EMG Application
Implementation notes and User Guide

Computer Engineering
Final Year Project

24/03/2014-DRAFT

David John Smith

Contents

System Overview

The overall system is broken into four main sections (As seen in Figure 1):

1. The sensors which gather the various types of input data.
2. The host platform which gathers sensor data and performs any necessary processing and local storage.
3. The Server which receives data from the host platform and stores it in the cloud for remote retrieval and visualisation.
4. End devices which can connect to web applications for data visualisation.

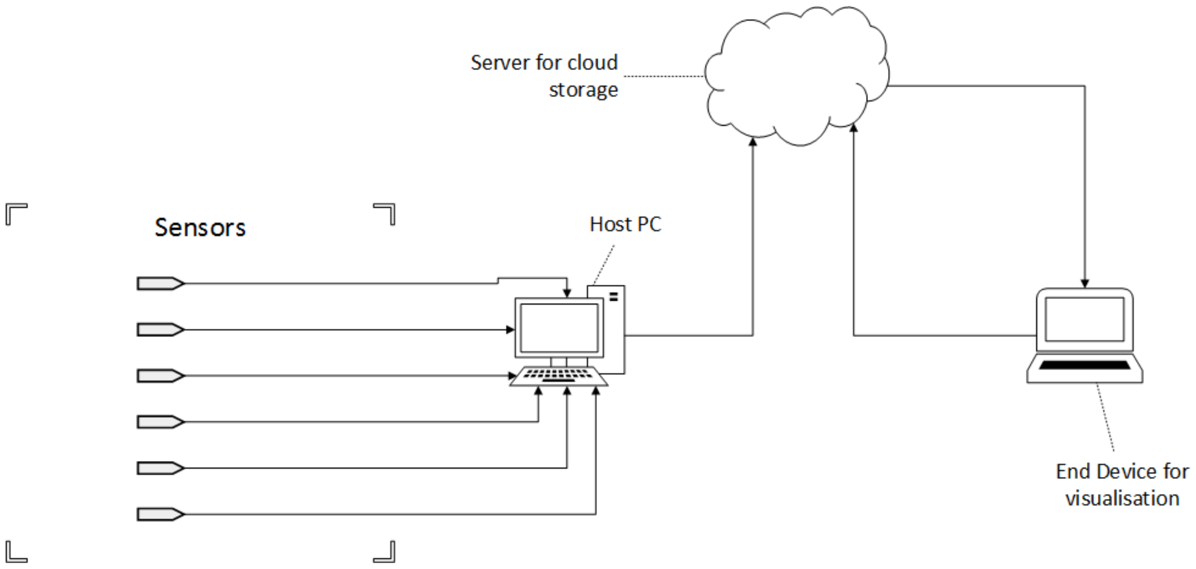


Figure 1: High level system diagram

This document is regarding the part of the system which provides the functionality of sections 1 and 2 above for EMG sensors and also sends the data to the server mentioned in section 3. This system consists of a program which was written to gather and process data from Shimmer EMG sensors before finally passing the data onto the server at a suitable rate. It interacts with the Shimmer EMG sensors through the Java API which they were supplied with. This API seems to have been intended for use on the Android platform but it does also include some classes for use on PC. It is uncertain how long and well they will support the PC section of the API but this shouldn't be an issue as we are only intending to use the Shimmer devices until in-house sensors are ready.

The system performs the processing which is inherently necessary when dealing with EMG data. This processing consists of the filtering of noise from the raw EMG data using Band Stop filters and High Pass filters. The data required further processing as the server required it to be rectified into a power signal from which relative changes in power would be readily apparent. This was performed using a rolling window Root Mean Square (RMS) filter. All of these filters were implemented in Java on the host system. The host receives data from the sensors with a sampling frequency of 1024 Hz which is far too high to send to the server. Therefore processed data is only selectively sent to give a sampling rate of approximately 200 Hz; this is adjustable before runtime.

Presently the system needs to be able to collect and process six different EMG channels. As each Shimmer device has two channels the software is required to connect to three separate devices. As each device transmits two channels (each with data at 1024 Hz) there have been issues with receiving Bluetooth devices not being able to handle the high throughput. This has proven to be an issue with USB 2.0 Bluetooth dongles and certain integrated Bluetooth receivers. However, it seems that the majority of integrated Bluetooth receivers running native firmware are more than capable to dealing with the high throughput of data.

