A Graphical User Interface for the Study of Heart Rate Variability

PP Domitrovich

Washington University School of Medicine, St Louis, MO, USA

Abstract

A truly windows-based graphical user interface (GUI) for Heart Rate Variability (HRV) is an easy-to-use, educational, and valuable research tool. The development of such a GUI is not a trivial endeavor.

The document-view architecture of the Microsoft Visual Studio (MSVS), which is based upon the Microsoft Foundation Classes (MFC) written in C++, is utilized. A dynamic splitter-window is included to provide 4 simultaneous views. The GUI allows 720 various segment lengths. A total 270 different 4-paned views are possible, and 59 adjustable parameters are available. One view contains an interactively editable heart-rate-arousal and duration detector. The Lomb periodogram, the fast-Fourier transform (FFT), Poincare' plots, HRV in text format, phase-plane plots, multi-line heart-rate tachograms, and 2-minute-averaged Lomb and FFT power spectral amplitude plots are available.

1. Introduction

The input information for any HRV calculation is the interbeat-interval or beat file. HRV calculations often provide output that can be displayed graphically to provide additional insight into the underlying physiology. Such graphical displays can also promote the development of intuition concerning HRV if approached cautiously and thoughtfully. This HRV GUI was developed to provide opportunities for learning with immediate feedback, which can be used to correct intuition when necessary. It was also developed to be the basis for applications requiring user interaction. Its success is heavily dependent upon the user in any case.

This paper provides a minimal overview of the work that went into the development of this GUI. The most basic concepts are covered, and several more difficult GUI programming steps are discussed briefly. HRV is discussed at the application level, with little detail provided. With supplemental literature concerning MFC and the document/view architecture, and with sufficient experience with HRV calculations and programming, reproduction of this GUI is possible.

2. Methods

The document/view architecture was written by Microsoft Corporation (Redmond, WA, U.S.A) with the MFC of the MSVS. There are several very useful texts written about MFC in the MSVS [1,2]. This architecture still exists today and still utilizes the MFC in the most recent versions of the MSVS [3].

The document/view architecture of MFC provides many classes to assist in the development of certain GUI's. This HRV application fits nicely into the document/view architecture. The architecture provides a method for storing and loading files, and a method for presenting the information stored in the files to the user. The classes of the architecture are abstract enough that the type of information and the method of presentation of the information are quite unrestricted. Many standard functions expected of a user interface are built-in, which allows the programmer to focus on the data presentation and manipulation. Since MFC is written in C++, and the details of the user interface must be written in C++, a good deal of programming is required. Nevertheless, a distinct advantage of using C++ and MFC is that the graphical user interface can manipulate and present data very rapidly on modern computers. Several C/windows functions, which have not been wrapped in MFC, are required for this application.

The classes of the document/view architecture most relevant to this work are called the document, the view, and the mainframe. The document class manages the data, the view class presents the data to the user, and the mainframe class manages the frame window containing the view along with menus, tool bars, and status bars attached to the frame window. Through pointers to the document, view, and mainframe objects, there is essential communication made possible. The communication between the document and view objects is bidirectional, while the communication from the mainframe object proceeds only to the view object.

Interbeat-interval files must include the start time of the first beat, the beat labels for all beats, and the RR intervals for all beats. Periods of noise or ectopy can be labeled as desired. It is convenient if all interbeat-interval

or beat files have the same or similar extensions, which can be selected with the typical dialog available for opening files. However, the extension chosen for the GUI must differ from commonly utilized beat file extensions. The extension for the serialized beat file is saved in the registry when implementing the wizard in MSVS to set up the single-document interface (SDI).

One way to serialize the beat file is first to rewrite the file in binary format as a series of structures containing sufficient information to allow for quick and easy navigation through the serialized beat file once it is stored in memory. The structure chosen for this GUI contains the segment number, the beat label, the real time of the beat, and the RR interval. The built-in File->Open menu selection must call a user-supplied method to call the built-in file-open dialogue, to accomplish the translation of the text file into a series of structures in binary format, and to save the translated file with the extension required by the GUI and the given base name of the original beat file. Then the supplied MFC methods in the document class for creating a new document, opening a document, and serialization are overridden for processing the saved binary file. Serialization results in storing the beat file in memory for use with the GUI.

