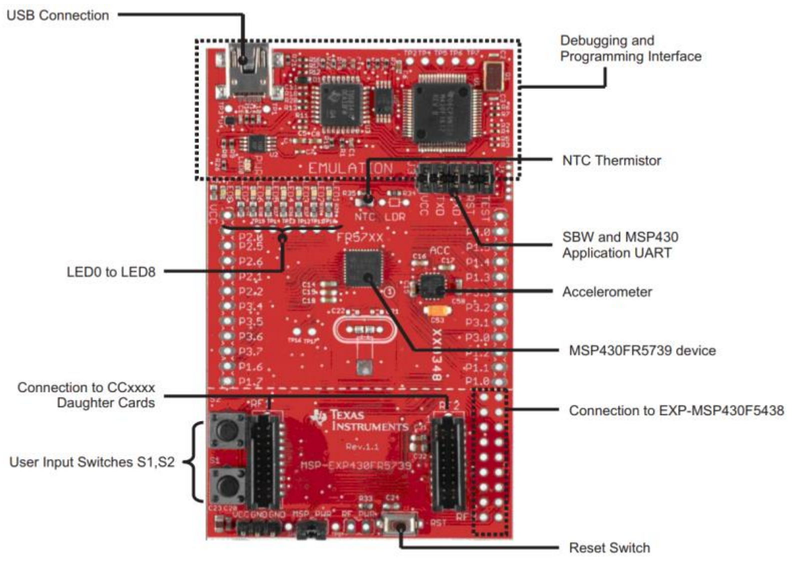
After setting up Visual Studios and learning from the online tutorials, begin working on the exercises. Course staff and TAs will be available during your scheduled lab session to help you. After completing each exercise, demonstrate your program to your TA, who will give a 1/0 grade for completing the exercises on time. The schedule for completing the exercises is in the syllabus.

The primary evaluation for this lab will be a lab exam, which integrates materials from the exercises together. An example lab exam is provided at the end of this document. The actual lab exam will be ~90% similar with some minor differences. You will have 90 minutes to write your program and demonstrate them for your TA. **Note: It is expected that you write all your C# code on your own in advance of your lab exam, and that you use the exam session to modify your code to fit the specifications for your exam session.**

This lab is worth **15%** of your final grade with **3/15** for completing the pre-exam exercises and **12/15** for the lab exam. 50% of the grade for the lab exam will be given for basic functionality and 50% will be given for creative elements.

Hardware and Interface

The hardware used in this lab is the MSP430EXP development board supplied by Texas Instruments (EXP board), which contains a microprocessor, a 3-axis accelerometer (axes noted in figure below), and additional circuitry for USB communications and power management. When you first plug the EXP Board into your USB port, the USB transceiver should automatically install a USB-Serial port and assign a COM port number. (Note: it is normal for some drivers to be installed twice.) If necessary, you can change the COM port number by going into the serial port properties menu. The UART on the microprocessor is configured to operate at 9600 baud, 8 data bits, one stop bit, no parity, and no flow control. Your C# software should be configured to match these parameters.


USB Connection
Debugging and Programming Interface
NTC Thermistor
SBW and MSP430 Application UART
LED0 to LED8
Accelerometer
MSP430FR5739 device
Connection to CCxxxx Daughter Cards
Connection to EXP-MSP430F5438
User Input Switches S1,S2
Reset Switch

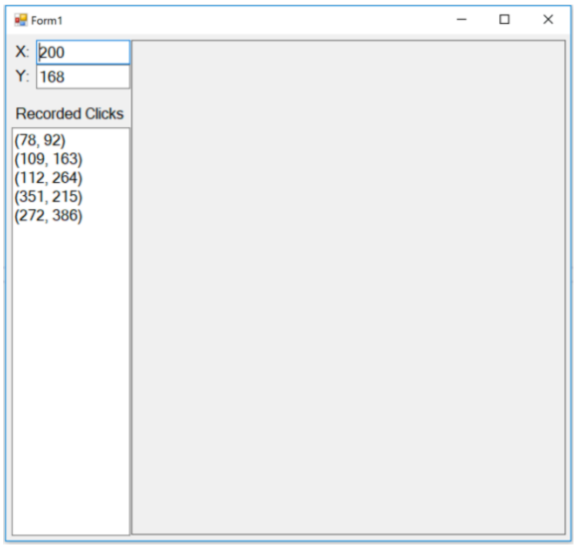
The firmware EXP board is configured to operate in several modes. The mode relevant to Lab 1 can be initiated by transmitting the character “A” to the serial port. In this mode, the microprocessor samples voltage readings from the 3-axis accelerometer and uses an on-board analog-to-digital converter to convert these voltages into an 8-bit digital value between 0 and 255. The digitized acceleration data is transmitted to your PC in 8-bit packets using a serial protocol managed by a Universal Asynchronous Receiver/Transmitter (UART) controller, which also embedded in the microprocessor. The data bits are transmitted at a speed of 9600 bits per second (or baud). The microprocessor has been programmed to output data in the following sequence:

Start byte	Data byte #1	Data byte #2	Data byte #3
255	X-acceleration	Y-acceleration	Z-acceleration

A start byte is necessary because the microprocessor will transmit data regardless of whether a PC program is available to respond to it. Therefore, if the microprocessor is powered before the C# program is activated, it will be impossible to distinguish Data byte #1 from Data byte #2. The number 255 has been designated as the start byte. Since each packet can only contain a number from 0 to 255, using 255 as the start byte means that this value cannot be used in the data byte. In fact, any acceleration data with the value of 255 is automatically converted to 254 by the microprocessor.

Exercise 1: Warm up – track mouse position in a PictureBox

1. Set up a form with two Labels, two Textboxes, and one Picturebox controls as shown below. Change the appearance of the controls to make them look nice. Change the picturebox border to FixedSingle.

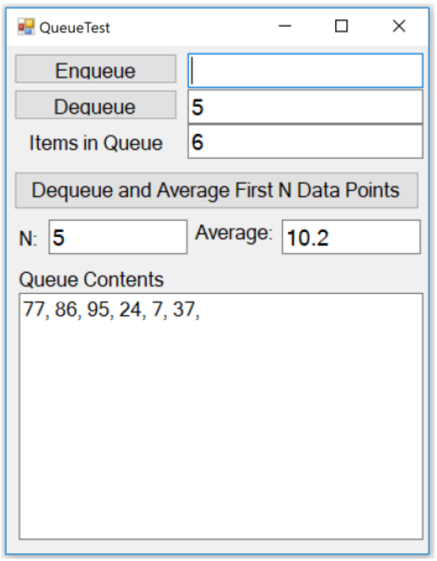


200
168
(78, 92)
(109, 163)
(112, 264)
(351, 215)
(272, 386)

2. Set up the MouseMove event handler on the PictureBox. When MouseMove is called, use the variable e of the MouseEventArgs data type passed into the event handler to acquire the (X,Y) mouse pointer position in the PictureBox and show them in the appropriate textboxes.
3. Add a Label and a Textbox. Enable multi-line in the Textbox and use it to display "Recorded Clicks".
4. Set up the MouseClick (Note: not the same as Click) event handler on the PictureBox. When MouseClick is called, acquire the (X,Y) mouse pointer position. Construct a string consisting of the (X,Y) mouse coordinate and an end-of-line character. Add this string to the Textbox using the AppendText method.

Exercise 2: Working with queues

1. Set up a form with buttons, textboxes, and labels as shown below. Give appropriate human-readable names to each.



Dequeue: 5
Items in Queue: 6
N: 5
Average: 10.2
77, 86, 95, 24, 7, 37,

2. Initialize a queue of Int32 in the Form class using:

```
        Queue<Int32> dataQueue = new Queue<Int32>();
```

3. Add the Enqueue button event handler to enqueue values from the Textbox into dataQueue. Use the Convert.ToInt32 to convert strings into an Int32.
4. Add the Dequeue button event handler to dequeue values from dataQueue. When the user attempts to dequeue from an empty queue show a MessageBox to provide the user with an error message.
5. Add a function called "UpdateQueue" to display the number of items in queue in a Textbox. The UpdateQueue function should also display the contents of the queue in the large multi-line Textbox. Use a Foreach statement to loop through the items in the queue and display each one to the Textbox using the AppendText method. The items should not be dequeued for this purpose.
6. Set up a timer with an interval of 100 ms. Enable the time during form load. During the TimerTick event, call UpdateQueue.
7. Add an event handler for "Dequeue and Average First N Data Points". First check if there are sufficient data points in the queue. If there's insufficient data, show a MessageBox to give the user an error message. If there is sufficient data, then dequeue N data points and calculate the average. Hints: dequeue one number at a time using a for loop. Be careful about data types and lost precision when calculating. Show the calculated average is correct.

