Polygraph Realization for Data Visualization and Analysis

Herman Watson  
Honor Design, Inc.  
Miami, FL USA  
herman@honordesign.com

Malcolm Heimer  
Dept. of Electrical Engineering  
Florida International University  
Miami, FL, USA  
M.Heimer@fiu.edu

Jean Andrian  
Dept. of Electrical Engineering  
Florida International University  
Miami, FL, USA  
JeanAndrian@fiu.edu

Armando Barreto  
Dept. of Electrical Engineering  
Florida International University  
Miami, FL, USA  
Armando.Barreto@fiu.edu

Abstract

A polygraph is a tool which provides access, visualization and analysis for large amounts of recorded physiologic data. It facilitates detailing and studying data from multiple transducers recorded at the same time. This work describes an implementation using an Open Source development framework. A specific data format stored real time sampled data. Key design details link the recorded data, internal epochs, montages, and the graphic display. Selected development framework components are discussed plus links between components and data which allows realization. Implementation details are shown where architecture is critical to overall functionality. Storage formats, lists, plots, and data retrieval are core components of the design.

1. Introduction: Polygraph analogy

A multichannel, multi-pen polygraphic recorder using paper output was the model for this System for Visualization and Analysis of Polygraphic Data. Each signal is given its own channel. Channels have been designed so filtering, or AC/DC coupling can be added.

A framed data stream design uses frames to facilitate an epoch based architecture for replay. Epochs are constructed during replay so as not to burden the recording side of the process.

Providing a channel for each signal allows expansion of the system adding derived and computed virtual channels. This architecture allows adding traces derived from measurements of real time data channels. Virtual channels and derived signals provide extra channels to function as a resource for visualization. Filtering and wavelet analysis are examples of channels having derived signals.

2. Defining the problem

A major challenge exists between joining a database which contains huge amounts of real time data with a sophisticated, complex graphic user interface. Each side of the equation is demanding with unique characteristics so finding a solution requires a way to join two unrelated worlds together.

The first problem requires identification of a software framework giving the highest level of integration and functionality possible plus still allowing easy development of a solution within a reasonable time using limited resources.

The second problem requires developing a format for real time data that uses simple recording techniques. Random access and fault tolerance are requirements for the format. Variable length sections of the data need to be selectable enabling snapshots or epochs for the graphic user interface (GUI).

The third problem requires developing a GUI capable of displaying variable numbers of channels. The display also must have varying time width and operate as a movable view port over the data. Channels need to identify themselves and be capable of varying gain, offset and time zoom. The GUI must display predefined montages (groups of channels) as well as gains and offsets. Mouse controls must provide ability to identify specific time and amplitude values of a single sample within the display.

The final problem requires tying data functionally to the GUI with interrelated controls and interfaces. The GUI has
to show specific data within the database at specific times and variable durations. Additionally, the GUI must also include capabilities for expansion, data analysis, and virtual channels.

3. Open Source Resources

Open Source was chosen as the development environment for writing code. C++ is the language that used for project implementation. The resulting application consists of more than 12,000 lines of C++ code. Object oriented philosophy and development techniques are important features used with the language.

Resources include compilers, development tools, environments and editors. Complete development frameworks already have a high level of sophistication and integration in place.

Pricing is not a challenge with open source resources, so Open Source provided an ideal solution with development, implementation, and quality.

A large collection of sample applications are available, providing a rapid learning environment. Additional stand-alone example applications helped reduce the learning curve.

4. Selection of wxWidgets

After systematic review of suitable platforms, wxWidgets framework was selected[1]. Platform independence is one major reason the framework was selected. The framework contains a collection of classes and libraries with over 400 different classes. Classes themselves provide many functions and methods, so development takes place almost exclusively at that level.

Many different types of classes are defined including file input/output, html, sockets, lists, arrays, and strings. The framework is very similar to commercial function class libraries and frameworks.

Code written using commercial 'standard' template libraries are not platform independent because of compiler implementation dependencies and have been eliminated.

wxWidgets is released under the GPL. There is an exception in that binary executables can be distributed without source code which can be an important concern for commercial projects.

wxWidgets is different than other multi-platform GUI libraries using native operating system controls whenever possible. The user interface of an application has the familiar look and feel of the host operating system. Controls when not available are emulated. wxWidgets is currently available for various flavors of Linux and UNIX plus Microsoft Windows, and Macintosh OS. There are also embedded versions for limited hardware devices.

