ABSTRACT

This paper presents the implementation of online acquisition software for electrocardiogram (ECG) signals. This program is based on the Python language and it allows the adaptation of a commercial offline signal acquisition board into an online signal acquisition board. Here, the concept of online acquisition is applied when the signal can be analyzed in short time intervals, such as a few seconds. The developed system is divided into two parts: the acquisition module; and the graphical user interface (GUI). The acquisition module is responsible for recording and processing the raw signal, making it available to the graphics module. The developed structure allows the algorithm to be used in more complex applications, abstracting the acquisition logic. Furthermore, the acquisition module has acquisition settings, defined by the user when initializing the module. This work presents a simple example of an application using the module. In this case, the system stores the information from the acquisition module and constantly plots it on the screen. The program also integrates settings available by code, using command-line arguments. Therefore, the developed system presents itself as a powerful tool for online ECG signal acquisition, allowing the use of the board in human-machine interfaces.

Keywords: Online acquisition; ECG signal; human-machine interface; Python programming language.

RESUMO

Este artigo apresenta a implementação de software de aquisição online para sinais de eletrocardiograma (ECG). Este programa tem por base a linguagem Python e permite a adaptação de uma placa comercial de aquisição offline de sinais em uma placa de aquisição online de sinais. Aqui, o conceito de aquisição online é aplicado quando o
PACKET ANALYZER OF ONLINE ECG SIGNALS

sinal pode ser analisado em intervalos curtos de tempo, como alguns segundos. O sistema desenvolvido é dividido em duas partes: o módulo de aquisição; e a interface gráfica do usuário (IGU). O módulo de aquisição é responsável pelo registro e processamento do sinal cru, disponibilizando-o para o módulo gráfico. A estrutura desenvolvida permite a utilização do algoritmo em aplicações mais complexas, abstraindo a lógica de aquisição. Além disso, o módulo de aquisição possui configurações de aquisição, definidas pelo usuário ao inicializar o módulo. Este trabalho apresenta um exemplo simples de aplicação usando o módulo. Neste caso, o sistema armazena as informações do módulo de aquisição e as plota constantemente na tela. O programa também integra configurações disponíveis por código, usando argumentos de linha de comando. Portanto, o sistema desenvolvido se apresenta como uma ferramenta poderosa para aquisição online de sinal de ECG, permitindo o uso da placa em interfaces homem-máquina.

Palavras-chave: Aquisição online; sinal de ECG; interface homem-máquina; linguagem de programação Python.

RESUMEN
Este artículo presenta la implementación de un software de adquisición en línea de señales de electrocardiograma (ECG). Este programa está basado en el lenguaje Python y permite la adaptación de una placa comercial de adquisición de señales offline a una placa de adquisición de señales online. Aquí, el concepto de adquisición en línea se aplica cuando la señal se puede analizar en intervalos de tiempo cortos, como unos pocos segundos. El sistema desarrollado se divide en dos partes: el módulo de adquisición; y la interfaz gráfica de usuario (GUI). El módulo de adquisición es responsable de registrar y procesar la señal sin procesar, poniéndola a disposición del módulo gráfico. La estructura desarrollada permite utilizar el algoritmo en aplicaciones más complejas, abstrayendo la lógica de adquisición. Además, el módulo de adquisición tiene configuraciones de adquisición, definidas por el usuario al inicializar el módulo. Este trabajo presenta un ejemplo sencillo de una aplicación que utiliza el módulo. En este caso, el sistema almacena la información del módulo de adquisición y la traza constantemente en la pantalla. El programa también integra configuraciones disponibles por código, utilizando argumentos de línea de comandos. Por tanto, el sistema desarrollado se presenta como una poderosa herramienta para la adquisición de señales de ECG en línea, permitiendo el uso de la placa en interfaces hombre-máquina.

Palabras clave: Adquisición en línea; señal de ECG; interfaz hombre-máquina; lenguaje de programación Python.

1. Introduction

Biological signal processing is an important area of science which allows developing systems as assistive technologies [1] and other kinds of human-machine interfaces [2]. These systems convey information or controls from the user in the external world to the machine. Thus, it is important that these systems must generate an outcome in short intervals of time [3].
Signal detection analysis is important to automate aspects of a system, for example, carry out certain functions that the human operator would normally perform [4]. Oftentimes, the signal acquisition boards used in clinical applications works with offline acquisition concept. It means that the signal is recorded by half an hour, an hour or even hours and its processing is performed in a posterior moment, which is not appropriated to the current work.

Thus, this paper shows the development of system capable of adapting a commercial offline signal acquisition board for online signal acquisition. For that, an online data acquiring program was developed, written in the Python programming language [5]. This program is composed of modules, a collection of statements and functions that can be used by importing them into a program. This paper is organized as follows. In section 2 provides a methodology for the online signal acquisition of the implemented packet acquirer, its inner workings and communication schemes. In section 3 presents the results and discussions. Finally, in the last section, the conclusions were considered.