Design and Implementation

This system was designed to be as flexible as possible with the number of sensors and the data which is gathered and processed from them. Below in Figure 2 the logical layout of the system can be seen along with how each class was planned to work together.

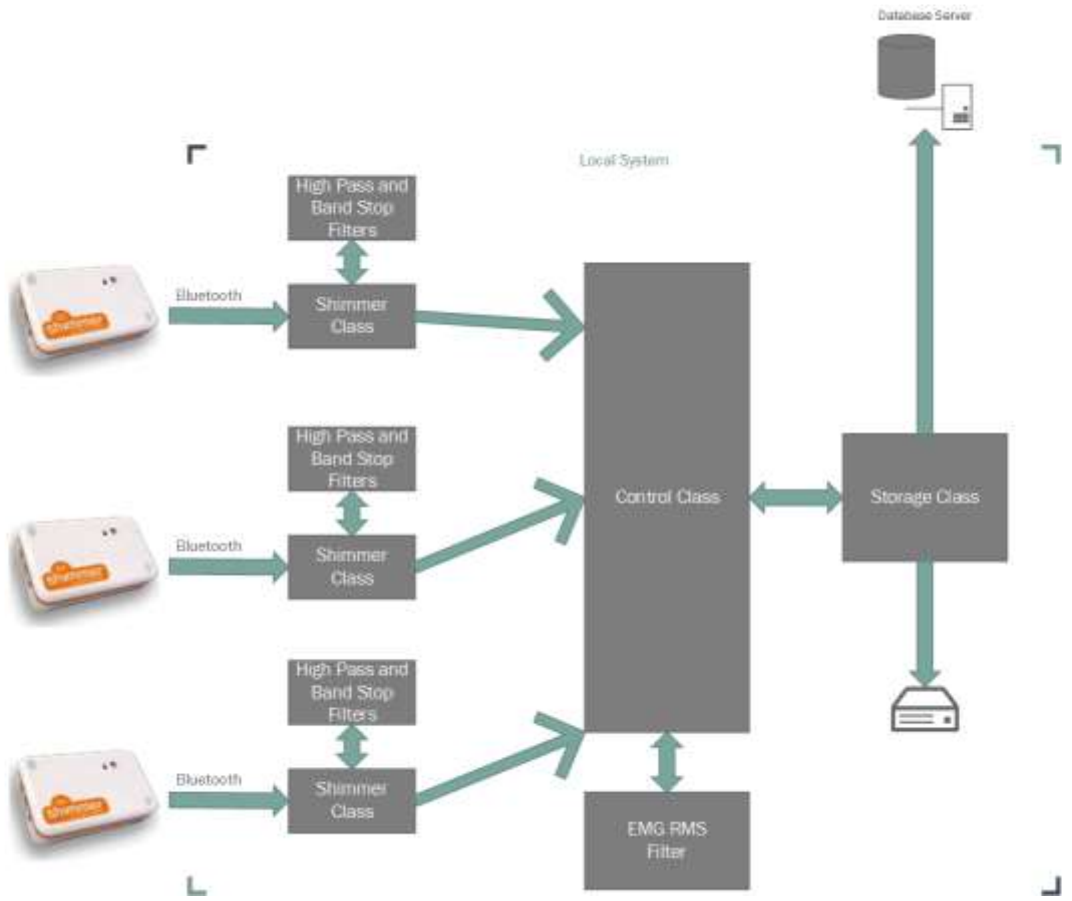


Figure 2: EMG System design

The EMG system has a number of different components which operate over a number of different threads in the host PC and the class diagram can be seen below in Figure 3.