An additional tool bar, created in the mainframe and positioned below the views, is utilized for navigation through the beat file. Buttons are created to move one segment to the left or right, and also one hour to the left or right. For segments greater than one hour, the one-hour change is not available. A reset button is provided to set the segment number to zero and bring the views back to the beginning of the beat file. The code for the buttons is user-supplied and written in the document class.

In detail, a sequential search through the serialized data in memory is performed until the desired segment number is found. Once these data are found, they are written to a text file and read back into memory, although the data can be copied directly within the program if desired. Two public methods are user-supplied in order to aide in the transfer of the raw segment data and other tracking data to the view classes. The tracking data consists of the document length in bytes, the number of document lines or structures, the segment number, the segment start time, and several other current display settings for several specific applications for the various views. The MFC document class method GetDocument is used in the view classes to obtain a pointer to the document object. This pointer allows access to the user-supplied methods in the document class for transferring the raw segment data and tracking data.

The actual code to create the MFC dynamic splitter-window object is available from a Microsoft textbook [4]. The splitter window object is instantiated in the mainframe class using the MFC method OnCreateClient. The default view, or view1, is created automatically in row 0, column 0. View2 is placed in row 1, column 0; view3 in row 0, column 1; and view4 in row 1, column 1. The views are filled sequentially. The splitter bars can be moved at will with the mouse pointer, which resizes the views accordingly. Font sizes for each view are programmed to adapt accordingly, utilizing the automatic changes in size of the characters as the view size changes.

Note that only view1 maintains a pointer from the mainframe, while all views have equal access to a pointer from the document class. The connection of view1 to the mainframe class dictates that view1 will always be displayed. Buttons can be added to the mainframe to display various combinations of a selection of the 4 views; i.e., either 1 or 2 views, where view1 must always be included. The code to implement these combinations is user supplied and written in the various view classes. In detail, a CWnd pointer from the parent of the particular view class is obtained and then cast as a general CSplitterWnd pointer. Next, this pointer is cast as a pointer to the specific splitter window class of the GUI. This allows access to the GUI's MFC splitter-window methods.

HRV classes are based upon a segment of data as input. Segments can be as short as 10s or as long as two hours, in increments of 10s. The data required for each segment contains the real time of each beat, each beat label, each RR interval, and the segment number. Whenever a segment is viewed, the document object writes a text file containing the segment's de-serialized data. This text file can be very useful when comparing the input data to the displayed data.

The HRV classes do not directly utilize any of the MFC. Each view instantiates a HRV object for manipulation of the segment data. The mainframe creates an HRV object to translate the decimal start and stop times of the current segment into Cstrings. Implementing this requires a user-supplied method to update the status bar when the segment number is updated and, within this method, a cast of a pointer obtained from the active document to a pointer to the document object of the GUI. The document instantiates a HRV object for a HRV computation over each segment and outputs the results into a file.

Each major GUI object dynamically creates memory for the data of the segment each time a segment is chosen for display. The memory is freed by hand when the object is destroyed. Note that the entire serialized beat file remains in memory throughout the view time. The MFC document method DeleteContents is overridden to clear the memory of the current serialized beat file and to reset other parameters when a new file is opened. Several GUI's can actually be run in the same directory without interference.

The HRV methods in the HRV class are public methods that calculate both HRV for a segment and provide some user-supplied utility methods. Methods for HRV calculation of the time domain and several non-linear variables are included in this class. The basic methods for a FFT [5] or Lomb [6] spectral analysis are also included. The special method for detecting sleep apnea is in this class. The general utility methods include a stationarity test, a sorting method, and a method to obtain RR intervals as a function of time. The data for a segment in this class is protected.

The HRVFFT class inherits from the HRV class. This class contains public methods that are auxiliary methods for the FFT and Lomb spectral analyses. Such methods include sampling of the NN interval time series for a segment, determining the sampling interval for any segment length automatically, and other standard methods for a FFT. The frequency band limits, sampling interval, and number of FFT points are all protected data members. Parameters adjustable for spectral plots are separately included in a 2-sheet set of property sheets available on the mainframe. This property sheet is called in the document class so that any changes can be shared with all views.