Exercise 3: Switch to ConcurrentQueue

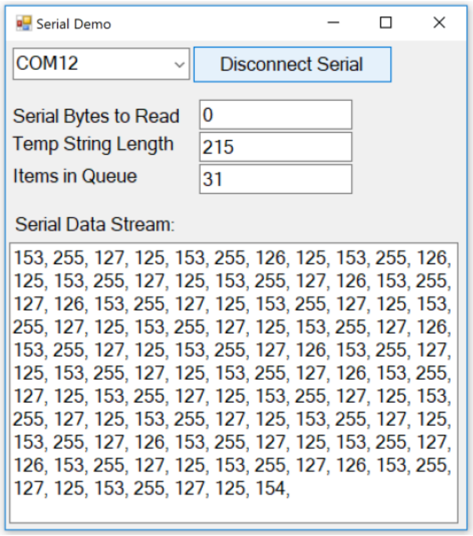
1. Make a copy of your exercise 2 code.
2. Change the type for dataQueue from Queue<Int32> to ConcurrentQueue<Int32>. Add the following using statement:

```
    using System.Collections.Concurrent;
```

3. ConcurrentQueue does not have a dequeue method, but uses TryDequeue instead. Change your code appropriately to dequeue using TryDequeue while anticipating that TryDequeue may fail.

Exercise 4: Receive data from the serial port

1. Set up a form with buttons, textboxes, and labels as shown below.



Serial Bytes to Read 0
Temp String Length 215
Items in Queue 31

Serial Data Stream:

```
153, 255, 127, 125, 153, 255, 126, 125, 153, 255, 126,
125, 153, 255, 127, 125, 153, 255, 127, 126, 153, 255,
127, 126, 153, 255, 127, 125, 153, 255, 127, 125, 153,
255, 127, 125, 153, 255, 127, 125, 153, 255, 127, 126,
153, 255, 127, 125, 153, 255, 127, 126, 153, 255, 127,
125, 153, 255, 127, 125, 153, 255, 127, 126, 153, 255,
127, 125, 153, 255, 127, 125, 153, 255, 127, 125, 153,
255, 127, 125, 153, 255, 127, 125, 153, 255, 127, 125,
153, 255, 127, 126, 153, 255, 127, 125, 153, 255, 127,
126, 153, 255, 127, 125, 153, 255, 127, 126, 153, 255,
127, 125, 153, 255, 127, 125, 154,
```

2. Add a Serialport object to the form. Configure it as 9600 8N1 (Baud rate = 9600, data bits = 8, flow control/handshake = false, stop bits = 1)

3. Plug in the MSP430EXP board to a USB port. A USB serial port should automatically install. Figure out the COM port number for the MSP430EXP.
4. In the FormLoad event handler, configure the Serialport object to the correct COM port.
5. Optionally, acquire the available COM ports and deposit them in a ComboBox using the following code

```
comboBoxCOMPorts.Items.Clear();
comboBoxCOMPorts.Items.AddRange(System.IO.Ports.SerialPort.GetPortNames());
if (comboBoxCOMPorts.Items.Count == 0)
    comboBoxCOMPorts.Text = "No COM ports!";
else
    comboBoxCOMPorts.SelectedIndex = 0;
```

   This code can be run during FormLoad or other times when there may be a change in the COM ports. In the ComboBox SelectedIndexChanged event handler, acquire the COM port from the ComboBox and use it to configure the COM port on the Serialport object.
6. Set up a button click event handler to open the serial port (after it is properly configured). Once the serial port is opened, the accelerometer data is automatically enabled as output.
7. Initialize a string in the form class called serialDataString to temporarily hold incoming serial data.
8. Set up a Serialport DataReceived event handler. In this function, first determine the number of BytesToRead in the serial buffer. Write a while loop to read the bytes, one at a time, from the serial buffer. Convert each byte to a string and append it to the serialDataString with “,“ and “ “ characters. This code is being provided below as an example:

```
int newByte = 0;
int bytesToRead;

bytesToRead = serialPort1.BytesToRead;
while (bytesToRead!=0)
{
    newByte = serialPort1.ReadByte();
    serialDataString = serialDataString + newByte.ToString() + ", ";
    bytesToRead = serialPort1.BytesToRead;
}
```