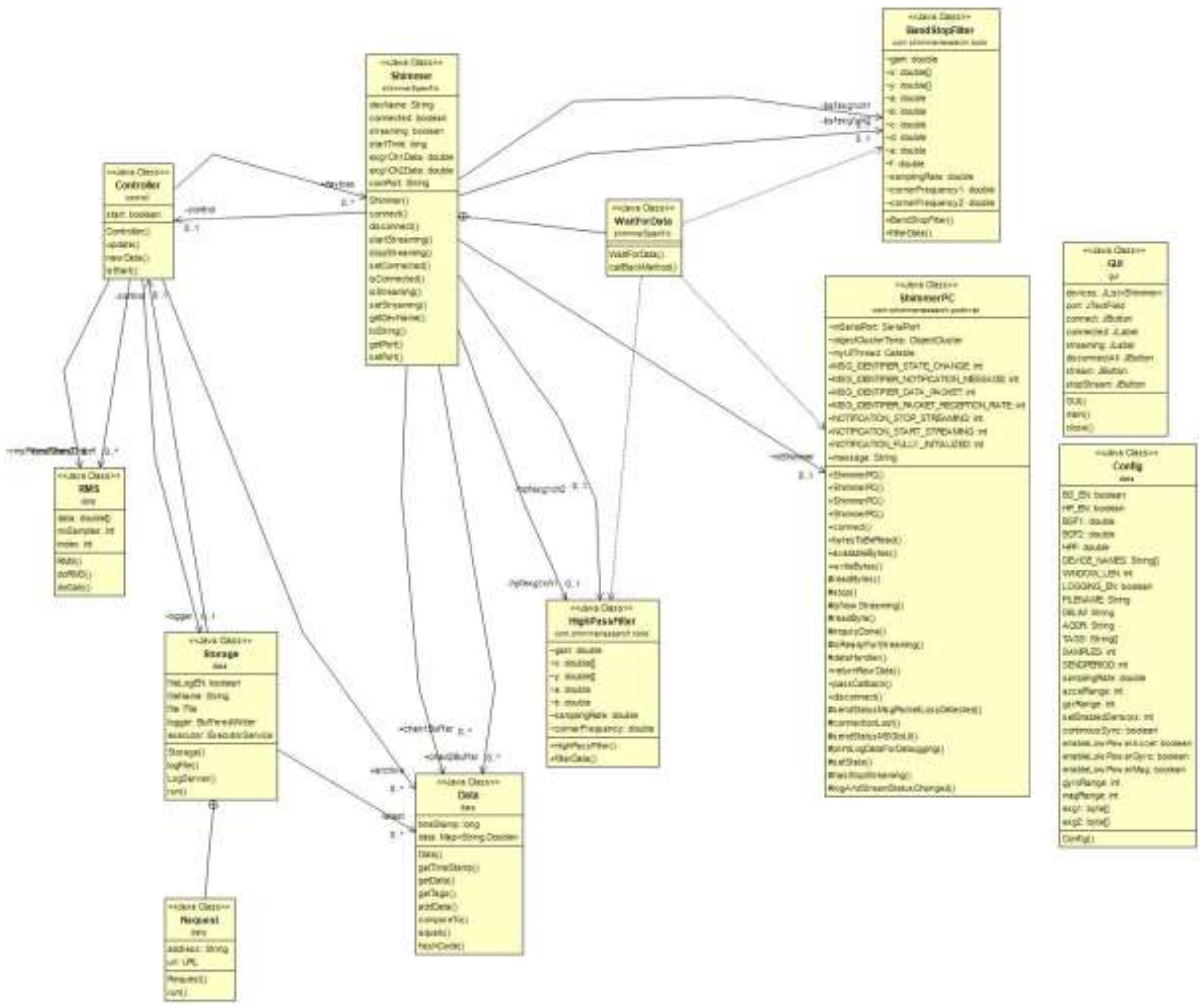


Figure 3: Class diagram

Shimmer API

As previously mentioned Shimmer provided an API through which it was possible to communicate with the sensors and this API was in the form of the *ShimmerPC* class. Each sensor required a single instance of this class through which further low level software is used to interface with the sensors. It would appear that the lower level classes which the API uses are multithreading which means that multiple instances of *ShimmerPC* can receive data at the same time.

All data is transferred from the API using shimmers own *ObjectCluster* data structure which consists of a *PropertyCluster*. A *PropertyCluster* is a MutliMap where each key represented a property (e.g. EMG) and each value the *FormatCluster* of said property. *FormatCluster* is another Shimmer object which holds the format (e.g. Calibrated), units and data of the property which it refers to. This is shown below in Figure 4 from the Shimmer API documentation. Further information about these classes can be found in Shimmers API documentation.

