BIOSIGNALS: General BVP, GSR, temperature (TMP) and accelerometers (ACC) experimentation in real-time, an open source software for physiological real-time experimental designs

September 16, 2020

1 Hardware

The following device has been used for the BIOSIGNALS:

1. Empatica E4 wristband by a Massachusetts Institute of Technology (MIT) spin-off.

1.1 Empatica E4

The E4 is a medical-grade wearable device that offers real-time physiological data acquisition. It measures blood volume pulse, from which heart rate variability can be derived, captures motion-based activity with a three-axis accelerometer, measures the constantly fluctuating changes in certain electrical properties of the skin (GSR), and reads peripheral skin temperature. It allows the synchronization of such signals by the use of an internal clock. The GSR and temperature sensors have a sampling rate of $4Hz$ while the accelerometers and the BVP signal are measured at $64Hz$. The combination of the GSR and BVP sensors enables to simultaneously measure the balance of sympathetic nervous system activity and heart rate. More detailed information can be found at `https://www.empatica.com/en-eu/research/e4/`.

2 Software

All the code has been developed with pure python libraries in an object-oriented paradigm. The software engineering process was carried out taking into account cohesion and coupling for proper modularization. This allows the applications to be modular, scalable, and easy to maintain; in fact, this is a key aspect of

any scientific tool to allow researchers to make modifications and to fulfill the specific requirements of each experimental scenario.

The philosophy of BIOSIGNALS application is based on a supervised machine learning approach and therefore it offers two modes of interaction:

- The first allows for real-time acquisition and processing to generate a database and build models.
- The second provides online signal processing using pre-trained models to classify physiological patterns.

The proposed application is versatile and easily adaptable to different experimental designs while maintaining high-performance real-time signal processing. Aware that pattern recognition is in constant development, BIOSIGNALS application offers the option of importing python external scripts, which must have a predefined structure, to include self-developed machine learning methodologies. Besides, as experimental designs require event synchronization, a TCP/IP interface has been provided. BIOSIGNALS application is expected to be accessible to the entire scientific community, providing a resourceful tool for experimental paradigms of human behavior, which encompasses the following functionalities:

- Real-time acquisition and visualization of physiological signals.
- Trigger synchronization by a TCP/IP interface that allows start/stop the recordings remotely.
- Recording of data on European Data Format (EDF) for physiological signals.
- Online behavior labeling interface whose labels are synchronized and stored in the EDF files.
- Online signal processing with self-developed methodologies through the possibility of importing external python scripts.

2.1 Biosignals application

Title: General BVP, GSR, temperature (TMP) and accelerometers (ACC) experimentation in real-time, an open source software for physiological real-time experimental designs

Authors: Mikel Val Calvo

URL: `https://github.com/mikelval82/Biosignals`

License: GNU General Public License v3.0

DOI: 10.5281/zenodo.3759262

This application has been built to further the research on the study of physiological signals, such as BVP, GSR, temperature (TMP) and accelerometers (ACC), related to human behavior, with the use of the Empatica E4 wristband. Figure 1 shows the layout of the BIOSIGNALS interface. Each section has been numbered to properly explain the role of each of the components:

1. Set Users: Allows for setting the name of the user file where acquired data will be stored.
2. Load Script: For loading external python scripts developed by users for specific experiments.
3. Empatica Server Connect/Disconnect: Enables the connection and disconnection between the Empatica Server and the Biosignals.
4. Empatica ID: A spin box to specify to which Empatica E4 device to connect.
5. Refresh: Requests to the Empatica Server the ID of Empatica E4 devices available.
6. Connect: Enables the connection and disconnection between the specified device through the Empatica Server and the Biosignals driver.
7. BVP Window size (seconds): A Spin Box to set the window size of the BVP signal.
8. GSR Window size (seconds): A Spin Box to set the window size of the GSR signal.