2. Materials and Methods

2.1 Board and Packets

The board used was the electrocardiogram (ECG) 12 simultaneous leads acquisition module MS-PCM-075-1206, from Andish Solutions [6]. It is composed of a 2,5 V, 10-bit A/D converter, band-pass filters with cutoff frequencies of 0.05 – 100 Hz (in diagnosis mode) or 0.5 - 40 Hz (in monitoring mode), as well as a notch filter, to prevent interference from the 60 Hz power line. The board communicates with computers using either serial or USB connection.

The data from the board is organized in packets. Each packet is a sequence of 20 bytes containing one sample of each board’s leads, as well as information about electrodes not connected and a Cyclic Redundancy Check (CRC) byte, for error checking. The transmission rate is 240 packets/s. The complete structure of a packet is shown in Figure 1.
Figure 1. The structure of a packet. The prefix 0x denotes a hexadecimal literal, while square brackets represent a byte.

Each byte of the packet is described as follow:

- **0x3a** – The synchronism byte of the transmission. Indicates the start of a packet. Hexadecimal representation of 58;
- **STATUS1** – Each of the seven least significant bits represent the status of an electrode in this order: RA, LA, LL, V1, V2, V3, V4. A value of 1 means the electrode is disconnected;
- **STATUS2** – Each of the two most significant bits represent the status of an electrode in this order: V5, V6. A value of 1 means the electrode is disconnected;
- **0x00** – Unused byte. Hexadecimal representation of 0;
- **HI1** – Since the A/D converter of the board represents values using 10 bits, a single byte is not enough to contain all the information. Therefore, it is split across two bytes. HI1 has the two most significant bits of the leads DI, DII, DIII and aVR, in this order;
- **HI2** – The two most significant bits of the leads aVL, aVF, V1 and V2, in this order;
- **HI3** – The two most significant bits of the leads V3, V4, V5 and V6;
- **DI - V6** – Starting from DI, up to V6, a byte named [X] represents the eight least significant bits of the lead X (e.g. [aVR] represents the eight least significant bits of the lead aVR);
- **CRC** – The XOR operation between all the bytes in the packet.
2.2 The Acquiring Module

The acquiring module is responsible for the acquisition and processing of the raw bytes from the board. The processing of the signal itself is done by other programs, built around the acquiring module. In this context, the task of processing means to apply the necessary bitwise operations to form the integer value of the sample using the most and least significant bits, which are transmitted separately in a packet. As an illustration to the operations being executed by the acquiring module, consider a hypothetical packet formed by the following bytes:

\[ Hl1 = 0b10011010 \]  \hspace{1cm} (1)

\[ DI = 0b01101111 \]  \hspace{1cm} (2)

Where

the prefix 0b denotes a binary literal. From the packet structure, we know the two most significant bits of \( Hl1 \) are the two most significant bits of \( DI \). An AND operation is used to extract them.

\[ DLm = Hl1 \land 0b11000000 \]  \hspace{1cm} (3)

\[ DLm = 0b10000000 \]  \hspace{1cm} (4)

Where \( DLm \)

is a byte whose two most significant bits are the most significant bits of \( DI \) and the other bits are zero, and \( \land \) is the logical AND operator.

Finally, shifting \( DLm \) two bits to the left and adding \( DI \) gives us the value of this sample for the lead \( DI \).

\[ DS = (DLm \ll 2) + DI \]  \hspace{1cm} (5)

\[ DS = 0b1001101111 \]  \hspace{1cm} (6)
Converted to decimal:

\[ DI_s = 623 \]  \hspace{1cm} (7)

Where \( DI_s \) is the value of the current sample of the lead \( DI \) and \( << \) is the logical shift left operator. The acquiring module creates one list for each lead to store this information and make it accessible to the programmer.

Each list holds a configurable number of samples from that lead. When a new sample is processed, it is inserted to the beginning of the list and the last sample (oldest one) is discarded. This means that the list has a fixed size at any time. The acquiring module does not store the information after it leaves the list. If this behavior is required, it must be implemented by the application using the module.