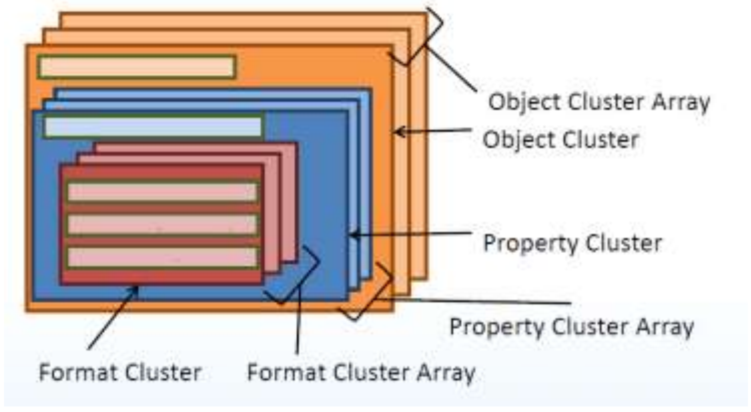


Figure 4: Shimmers Data structure

Shimmer Class

To improve functionality the *ShimmerPC* object was wrapped in a newly written class called *Shimmer* which provides methods to interact with the device and receives then filters raw data before repackaging it in another object. This class implements High Pass filters and Band Pass filters to remove noise from the raw EMG data. These filters were also provided in the Shimmer API. The *Shimmer* class receives data and command replies from the API through a CallBack method and then properly handles whatever it receives. For example it will filter and add new data to its channel's buffer or it will change the devices state if it receives a "Stopped streaming" reply. The *Shimmer* class also extends the *Observable* abstract class which allows it to notify other classes of changes within itself; this functionality is used to inform the *Controller* class when there is data in each channel's buffer.

Data Object

Unfortunately the *ObjectCluster* data structure which the Shimmer API provided was not perfectly suitable to the archiving needs of multiple devices and therefore another similar data element was created which stores multiple data points along with a time stamp which they refer to. This data element is the *Data object* and has a natural order which is decided by its time stamp; allowing it to be easily used with existing data structures. The data structure which the *Data object* is stored in is a *ConcurrentSkipListSet.* This was chosen as it allowed concurrent access to the data while also being a sorted Set which means that duplicate entries are not possible. Therefore, as each *Data object* is sorted based on its timestamp it is trivial to check for other entries with that timestamp and then combine their data when necessary. This is especially important due to the fact that the data reception is not synchronised over different sensors; therefore identical timestamps will be received at different times.

Data Synchronisation

During the design stages of this system there was a data synchronisation issue that was identified as being particularly important. The problem stated that it will be very difficult to identify the data gathered at any specific instant in time due to the delay between measurement of data and reception/time stamping on the host PC. This problem arose because of the lack of a Real Time Clock on the sensors meaning that the UNIX timestamp needs to be added on reception. However, this proved to be a bigger issue for other sensors as it was possible to remove most of the transmission delay in software for the EMG sensors. This was made possible due to the EMG sensors sending a "time since streaming" timestamp along with data; therefore it was trivial to log the time streaming started then add the time since started timestamp to get the UNIX timestamp for the data. This cut the transmission delay error from a variable length of time for each data point to a single length of time between sending the start streaming command and receiving the “streaming started” response.

Controller class

As the name suggests this class controls most of the programs functionality as well as initialising each separate component. It maintains a list of *Shimmer* objects, a number of RMS filters for each channel of data, the concurrently accessible Data Archive and the *Storage* class which deals with logging. The class starts by initialising each of these components before waiting for data for processing. Once it is notified of data it will retrieve this from the respective *Shimmer* object before putting it through a moving window RMS filter and then storing in the central archive.

When initialising the *Storage* class the *Controller* class passes a reference of itself so that the Storage class can access a version of the central archive. The *Storage* class is then started in its own thread to let it read from the central archive while the *Controller* class is still adding new data. This is why it was important that the central archive was a data structure which facilitated concurrent access.

Furthermore *Controller* implements the *Observer* interface which allows the *Shimmer* class to notify it when there is data waiting in its buffers. This is important as each Shimmer class may receive data at different points in time and we want to retrieve this data as efficiently as possible.

RMS Object

The *RMS* class contains the implementation of the moving window RMS Filter. It performs this filter by filling up a queue with sequential data points. Once the queue has reached a predetermined length the following equation will be carried out giving the RMS of the window length n:

$$x_{rms} = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + \cdots + x_n^2)}$$

After this any new data points will be added onto the end of the queue after the first element has been removed before finding out the new RMS value. This way a moving window is produced giving an RMS value for a variable window length.