The remainder of the code required for the various views resides in the particular view classes themselves. Plotting-related classes could be implemented to reduce the amount of code contained in the views. All calculations required for the views are selected with parameters set in the various dialogue boxes, menus, toolbars, or additional buttons. Each possible view is available during runtime due to message handling of the architecture of the GUI and the MFC update methods for views.

3. Results

For brevity, not all of the applications available for each view will be discussed. Each view has its own dialogue box, where most of the applications can be selected for display. The dialogue boxes can be accessed through the context menu included in each view, through buttons on the frame window, and through menus on the frame window. Accelerator keys and tool tips are provided appropriately. The data exchange code for each dialogue box is contained in the appropriate view class, as is the code for each context menu. Each view will be discussed separately. See Figures 1 and 2 for examples.

View1 is always displayed, as discussed earlier. It contains a text editor for taking notes while viewing the other 3 views and a menu item and associated method to save the notes to a text file. The view also contains a multi-line heart-rate tachogram, the format of which was originally developed by the author and utilized to detect sleep apnea [7]. The time duration of the lines in the multi-line tachogram is a settable parameter in the dialogue box for view1. View1 also displays a power spectral plot using a FFT associated with 2 property sheets, a Poincare' plot with adjustable limits for the maximum and minimum NN plot limits, and a unique graphical representation of approximate entropy. The approximate entropy (ApEn) display plots the re-sampled point number i versus ln of the total number of times points i and i+1 are both within a distance r of the points j and j+1, respectively, for all j, divided by the total number of times points i, i+1, and i+2 are within a distance r of the points j, j+1, and j+2, respectively, for all j. ApEn is the mean of the plotted values. The Poincare' plot indicates the number of occurrences of a particular point with darker shades of red corresponding to larger numbers of occurrence.

View2 contains a single-line and multi-line heart-rate tachogram with adjustable maximum- and minimum-plotted heart rates. The multi-line tachogram also has adjustable time duration of the lines. This view contains an editable heart-rate arousal detector, which automatically locates the peak, beginning, and end of each arousal. The algorithm is original and is based upon slope for all 3 times of interest and upon heart rate maximum and minimum differences. Peaks, beginnings, and ends of the arousals can be removed with the mouse wheel click, Cntrl+mouse wheel click and Shift+mouse wheel click, respectively. Peaks, beginnings, and ends can be added in the same manner, but with a double-click instead of the mouse wheel. The times of the peak, beginning, and end of the arousal, along with the duration of the arousal and the maximum to minimum heart rate change are automatically written to a text file as one proceeds through the beat file. One can take advantage of the capability of the button that displays only view1 and view2 in order to enlarge the single-line tachogram for heart rate arousal detection.

View3 contains a 2-dimensional phase plane plot and a textual listing of HRV results for the current segment. The phase plane plot has adjustable maximum and minimum NN plotting limits.

View4 offers the same Poincare' and power spectral plots as view1. It also offers a Lomb power spectral plot. These options are contained in the dialogue box for view4. In addition to the options offered in the dialogue box, two buttons on the tool bar offer a 2-minute-averaged FFT and Lomb power amplitude spectral plot. HRV options located on the tool bar were written primarily to facilitate applications not presented in this paper and utilized for other publications [8].

4. Discussion and conclusions

The GUI's most important classes, the document class and the view class, along with the HRV classes, require