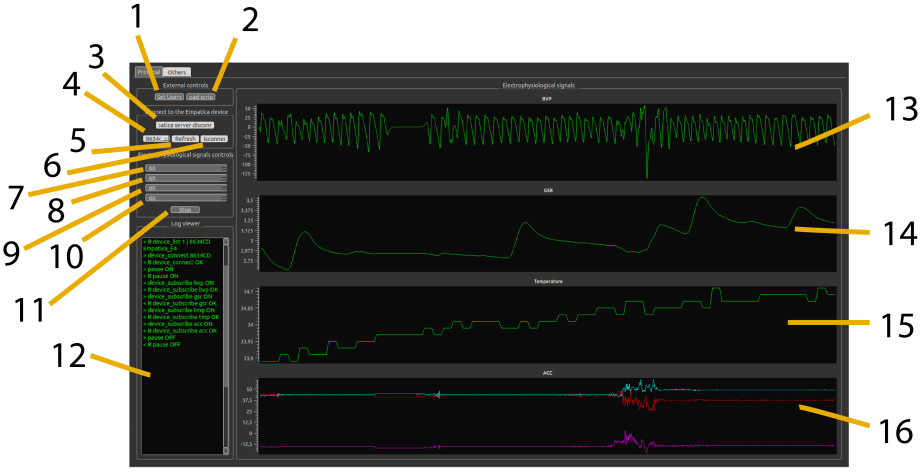

Figure 1: BIOSIGNALS: General physiological experimentation in real-time, an open source software for real-time physiological signals acquisition and processing with the Empatica E4 wristband.

9. TMP Window size (seconds): A Spin Box to set the window size of the TMP signal.
10. ACC Window size (seconds): A Spin Box to set the window size of the ACC signals.
11. Start/Stop: A button to start/stop real-time visualization.
12. Log viewer: Shows information regarding the internal state of the application.
13. BVP Long Term View: Allows the visualization in real-time of acquired BVP signals.
14. GSR Long Term View: Allows the visualization in real-time of acquired GSR signals.
15. TMP Long Term View: Allows the visualization in real-time of acquired TMP signals.
16. ACC Long Term View: Allows the visualization in real-time of acquired ACC signals.

2.2 Software design principles

As mentioned before, the design principles have been followed taking into account cohesion and coupling for proper modularization. Therefore, a high-level description is provided that allows us to understand the design principles. As can be noted in figure 2 where the high-level flow diagram scheme is detailed.

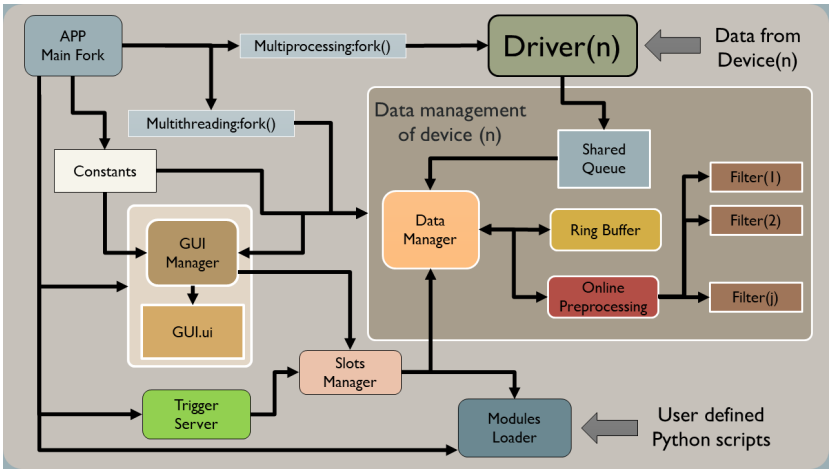

Figure 2: Flow diagram.

- APP: This class serves as the main software module that builds the application. It has been designed using PyQt5, a Python binding of the cross-platform graphical user interface (GUI) toolkit Qt, implemented as a Python plug-in. It launches all the other modules and forks several processes. First, the APP main fork. Second, a Driver is forked for each device, using the multiprocessing python library. Finally, it forks a thread for each Data Manager, using the multithreading python library.

- Constants: Sets the whole constants used along with the application. Thus facilitating the specification of such general variables in a unique module. All adjustable parameters are set in this object.

- Queue: This Queue follows the FIFO rule.

- Ring Buffer: Stores the last values for each sampled source of information. The fixed size is defined by default in the Constants but it can also be dynamically modified during the GUI interaction through the 'Window size' Spin Box. Each time the fixed size is modified, the Ring Buffer is emptied.

- Driver: Meets the requirements of the real-time acquisition. It inherits from multiprocessing python library. It offers the interface between the APP and the hardware. Inter-process communication is facilitated by Queue. Once the fork starts running, iterates acquiring samples that are indefinitely queued.