Storage Class

The *Storage* class deals with all logging of data for the system both to a local file and to the server where it can be remotely accessed. On initialisation the class is given a reference to the *Controller* class so that it can access the central archive. If logging to file is enabled then *Storage* will attempt to set up and access a local file for writing data to.

Once the *Storage* class is running it will constantly poll the central archive in order to see if there has been any new data added. When new data is found it will also ensure that the data is old enough to send as we do not want to prematurely send a single sensors data before we receive data from the other sensors. If data is suitable for sending it will be removed from the central archive and all data before this will be cleared. This has to be done in order to stop the Java Virtual machine from running out of available memory.

Once the class has acquired data suitable for logging it will pass the data onto helper methods which will parse it into a suitable string and then write it to file. This data is also passed to another class for sending to the server. The actual transmission to the server needs to be in a separate thread due to the time taken to create the connection and send the request. If this was done in the same thread of the *Storage* class then it would never be able to keep up with the new data being added into the central archive.

Config Class

The *Config* class is just a helper class which declares a number of static variables which are used throughout the rest of the program. These have been declared here so that any changes to the program can be made from one single place. By changing the declarations here it is possible to the following:

- Filter Settings:
    - Toggle the Band Stop Filter
    - Modify the corner frequencies of the Band Stop Filter
    - Toggle the High Pass Filter
    - Modify the corner frequencies of the High Pass Filter
    - Increase the window length for the RMS Filter
- Device Settings:
    - Rename and change the number of devices
- Logging Settings:
    - Toggle logging
    - Modify the list of data points which are logged
    - Local logging:
        - Modify the name of the log file
        - Modify the delimiter used in the log file
    - Remote logging:
        - Modify the address of the remote logging server
        - Modify the Number of samples sent in a single request
        - Modify the time between requests

User Guide

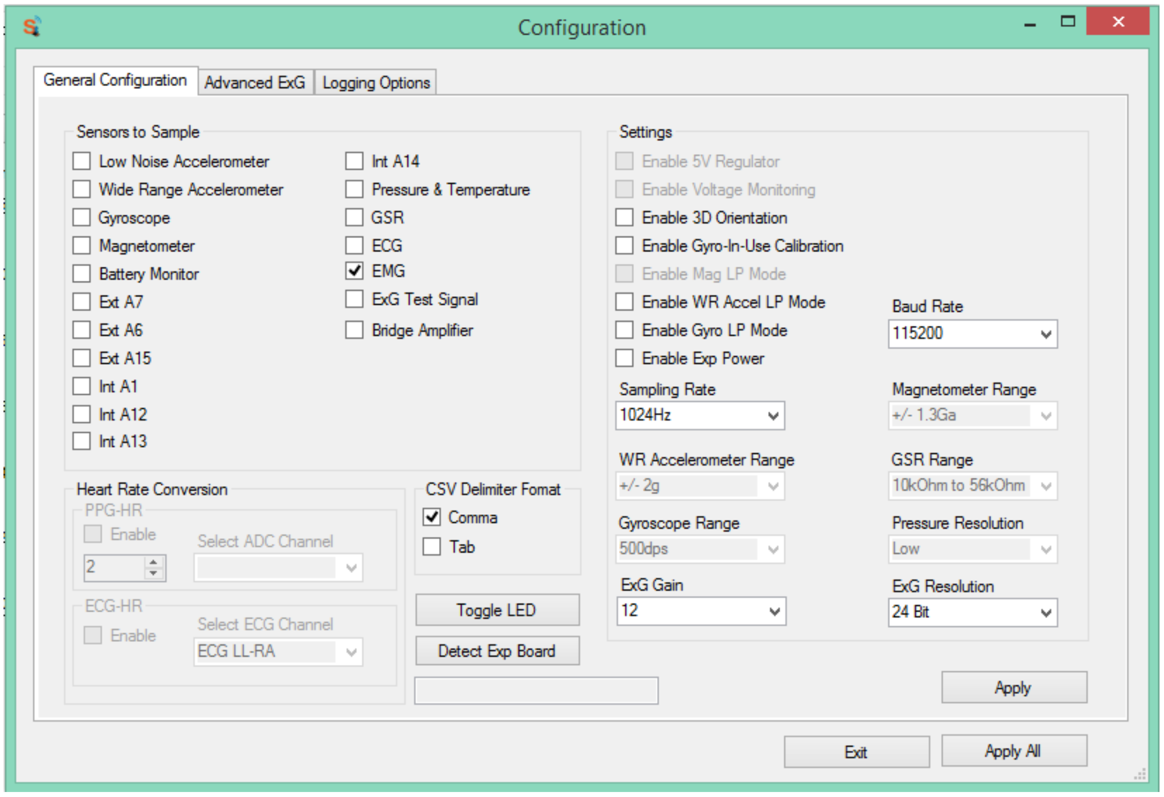
This software has only been tested and verified on Windows 7 and 8.1, functionality on other operating systems is not guaranteed.