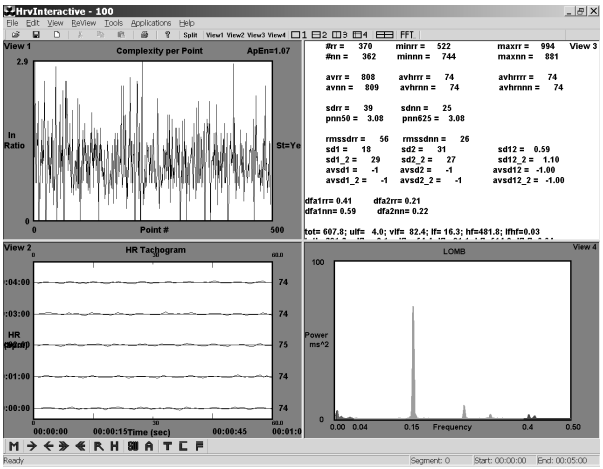

Figure 1. View1 displays ApEn. View2 displays the multi-line-tachogram. View3 displays HRV results. View4 displays the Lomb periodogram.

considerable user-supplied development. The basic functions of the GUI are handled primarily with built-in code. Plotting code and HRV calculations can be added to the view classes, while parameters to run these applications can be chosen or modified in dialogue boxes.

The HRV applications displayed in this GUI are mostly standard HRV calculations. Clearly, freedom exists to calculate and display any HRV application desired. The fact that the GUI is based upon segments still allows for 24-hour HRV calculations segment-by-segment. The results of these 24-hour calculations can be displayed in certain applications of this GUI.

Once a functioning basic SDI GUI is developed, one may create a HRV GUI as desired. The document/view architecture of the MSVS is a reasonable way to proceed.

Acknowledgements

I wish to that Phyllis K. Stein, Ph.D. for her funding support for this work.

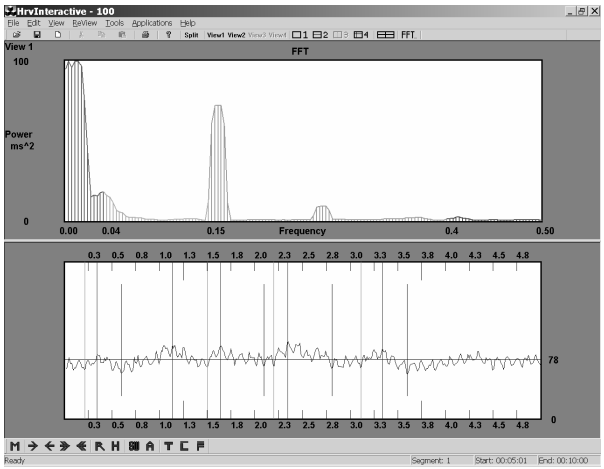

Figure 2. View1 (top) shows the FFT power spectrum, and view2 shows the heart-rate arousal detector.

References

[1] Blaszczak M. Professional MFC with Visual C++ 6. Birmingham UK: Wrox Press Ltd, 1999.

[2] Prosise J. Programming Windows with MFC, 2nd Ed. Redmond, WA, USA: Microsoft Press, 1999.

[3] Skibo C, Young M, Johnson B. Working with Microsoft Visual Studio 2005. Redmond WA, USA: Microsoft Press, 2006.

[4] Blaszczak M. Professional MFC with Visual C++ 6. Birmingham UK: Wrox Press Ltd, 1999:338-46.

[5] Rottman JN, Steinman RC, Albrecht P, Bigger JT Jr, Rolnitzky LM, Fleiss JL. Efficient Estimation of the Heart Period Power Spectrum Suitable for Physiologic or Pharmacologic Studies. Am J Card 1990;66:1522-4.

[6] Press WH, Teukolsky SA, Vetterling WT, Flannery BR. Numerical Recipes in C, 2nd Ed. New York, NY, USA: Cambridge University Press, 1992:575-84.

[7] Stein PK and Domitrovich PP. Detecting OSAHS from Patterns Seen on Heart-Rate Tachograms. Computers in Cardiology 2000;27:271-4.

[8] Stein PK, Cohen RJ, Mau B, Domitrovich PP, Gottdiener JM, Redline S. A Comparison of Holter- and Polysomnogram-Based Detection of Bed and Wake Times. Computers in Cardiology 2007 (in press).

Address for correspondence

Peter P. Domitrovich, Ph.D.
Washington University School of Medicine
Department of Internal Medicine
Division of Cardiovascular Diseases
Box 8086
St. Louis, MO 63108
U.S.A.
ppd@hrv.wustl.edu