9. Add a timer to the form. Enable the timer in the FormLoad. Set up a TimerTick event handler to show the number of bytes in the serial buffer and transfer data from serialDataString to the Serial Data Stream Textbox using the following code:

```
if (serialPort1.IsOpen)
    textBoxBytesToRead.Text = serialPort1.BytesToRead.ToString();
textBoxTempStringLength.Text = serialDataString.Length.ToString();
textBoxSerialDataStream.AppendText(serialDataString);
serialDataString = "";
```

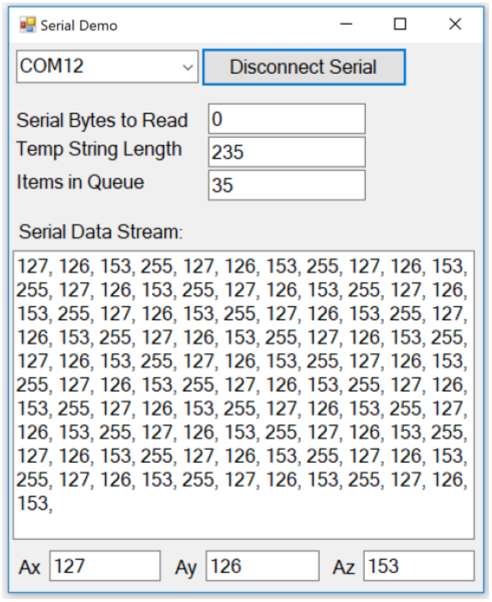
10. **Checkpoint:** Run your code and show that you are getting the expected data stream. Change the timer interval and show how it changes the length of serialDataString. An interval of 100 ms should give a serialDataString length of ~50, while an interval of 50 ms should give a length of ~25.
    a. Note: Your serialDataString length may differ. Some code implementations output a serialDataString length of ~220 for a 100 ms interval (and ~120 length for 50 ms). Both speeds are acceptable. Ask your TA if you have problems or questions.
11. Initialize a ConcurrentQueue of Int32 in the Form class using:

```
ConcurrentQueue<Int32> dataQueue = new ConcurrentQueue<Int32>();
```

12. In the Serialport DataReceived event handler, store each new data byte in a ConcurrentQueue instead of a string.
13. In the TimerTick event handler, set up a loop to dequeue the data from the ConcurrentQueue one byte at a time. Append each dequeue'ed byte to the Serial Data Stream Textbox. Show the number of items in queue in a Textbox.
14. **Checkpoint:** A timer interval of 100 ms should give a ConcurrentQueue size of ≤10 (or 30-50 if your serialDataString length is ~220).

Exercise 5: Parse accelerometer data stream and perform simple data analysis

1. Add Labels and Textboxes to the exercise 4 Form as below.



Serial Bytes to Read: 0
Temp String Length: 235
Items in Queue: 35

Serial Data Stream:

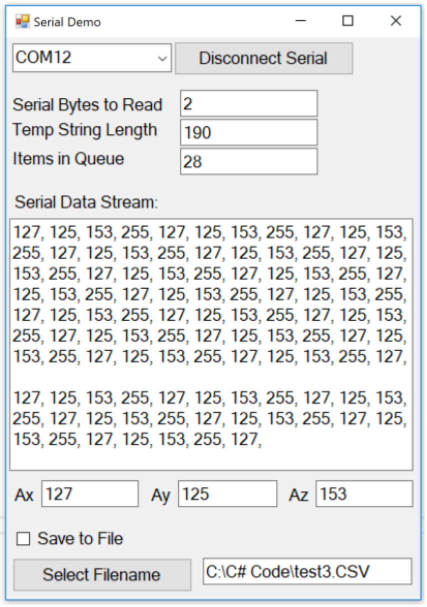
```
127, 126, 153, 255, 127, 126, 153, 255, 127, 126, 153,
255, 127, 126, 153, 255, 127, 126, 153, 255, 127, 126,
153, 255, 127, 126, 153, 255, 127, 126, 153, 255, 127,
126, 153, 255, 127, 126, 153, 255, 127, 126, 153, 255,
127, 126, 153, 255, 127, 126, 153, 255, 127, 126, 153,
255, 127, 126, 153, 255, 127, 126, 153, 255, 127, 126,
153, 255, 127, 126, 153, 255, 127, 126, 153, 255, 127,
126, 153, 255, 127, 126, 153, 255, 127, 126, 153, 255,
127, 126, 153, 255, 127, 126, 153, 255, 127, 126, 153,
255, 127, 126, 153, 255, 127, 126, 153, 255, 127, 126,
153,
```