Paring the sensors

Initially it is necessary to pair each shimmer device with the host PC which can be done through the operating system's Bluetooth pairing wizard. This can easily be done in windows through "Add Device" in the Devices and Printers section of control panel. During the pairing process it is necessary to enter the Shimmer sensors pairing code which is "1234" as default. It is useful at this point to note the Com port which is associated with each sensor as they will be required.

Configuration

Once each device has been paired it will be necessary to ensure that they have been properly configured. Unfortunately due to limitation within the Shimmer API and its documentation each Shimmer sensor needs to be configured using Shimmers own software. This should be done using the latest version of the ShimmerCapture software from which it is possible to connect to the already paired Shimmer sensors by specifying the associated Com port. Once the sensor is connected to the ShimmerCapture software it is possible to access the configuration window from the drop down menu. The necessary settings for correct configuration are shown below in Figure 5 and Figure 6:



Sensors to Sample
- ☐ Low Noise Accelerometer
- ☐ Wide Range Accelerometer
- ☐ Gyroscope
- ☐ Magnetometer
- ☐ Battery Monitor
- ☐ Ext A7
- ☐ Ext A6
- ☐ Ext A15
- ☐ Int A1
- ☐ Int A12
- ☐ Int A13
- ☐ Int A14
- ☐ Pressure & Temperature
- ☐ GSR
- ☐ ECG
- ☑ EMG
- ☐ ExG Test Signal
- ☐ Bridge Amplifier

Heart Rate Conversion
- PPG-HR: ☐ Enable; 2; Select ADC Channel
- ECG-HR: ☐ Enable; Select ECG Channel: ECG LL-RA