According to its usage, different settings for the acquiring module may be useful. For instance, the GUI script allows the user to select how many seconds the time axis of the ECG plot should have. Internally, this is done by increasing the size of the list that holds the samples for each lead. The complete list of options for the acquiring module is:

- **Data size** – It corresponds to the size of a lead’s list, or how many samples should it hold at any time. Imagine a program wants to process the last two seconds of data from the board. At a transmission rate of 240 packets/s, two seconds are represented by 480 samples. The program would then set the data size to be 480. Every time it requests the data from the acquiring module, it will receive a list of that size containing the most recent samples. When a new value is processed and integrates the list, the oldest one is discarded. The default size is 240 samples;

- **Sample size** – It corresponds to the number of packets requested from the serial port. Higher values preserve more information, but are not meant for online applications, as the processing will only occur after all packets are received. Extreme low values may cause the internal buffer memory to become full, as packets arrive faster than they can be taken, due to the
processing overhead between reads. The default size is 20 packets. Note that, when the bytes from the serial port are being read, there is no guarantee that all packets formed by those bytes are complete. For this reason, the acquiring module always requests one additional packet than the configured one, which ensures that at least sample size packets will be complete, and discards the one left over. Figure 2 illustrates this process. Increasing the sample size makes the proportion of packets discarded approach zero;

- **Port** – It corresponds to the COM port used for the serial communication between the board and the PC. In case no port is provided, the module will attempt to find a port whose device’s name matches the string “Silicon Labs CP210”, which is the name of the USB controller of the board used in this work. This setting is useful if multiple devices matching the string are connected to a single PC, in which case the programmer can force the module to communicate to the correct one;

- **Verbose** – Whether the module should output debug messages to the console. The default is false.

![Figure 2](image.png)

Source: prepared by the authors.

Although lossy, this data acquisition approach was chosen to prevent the acquisition module from having to request and test individual bytes from the serial port until the data size argument was satisfied. This would add a considerable processing overhead to the module, which is undesirable in an online process and could result in the overflow of the serial port’s buffer. As will be showcased later in this paper, the choice of an appropriate sample size minimizes the effect of the discarded packets on the final signal.
The ratio of packets lost due to discarding can be estimated by the following method. Considering the module at default settings:

\[ DS = 240 \text{ samples} \quad (8) \]

\[ SS = 20 \text{ packets} \quad (9) \]

Where:

\[ DS \] is the current data size and \[ SS \] is the current sample size.

Also,

\[ PS = 20 \text{ bytes} \quad (10) \]

Where:

\[ PS \] is the packet size, in bytes, determined by the board being used in this work.

The number of requests \[ N \] necessary to acquire \[ DS \] packets is, therefore:

\[ N = \frac{DS}{SS} \quad (11) \]

\[ N = \frac{240}{20} = 12 \quad (12) \]

For each request, one additional packet is requested to ensure at least sample size packets are complete.

This means that the number of discarded packets to acquire data size packets, \[ DP \], is the own number of requests.

\[ DP = N \quad (13) \]

\[ DP = 12 \quad (14) \]

The total number of packets required to acquire data size packets
TP, is the own data size, plus the number of packets discarded in the process, DP:

\[ TP = DS + DP = DS + N \]  \hspace{0.5cm} (15)

\[ TP = 240 + 12 = 252 \text{ packets} \]  \hspace{0.5cm} (16)

Finally, the ratio of packets lost due to discarding R is:

\[ R = \frac{DP}{TP} \]  \hspace{0.5cm} (17)

Replacing (13) and (15) in (17) gives:

\[ R = \frac{N}{N + DS} \]  \hspace{0.5cm} (18)

Lastly, replacing (11) in (18) gives:

\[ R = \frac{1}{1 + SS} \]  \hspace{0.5cm} (19)

At default settings:

\[ R = \frac{1}{1 + 20} = 4.76\% \]  \hspace{0.5cm} (20)

For non-default settings, (19) can be used to calculate R, given SS.

The following discussion outlines the communication between the acquiring module and other programs that make use of it. Any application using the acquisition module, including the GUI script, communicates with it using system memory. This was done to prevent the slow nature of, for example, writing the data to a file. Applications using the module initialize it programmatically using the following statements:
Figure 3. Initialization of the acquiring module programmatically, using default settings.

```python
acquiringThread = acquiring.acquiringThread()
acquiringThread.start()
```

Source: prepared by the authors.

This will start a new acquiring thread at default settings, assigning it to the variable `acquiringThread`. For non-default settings, use the keyword arguments `data_size`, `sample_size`, `port` and `verbose` in the constructor function. For instance,

Figure 4. Initialization of the acquiring module programmatically, using custom settings.

```python
acquiringThread = acquiring.AcquireThread(
    sample_size = 30,
    port = 'COM3'
)
```

Source: prepared by the authors.

To get the data from the board, use the following statement, replacing `lead_index` by the index of the lead you want the data to come from. Indexes start at 0 for DI and end at 11 for V6.

Figure 5. Accessing the data from the acquiring module using the lead’s index.

```python
data = acquiringThread.data[lead_index]
```

Source: prepared by the authors.