Ax 127 Ay 126 Az 153

2. Add code in the TimerTick event handler to parse the accelerometer data stream. First check if the incoming byte is 255. If it is not a 255, do nothing and check the next data point. If a 255 is received, then the next byte is an X-axis acceleration (Ax). Display this value in the Textbox for Ax and enqueue it in a ConcurrentQueue. Repeat for Ay and Az. Add a state variable in the form class to keep track of whether the next byte is Ax, Ay, or Az for subsequent calls to the TimerTick event handler.
3. Add code in the TimerTick event handler to determine the orientation of the MSP430EXP PCB based on the values of Ax, Ay, and Az. Indicate orientation in a Textbox.

Exercise 6: Save accelerometer data stream to a CSV file

1. Add the following buttons and Textboxes to the form in Exercise 5:



Serial Bytes to Read 2
Temp String Length 190
Items in Queue 28

Serial Data Stream:

```
127, 125, 153, 255, 127, 125, 153, 255, 127, 125, 153,
255, 127, 125, 153, 255, 127, 125, 153, 255, 127, 125,
153, 255, 127, 125, 153, 255, 127, 125, 153, 255, 127,
125, 153, 255, 127, 125, 153, 255, 127, 125, 153, 255,
127, 125, 153, 255, 127, 125, 153, 255, 127, 125, 153,
255, 127, 125, 153, 255, 127, 125, 153, 255, 127, 125,
153, 255, 127, 125, 153, 255, 127, 125, 153, 255, 127,

127, 125, 153, 255, 127, 125, 153, 255, 127, 125, 153,
255, 127, 125, 153, 255, 127, 125, 153, 255, 127, 125,
153, 255, 127, 125, 153, 255, 127,
```

Ax 127 Ay 125 Az 153

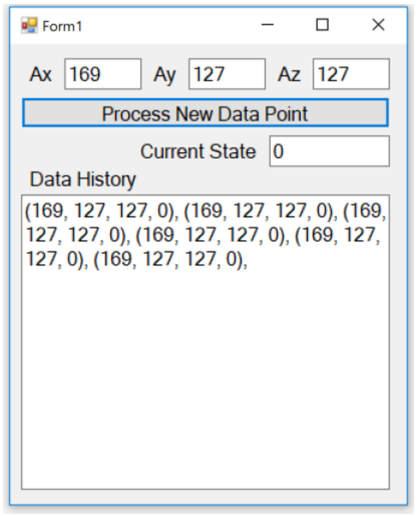
☐ Save to File

C:\C# Code\test3.CSV

2. Initialize a new StreamWriter object
```
StreamWriter outputFile;
```
3. Open a new file for writing by call the StreamWriter constructor with the filename string as parameter
```
outputFile = new StreamWriter(textBoxFileName.Text);
```
4. If you want to be fancy about it, initiate a SaveFileDialog box to allow the user to choose the location to save the file and enter the filename.
5. Add a recording feature to your program that write the parsed data points to a file using the outputFile.Write method. Convert the data values to strings and separate them with a comma. After each set of Ax, Ay, Az, add a time stamp and end-of-line character (\n). Check the file to make sure it is correctly formatted and can be imported into Excel.
6. Record accelerometer data for the three gestures in the example lab exam. Record several examples for each gesture. Plot the data in Excel.

Exercise 7: State machine testing program

1. Make a simple state machine testing program as below



Ax 169 Ay 127 Az 127

Current State 0

Data History

```
(169, 127, 127, 0), (169, 127, 127, 0), (169,
127, 127, 0), (169, 127, 127, 0), (169, 127,
127, 0), (169, 127, 127, 0),
```

2. The “Process New Data Point” button feeds the current values of Ax, Ay, Az into the state machine
3. After processing, the current state is displayed in a Textbox.
4. Data history records the values of Ax, Ay, Az, and the resulting state.
5. Use this program to test your proposed state machines for gesture recognition.