Debugging is an important feature for code development. A logging class enables insertion of print statements in code with output directed to a “log” window. Additionally, the whole framework can be compiled with a 'debug' option which provides full traceback facilities through internal libraries and classes. Traceback uses native system debugger which accompanies the compiler. Debugging helps when an application hangs or causes an exception which halts execution. The stack is saved including the function call and method names indicating where the exception occurred.

A wxPrinter class exists which not detailed here. This class uses device contexts providing ability to print graphics and text. Having an integrated printer feature is necessary for realizing the polygraph system with hard copies.

Documentation is excellent with a huge (1000 page) reference manual. The docs are available in PDF, HTML, and Windows Help files.

Support is tremendous because of available specific news groups. A large user community exists throughout the world. Newsgroups are archived and available through search engines giving a rich and diverse knowledge base.

5. Framing the Raw Data

Real time data by nature are sequential and flow continuously without interruption.

A recording system will digitize real time input signals continuously and form a stream saved to a data file. By adding data minimal processing on the recording side, the data file can be blocked into frames.

Advantages can be gained by formatting recorded data into frames. One is that frames permit the recorded data to become fault tolerant as regards breaks or losses in the saved data stream. Breaks and losses can be almost ignored (jumped over) in replay processing, allowing all of the remaining data stream to be recovered. This ability eliminates loss of a whole file. Only spots where data losses occur will have gaps, not loss of all the data following the break.

Data frames are an important requirement to implementation of the Polygraph system because integrity, multiplexing, and checksums are all incorporated into frames as well as synchronization characters, byte counts, time of day, etc.. Simple enhancements added to recording the data stream with simple processing techniques give sophisticated analysis and manipulation capabilities on the replay side.

Once the data stream is formatted into frames, sections can be edited, selected, and cut as part of the playback process.
Framing adds the capability for multiple sample rates. Adding frames allows multiples of the base frame sample rate for data sampling. A single frame can contain multiple samples or conversely multiple frames can span across a single sample.

Using frame architecture in a sequential serial data stream makes simple demands on the recording side during the data collection process. It does not require multiple output files or streams. A single output file or data channel can be used to accommodate multichannel inputs formatted into frames. The single file architecture assures time synchronization of data samples preventing problems associated with multiple parallel files.

6. Polygraph Framework - wxList

wxList provides linked list functionality for wxWidgets. Linked lists are used extensively within this application. Linked lists provide optimized rapid sequential accesses of data contained within a list. In this application lists are used to drive graphics visualization by tying snapshots (epochs) of the external large recorded database file with the display GUI library.

![Figure 1 Block Data Class](image)

A data class is defined to implement the list block structure carrying the format of the binary data from the recorded database. wxList is used to build, store and access the blocks of data.

A block data class is defined and shown in Figure 1. The block data class maps frames to an internal list of blocks. The list blocks contain data supplied from the external database file.

A data block is filled and appended to the list. An input loop is used to read data from the data source, place the data into a block class instance and then append the new block onto the list. Repeating this process results in the linked list of data blocks giving the epoch shown on the GUI display.

GUI processing methods require high speed retrieval of list data. Display access is sequential because GUI methods use iteration. wxList data structure is optimally designed for sequential access. Iteration is accomplished using a 'for' loop. Loop initialization takes place by creating a Node pointer for the first node of the list using the GetFirst() method of wxList. Incrementing the loop is done by the GetNext() method. Incrementing causes pointer indexing to the next list node. Loop iteration stops when GetNext() returns NULL, indicating the end of the list has been reached.

7. Mapping input data to wxList

Input frames are saved in the recorded database in a way similar to a motion picture. Database design is intended to prevent data loss, breaks, or interruptions rendering the database unusable if an error occurs. The database can recover from losses and retain all remaining data.

Considering the analogy of a motion picture, if a film breaks it can be spliced back together while showing the remainder of the movie, even if a small group of frames are lost. Consider a movie which is 2 ½ hours in length, if 10 seconds are lost because film is crushed or ripped, the remainder of the movie is not lost, just a few seconds.

The recorded database is saved in one continuous file to reduce the number of pathways and amount of operating system overhead required if using multiple files. Additionally, a single sequential data path can be used between data collection and recording equipment.