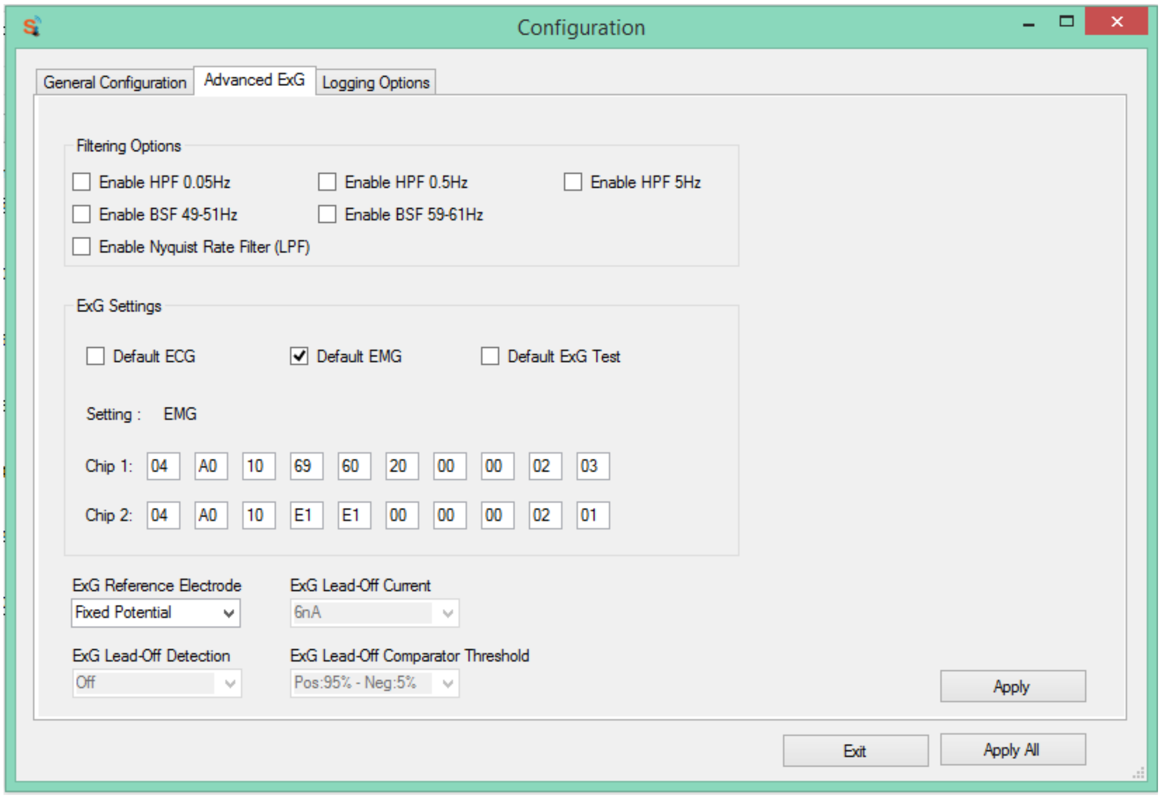
CSV Delimiter Fomat
- ☑ Comma
- ☐ Tab

Settings
- ☐ Enable 5V Regulator
- ☐ Enable Voltage Monitoring
- ☐ Enable 3D Orientation
- ☐ Enable Gyro-In-Use Calibration
- ☐ Enable Mag LP Mode
- ☐ Enable WR Accel LP Mode
- ☐ Enable Gyro LP Mode
- ☐ Enable Exp Power

Baud Rate	115200
Sampling Rate	1024Hz
Magnetometer Range	+/- 1.3Ga
WR Accelerometer Range	+/- 2g
GSR Range	10kOhm to 56kOhm
Gyroscope Range	500dps
Pressure Resolution	Low
ExG Gain	12
ExG Resolution	24 Bit

Figure 5: Sensor configuration part 1

Figure 5 shows the correct settings for the first configuration page within ShimmerCapture. Although it is possible to select any number of "Sensors to Sample" it is recommended that only the EMG sensor is selected. The EMG sensors alone send a large volume of data and therefore we do not want to saturate the receiving device by sending unrequired sensor data. The sampling rate should be set to 1024 Hz in order to ensure aliasing does not occur and to allow the processed data to be as informative as possible. The ExG Gain should be set to 12 dB and the Resolution to 24 Bit in order to make sure data is within the expected range. Finally the Baud Rate should be set to 115200 bit/sec so that seamless communication occurs. All other options should be un-selected as seen in the image



Filtering Options

☐ Enable HPF 0.05Hz ☐ Enable HPF 0.5Hz ☐ Enable HPF 5Hz
☐ Enable BSF 49-51Hz ☐ Enable BSF 59-61Hz
☐ Enable Nyquist Rate Filter (LPF)

ExG Settings

☐ Default ECG ☑ Default EMG ☐ Default ExG Test

Setting : EMG

Chip 1:	04	A0	10	69	60	20	00	00	02	03
Chip 2:	04	A0	10	E1	E1	00	00	00	02	01

ExG Reference Electrode: Fixed Potential
ExG Lead-Off Current: 6nA
ExG Lead-Off Detection: Off
ExG Lead-Off Comparator Threshold: Pos:95% - Neg:5%

Figure 6: Sensor configuration part 2

Figure 6 shows the correct settings for the second configuration page within ShimmerCapture. The only settings which may have to be altered on this page are the ExG settings. These are set using ten single byte registers for each ExG chip. Fortunately it is not necessary to know what setting each byte register represents as the “Default EMG” checkbox will change each to acceptable values. However, if necessary the function of each byte register can be found within the Shimmer Device manual. All filtering of data is performed by the application receiving the data; therefore it is not necessary to change anything in the “Filtering Options”.

In order to save and update the configuration on the Shimmer device it is necessary to press the “Apply All” button.

GUI Operation

Once each Shimmer sensor has been properly configured it will be possible to run the EMG system by using the Graphical User Interface (GUI) provided. The interface has been designed with ease of use in mind and it can be seen in Figure 7. It is possible to use the list box to select different devices and the settings for each device is shown to the right.