- Data Manager: It is defined as an interface between the GUI and the Driver to properly separate the management of the acquired data. It inherits from a multithreading python library. It has access to the shared Queue so its role consist on extracting iteratively samples from the queue to insert them on a Ring Buffer.

- GUI Manager: Offers the management of each of the components of the GUI.

- GUI.ui: Implements the APP graphics. It has been designed using QtDesigner, the Qt tool for designing and building GUIs.

- Trigger Server: A TCP/IP server for event synchronization. It can receive client connections and handles each request by notifying the Slots Manager using a callback function.

- Slots Manager: The purpose of this module is to bring the application to the versatility of adding a set of callbacks that allow synchronizing the overall logic.

- Online Filtering: This module has been implemented as an interface for a set of real-time signal processing methods.

Algorithm 1 Imported python scripts required structure.

```
# -- Add imports --
# import numpy as np

# -- Class definition --
class pipeline():
  ''' Predefined structure.'''
  def __init__(self, app):
      # -- initialize variables --
      self.log = app.log

  def run(self):
      # -- code desired behaviour --
      self.log.update_text('Start␣computing␣the␣user␣module.')
```

- Modules Loader: The main purpose of this module is to expand the versatility of the APP for a wide range of research scenarios. It is required that users code the scripts with a predefined structure as shown in 1. The module will receive a reference to the APP instance so total access to the underlying object instances is offered.

Main workflow scheme: Once BIOSIGNALS app is launched it can work as follows; Data Manager is threaded and starts iteratively trying to extract values from the queue. The thread runs a loop that is controlled by a logical condition. The iteration process cannot acquire data if the queue is empty. Then, the Driver is forked and automatically detects the corresponding device to connect with. Following, the driver starts iteratively stacking data into the Queue from the hardware. Both the thread and the fork form a consumer-producer system. The Ring Buffer acts as a short term memory. The Data Manager fills the Ring Buffer each time a new value is obtained from the Queue. Once the Ring Buffer is full, it starts overwriting the oldest data and so on. At this point, two stages are provided:

- Stage 1: Real-time visualization can be performed by pushing the start button. It does not imply user data storage. The start button has been designed to test the quality of the signal acquisition. The start button state will be changed, so pushing it again will stop visualization. Parameter settings (user filename, filter order, window size, online artifact removal method, and so on) should be done on this stage.
- Stage 2: Once the signals are properly acquired, the Trigger button can be pushed that initializes a TCP/IP server. The application only stores data in between a pair of client received requests. The first request must send a string containing 'start' message to init the recording. The last request

must send a string containing 'stop' message. In between, any other string can be sent to allow codification of events that will be stored in the EDF file. It is expected that one request indicates the start instant for event synchronization while the second, the stop instant. For the following, two pairs of events form a trial. During the trial duration, the Ring Buffer slots a callback to the Data Manager whenever an amount of data equal to the window size is fulfilled, that is, a sample is generated. At the end of the event, the Data Manager permanently stores acquired samples in the users' file with its corresponding metadata. Data Manager counts the number of samples generated and the number of trials, other metadata values are obtained from the Constants module.

Versatile workflow scheme Modules Loader allows the definition of own produced scripts. Each imported script must follow a predefined structure. The versatility comes due to the reference given to the imported script to the main object instance. Thus, different scenarios could be provided through this component.

- Offline scenario: The user could experiment with the main workflow scheme. After collecting the required data, a script could be loaded containing the coding for an offline analysis, i.e., training a machine learning model and store it on disk for future use.
- Online scenario: The user, could perform online computation thanks to the reference given, having access to all the underlying object instances. Access to the Data Manager to acquire EEG samples online is therefore provided. Moreover, applying OAR methods or even using pre-trained machine-learning models for online predictions can be also easily implemented.

The main highlights of the application consist of its versatility: First, modularizing the interaction between the GUI manager and the Driver, which allows the embedding of drivers for other acquisition systems. Second, the separation between the GUI and the logic, where the former allows future modifications and performance of the GUI components, without altering the logic, and vice-versa; Finally, the Modules Loader, further expands the versatility, by offering the flexibility of incorporating self-developed scripts for specific scenarios and workflow schemes. The proposed application IS versatile and easily adaptable to different experimental scenarios while maintaining high-performance signal processing in real-time.