![Figure 2 Input File Frame](image)

The database begins with the first frame, then the second, third, and fourth frames all the way through the Nth frame. Data are segmented becoming the data frames.
Even though the data are sampled continuously, they are formatted and saved in frames. The frames are sequential through time within the file and are saved at a rate of 50 frames per second. Data are real time, continuous and are sampled without interruption. The frame architecture allows sample rates which are multiples of the frame rate to be hosted.

The frame is constructed with different types of information inside it. Figure 2 The first information in the frame contains a header. The header contains a synchronization value. Following the synchronization header is a multiplex code byte which carries decoding keys for data contained within the frame.

Real time data consists of samples acquired at regular clock timed intervals. First within the block is a group of real time data consisting of samples continuously recorded from an analog to digital converter. The real time data has channels for ECG and respiration samples. ECG is sampled at 200 points per second, so each block contains four real time ECG samples which are sequential in time.

Another type of real time data is where respiration data are sampled at 50 points per second. Each block contains one sample per respiration channel; one for the rib cage and one for the abdominal channels. Respiration data are scaled by the monitor when recorded, so they are added together on playback to generate the derived 'Sum' signal which represents respiration.

Another group of information is called vital data which records monitor conditions including alarm and display values occurring during the monitoring period; including heart and breath rate values. Values saved are numbers that were displayed on the recording monitor which reproduce the values shown while recording.

Another group of information is an auxiliary data group which is also a function of the state of the monitor. This group includes monitor status plus information generated while monitoring. Included in this group are preference settings of the recording monitor such as system alarm code, calibration values, alarm threshold settings, alarm indicator status, time and date of the recordings. There are 50 slots for auxiliary data; one slot is recorded for each block of data. With 50 slots and frame block rate, the auxiliary data are repeated once each second. Each block contains one auxiliary data value and those values are unique for each of the 50 frames that are recorded in a second.

There is a multiplex code byte that is recorded with the header information which functions as a serial decoding key. That code byte is used to indicate which type of vital and auxiliary data are contained within a each single data block.

The blocked frame architecture allows fairly simple recording requirements for encoding while allowing sophisticated replay analysis techniques. The block frame scheme allows random access in a large database even though it was recorded sequentially in real time. Time and location are determined as data is read from any randomly placed starting point within the binary database by retrieving values stored locally within the frames. Timing and location information is included within the frame sequence and is repeated often. The design allows a pointer to be positioned randomly in the file, then data can be recalled locally with time and location accuracy.

The database architecture is an important element for visualizing and analyzing records. Use of a database design as part of the polygraph replay system architecture accommodates multiple channels of information required. Random access is possible within the serially recorded database without requiring an external index. Sophisticated analysis can be performed on selected sections or the whole body of data within a single patient recording. The integrity of the data is confirmed through the serial repetition of key sequence information and also utilizes checksums and sequence numbers within the frame blocks. Dropped and lost information can be detected from key number sequences, corrupt data can be determined from checksums, and recovery from errors is quick as soon as data stream replay frame synchronization recovers.

8. GUI Design

wxPlotWindow is a specialized window class designed to display repetitively acquired data that may have thousands of values. The recording can produce thousands of values per minute with multiple measurements being done simultaneously. Resulting curves can be examined, enlarging them or scrolling from one position to another.

wxPlotWindow class features controls that allow shifting the baseline (offset) and increasing/decreasing the gain of a trace. The x-zoom time detail of a trace can be modified interactively while being displayed. A single trace is selected with a click of the mouse. When selected, the intensity of the trace changes indicating that it has been highlighted and selected. The selected trace can then be adjusted with gain and baseline controls interactively.

wxPlotWindow class provides all the features described above including trace controls and multichannel capability.

What is required is that specific class code has to be written giving the connection between the plot library and the data to be plotted. The wxPlotWindow is placed in a wxFrame placed in a wxApp application showing plots on the display.

9. Channel Design
In Figure 3 m_plot is the plot window displayed on the screen. MyPlotTraces are the channels shown on the screen. MyPlotTrace instances are added to the display with an m_plot method. The traces are specified by MtgList which is a list of all active traces on the screen. The MtgList is a subset of the Master Item List which is a catalog of all potential traces. Each Master Item has information included within it including an instance of a Signal Processing Specification that allows information to be retrieved for m_plot.

![MyFrame:wxFrame][1]

**Figure 3 wxPlotWindow design**

A channel is defined as the data trace placed in a wxPlotCurve class instance.

From the lowest layer, a channel uses several elements of background information. There is a MySignalProcessing data class used to specify the physical layout of types of signals that will be coming into the polygraph from the database.