This assigns a list of the latest `data_size` samples for the lead whose index is `lead_index` to the variable `data`. Please note that the variable `data` is not updated automatically after this process, requiring this statement to be executed as many times as necessary. In the simplest case, a loop like the one presented in Figure 6 can be used for this. The outer while loop repeats indefinitely. Every iteration, the inner for loop goes through every possible lead index (0 to 11), acquires the data for that lead, and, possibly, executes a processing algorithm with this data.
Figure 6. Simple example of accessing the data from the acquiring module using the lead’s index.

```plaintext
repeat indefinitely
while True:
    for i in range(len(acquiringThread.data)):
        update data with the current lead's data
        data = acquiringThread.data[i]
        plotting algorithm, for instance,
        plot(data)
```

Source: prepared by the authors.

Figure 7 shows a block diagram for a complete program structure. The sample processing loop is the one responsible for processing the raw bytes from the board, as described earlier. The data processing loop then uses this data to accomplish the application’s task, which, in the case of the GUI script, is plotting it. Note how the interaction between the acquiring thread and the application is bidirectional (the thread transmits data one way and receives settings the other way), but their loop flow is independent. This prevents the sample processing loop from being delayed by a long data processing loop. On a higher level, this means the conversion of the board from offline to online acquisition is independent from the application using the data, which makes it very modular.

Figure 7. Block diagram for a complete program structure and the interactions between its parts.

Source: prepared by the authors.

2.3 The GUI Script

The GUI script plots the content of each ECG lead to the screen at a constant refresh rate of 4 frames per second (FPS), making it possible to see the updated data being received from the board. It is executed by calling it from the command-line with appropriate arguments. In this paper, it was used to verify the functionality of the acquiring module against a known simulated ECG signal. The script integrates all the acquiring module settings as command-line arguments. This allows a quick prototype of different acquisition settings, as well as an interface for non-programmer users. The complete list of arguments for the GUI script is:
- `-t` or `--time` – It corresponds to the length, in seconds, of the time axis of the plot. Internally, it modifies the data size. Defaults to 1 second. For instance, `gui.py -t 2`;
- `-s` or `--samples` – It sets the acquiring module sample size. For instance, `gui.py -s 40`;
- `--port` – It sets the serial port used by the acquiring module. For instance, `gui.py --port COM3`;
- `--v` or `--verbose` – It sets the acquiring module’s verbose level. For instance, `gui.py --v`.

Figure 8. Plot for a 30 bpm ECG signal made using the GUI script. The acquiring module’s settings were adjusted to the default values. The lead’s titles (from top-left to bottom-right) are DI, DII, DIII, aVR, AVL, aVF and V1 up to V6.

Source: prepared by the authors.
3. Results and Discussions

Figure 8 and Figure 9 show two ECG signals registered by the module. The signals were generated using an ECG simulator at two different rhythms, 30 and 120 beats per minute (bpm), respectively.

The leads V1 to V6 were disconnected, as the simulator handles only 5 channels. The board treats these as having a constant value of 511 (Note, the A/D converter range is 0 to 1023).

Considering that the horizontal axis (time) length of both figures is set to 1 second, the heart rate measured in both figures matches the settings from the simulator. The signal plots showed in Figure 8 and Figure 9 demonstrate that the module works as expected.
Although the signal used in this paper was an ECG signal, this commercial board can be used to acquire EMG and EOG signals in online mode, since their amplitudes are similar to the ECG signal. This allows the usage of the set board-acquiring module for application using biological signals. For future works, the software application could be used for tasks such as blink detection and muscle movements evaluation in the development of assistive technologies and Human-Machine Interface (HMI).

4. Conclusions

This work presented a software developed in Python programming language applied to a commercial signal acquisition board. This program captures the signal in short intervals of time and makes them available for registration or other desired operations, like make a plot curve. For that, the communication protocol between the board and the computer is observed and the data packets are evaluated to compose the segmented signal.

The software is organized in two structures, the acquiring module and the GUI script. The first one is responsible for the acquisition and processing of the raw bytes from the board. The second structure is set up to perform the desired operation. In the case of this paper, the GUI script was written to plot the content of each ECG lead to the screen at a constant refresh rate of 4 frames per second.

Thus, the developed program allows the commercial board to be used for capturing biological signals of the same magnitude of ECG signal. This extends the usability of this board and allows the development of human-machine interaction tools with the use of other biosignals. Moreover, it could be used as a tool of interaction between a user of these technologies and the world around them.

As future work is proposed to deal with the ECG signals by the Internet of Things media and then the current system would be useful for diagnostic processes as described in [9-11]. Moreover, it would be creating a dataset with the features extracted from the electrocardiogram signals [12]. Feature extraction [13] is an important procedure in biomedical signal analysis and the developed system can assisted in this task.
Acknowledgment

The authors would like to thank to the Instituto Federal do Espírito Santo (Ifes) for supporting the development of this work. This study was partially supported by Edital PRPPG Pibic/Pivic in Ifes, number of project PJ4373/PT6811.

REFERENCES