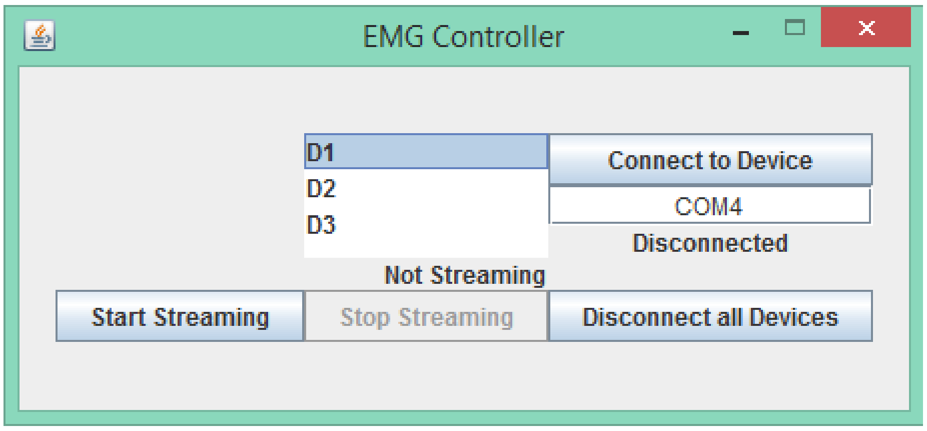
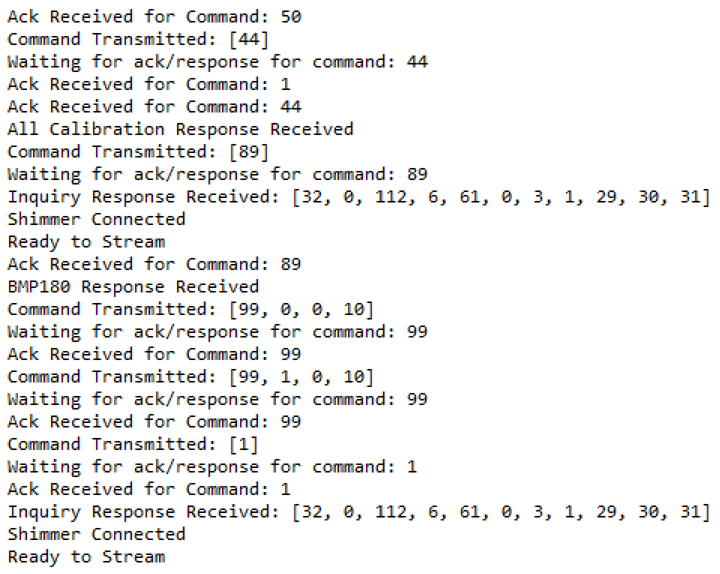


Figure 7: Graphical User Interface for EMG

Status information such as connection status, streaming status and Com port are shown. It is possible to change the Com port which the device is associated with through the Com port text box, all changes should be completed with the return key to make sure they update.

All information for the user is printed out to the console. In order to start data acquisition it is necessary, once paired, to connect to each device. Once each device is connected the “Start Streaming” button will send a stream command to each device and the received data will be printed onto the console. An example of this can be seen below in Figure 8:



```
Ack Received for Command: 50
Command Transmitted: [44]
Waiting for ack/response for command: 44
Ack Received for Command: 1
Ack Received for Command: 44
All Calibration Response Received
Command Transmitted: [89]
Waiting for ack/response for command: 89
Inquiry Response Received: [32, 0, 112, 6, 61, 0, 3, 1, 29, 30, 31]
Shimmer Connected
Ready to Stream
Ack Received for Command: 89
BMP180 Response Received
Command Transmitted: [99, 0, 0, 10]
Waiting for ack/response for command: 99
Ack Received for Command: 99
Command Transmitted: [99, 1, 0, 10]
Waiting for ack/response for command: 99
Ack Received for Command: 99
Command Transmitted: [1]
Waiting for ack/response for command: 1
Ack Received for Command: 1
Inquiry Response Received: [32, 0, 112, 6, 61, 0, 3, 1, 29, 30, 31]
Shimmer Connected
Ready to Stream
```

Figure 8: Sample of feedback printed to console

When it is desired to stop data acquisition the “Stop Streaming” button should be pressed followed by the “Disconnect all Devices” button. This will ensure all devices gracefully sever the connection with the host PC. If it is desired for streaming to be started again then it is recommended that the application is shut down and restarted in order to ensure correct operation.

Sometimes a device may not connect on first attempt and a “port busy” exception will be thrown. This intermittently happens, and it can be solved by just reattempting the connection. If the problem persists then it is recommended to double check the port number is correct and the device is powered on.