A list of signal processing specifications is built which contains a list of signal processing class instances for each signal to which it refers.

Each display channel has a signal processing specification instance in its definition. ECG, respiration RC, AB and Sum, etc..., all have specifications detailed for every type of signal.

From a bottom-up perspective, the lowest layer is a Signal Processing Specification for each type of signal coming into the polygraph. The next higher layer is a Master Item list for each trace that can be displayed on the polygraph. Master Item list includes signal processing specifications and details for each available channel. Higher up is a montage list (MtgList) which contains Master Items for all of the selected channels to be actively committed to the display. The top layer places the MtgList channels on the display window by using the MyAddTrace method for m_plot (the wxPlotWindow).

From a top down perspective the highest layer contains a frame instance created within the wxWidgets application. The next lower layer places a wxPlotWindow instance called m_plot inside the frame.

The next layer is MtgList, a list containing a montage of all the channels actively displayed on the polygraph. MtgList is committed to m_plot with a method called MyAddTraces, a method whose purpose is to pass parameters for data class instances to the wxPlotWindow class and make them active traces.

The next layer down has instances of MyPlotTrace curves appended to and contained in the list called MtgList described above. MtgList is used to place wxPlotCurve instances inside of m_plot.

10. **FindY - bridge from data to plot**

When m_plot refreshes and wants to draw a trace, it calls the methods in the MyPlotTrace class. Method design provides a bridge between a raw data wxList and the display screen.

The first method called is GetStartX() which gets the beginning X Value. Then GetEndX() gets the ending X value. A loop calls GetY() repeatedly returning data points from the the Start X value to the End X value in a sequential fashion.

Since the MyPlotTrace methods were written in C++ , the tradition is to have a routine for each instance method of each trace and type of data. C++ conventions require that each trace has its own local set of data. The challenge in this project is to establish a common link between multiple displayed traces and a single set of data; the RawData wxList containing data frames for the Epoch.

A single set of routines was generated which have a global reference. This contrasts C++ conventions, but accessing the data globally allows a single set of functions to be called for all the different types of display traces available. In this polygraph implementation only one set of routines used to supply data to the plot libraries. The routines are defined as FindStartX(), FindEndX(), and FindPointY(), ()

FindPointY() is a global routine. FindPointY() is supplied a calling argument of MyPlotTraceData which includes details of which data is required and where it is located. Every MyPlotTraceData instance tracks its own trace data used on the screen. Each trace has its own unique instance of MyPlotTraceData class.

MyPlotTraceData class information for each polygraph channel displayed contains pointers, details from the Master Item List, and a Signal Processing Specification instance.
Using an architecture for global routine access which includes different trace styles, specific trace data can be determined from a frame data block. Data in a node of the RawData list can be queried in a unique way by the FindPointY() routine. Each query supplies a sample data point to the plot library. Multiple plot traces generate multiple queries. Each plot trace carries its own data description instance including the Master Item details and the Signal Processing Specification instance which is appropriate for the channel.

11. Result

Figure 4 Polygraph Display - 12 channels

Figure 4 is an example of the display with a 12 channel montage displayed. The controls on the left side of the display allows size and offset adjustments for a selected channel.

A channel can be selected by clicking the trace with the mouse cursor. Two more controls provide for expanding or contracting the scale for the x-axis which allows zooming in the time axis to provide detail. The layout of channels plus their respective gains and offsets can be saved as a montage which can later be recalled.

Figure 5 shows a single channel montage with the electrocardiogram signal whose gain and zoom is increased to show detail in the trace. Individual measurements with the cursor are possible down to individual sample resolution.

12. Conclusion

This work presents details for the realization of a polygraph for standard personal computer hardware. Major components were the software framework, formats for data, graphic user interface, and tying all the components together with a design architecture.

Publicly available components were used which offered high levels of functionality which simplified realization. Open Source code allowed modification and enhancements that provided critical functionality. All the resources allowed development of a scientific tool with limited time and personnel.

Advantages of this system are as follows:

1. Becomes an Open Source resource which can be reused, modified and/or expanded. Might fast track additional research.

2. Used to analyze data recorded in sleep disorders center. Wavelet analysis was added allowing specialized filtering. Cross correlation calculations between channels were added.

3. Visualization and analysis is now available for public domain data. Prerecorded data exist in a similar format public database for CHIME[2] study which could provide future analysis opportunities.

13. References