Exercise 8: Develop gesture recognition algorithms

Now you are ready to develop gesture recognition algorithms for the practice lab exam. Note that each gesture is a sequence of movements in the stated directions. For example, (Z+, X+) means the first movement is positive in the Z axis, which is followed by a second movement in the X axis. For each gesture in the practice lab exam, do the following:

1. Record the gesture by saving your data to a .CSV file.
2. Identify unique features associated with each gesture and design a state machine to recognize these features.
3. Write code to implement this state machine.
4. Test the state machine by entering test data into exercise 7. Check the data history to ensure the correct intermediate and final states are reached.
5. Indicate the detected gesture using text and/or graphics output
6. Transfer your state machine to analyze the live data stream. For each gesture, confirm from the history of the detected states to show the correct intermediate and final states are reached. Iterate your state machine code as necessary.
7. Practice for the lab exam by trying to detect some different gestures.

Exercise 9: Creativity with C#

1. Develop a piece of interactive software of your own creation. You can incorporate use of the accelerometer data (e.g. display, analyze, graph), or not. You can also incorporate available code written by others from repositories. Please let your TA know which code you incorporated. Here are some ideas:
    a. Make your own version of MS Paint
    b. Interactive animation (e.g. birthday card or screen saver)
    c. Snake game or similar obstacle course game
    d. Hockey goalie game using the accelerometer as the controller
    e. Glider game using the accelerometer as controller
    f. Use the accelerometer as a musical instrument
    g. Integrate accelerometer data to measure velocity or position (difficult!)
    h. Use the EXP board to air write letters or symbols
    i. Use the EXP board for a game or interactive contest (e.g. balancing contest)
2. Use Zoom to record a screen-capture a demo of your code. Use the camera to capture your use of the accelerometer.

MECH 423 – Mechatronic Product Design
Lab 1: Data Acquisition using Visual C#
Example Lab Exam

Instructions

- You have 90 minutes to complete this lab exam and demonstrate your work for a TA to grade.
- You will work individually without getting any help from anyone else
- This exam is "open computer". You can use any notes and any code that you wrote previously
- When ready, demonstrate the mandatory elements of your lab for a TA. For the gesture recognition portion, your hands should only be holding the EXP board and not touching your computer or mouse.
- Demonstrate the creative elements of your lab for a TA. The creative elements can integrate the mandatory elements but is not required.

Mandatory Elements (50%)

You have been asked to develop a hand-held controller for a Street Fighter style game. The player will be using a series of acceleration gestures on the controller to generate punch and kick combinations. First, develop a data acquisition and processing program using C# to satisfy the following functional requirements:

Functional Requirement	**Grade**
1. Display acceleration in X, Y, and Z	3
2. Display serial buffer size and queue size	1
3. Display the average of the last 100 data points in X, Y, and Z	3
4. Indicate EXP board orientation (X+, X-, Y+, Y-, Z+, Z-)	3
5. Gesture #1 recognized	5
6. Gesture #2 recognized	5
7. Gesture #3 recognized	5
Total	**25**

Develop a gesture recognition algorithm to recognize the gestures below. Provide a live data processing history textbox to show the correct intermediate and final states are reached. When a gesture is detected, the program should provide an indication (e.g. textbox, animation, sound etc) that should persist for 1s. Show that your software can detect each of the three gestures 5 times in a row using a total of 6 tries.

Table 1: Example Table of Gestures

Gesture	**Acceleration Sequence**
1. Simple punch	+X
2. High punch	+Z, +X
3. Right-hook	+X, +Y, +Z

***Note: Greyed areas will be changed in the actual lab exam

Creative Elements (50%)

Develop a piece of interactive software of your own creation as described in Lab 1 Exercise 9. You can incorporate use of the accelerometer data (e.g. display, analyze, graph), or not. There is no set agenda. You are free to make anything you want. Your work will be scored based on functionality, creativity, difficulty, and sensibility.

Perform a demonstration of your code to a TA. Upload your demonstration video to Canvas.