FlightStone Documentation

Release 1.0

A. S. Mateus
J. O. Pacheco

Nov 24, 2017

Contents:

CHAPTER 1

Installation Guide

To execute the application, there are a minimum set of steps you should execute, basically we need to have an adequate working environment. First we will create a virtual environment, if you have not installed `virtualenv` do it now:

```
pip install virtualenv
```

The project has been tested with both python 2.X and 3.X, no issues have been found, so use the one you prefer. To keep things standard it is advised that all your virtual environments lay in the same directory so, lets create create one and place our virtual environment in it:

```
cd
mkdir pyenvs
cd pyenvs
virtualenv vip
```

Now activate it by sourcing the `activate` script in the `bin` folder of your virtual enviroment.

```
cd vip/bin
source activate
```

With this we have a working environment, so lets install the dependencies of our application. The application dependencies are listed below and the installation commands using `pip` are provided, it is up to you if you wish to use another method:

- PyGames `pip install pygame`
- PyLSL `pip install pylsl`
- SciPy `pip install scipy`
- SciKit Learn `pip install scikit-learn`
- Sphinx `pip install sphinx` (Optional, if you want to build the documentation)

Finally clone the latest version of our code in a directory of your preference:

```
cd <path-to-container-directory>
git clone https://github.com/JorgeluissilvaC/VIPproject.git
```

Normally this is all you need to do. The application provides different interaction modes such as demos and two games for training and testing, this information is explained in detail in the *Tutorials* section. However, if you simply want to test the correct execution of the program, try starting the **car game**. Assuming you are in the directory you cloned from github:

```
cd game/cars_game
python test.py
```

If you want to build the documentation, for any reason you may have, you should switch to the documentation branch, as it has the latest documentation version:

```
git checkout docs
```

If you face trouble saying the branch does not exist try fetching from Github via `git fetch` command. Then simply run the following command in the `docs` directory:

```
make html
```

CHAPTER 2

Theoretical Background

2.1 Overall system considerations

The BCI system is focused in identifying two types of mental states that modulate the alpha rhythms present in the human brain. The alpha rhythms are brain rhythms that range in the 8-13 Hz frequency band, they are produced mainly by activities in the thalamus (Figure 1). Alpha rhythms are predominant (in regards to other rhythms, such as the beta rhythm) when the person has the eyes closed and is in a relaxed state, they are also important in network coordination and communication. Alpha rhythms (or waves) can be detected either by electroencephalography (EEG) or magnetoencephalography (MEG); due to acquisition easiness, a commercial electroencephalography (EEG) device is used.

Fig. 2.1: Figure 1. Thalamus view

Given the particularity of alpha waves for detecting relaxed-to-stressed states, our system we have established this two activities as the ones the user must perform. The overall aim is, thus, design a system capable of determining in each state the user is by analysing the alpha waves of the subject, in real time. A second objective is desired: the

user is asked, during the stressed state, to think in one of two actions, if possible opposite actions (like moving left, then right); the action the user want to perform will then modify the spectral disposition of the wave, allowing us to characterize the alpha waves for the two actions, and, with that information let the user control an object on-screen.

To keep matter short, the aim is: Acquire alpha waves with a EEG system Classify waves between relaxed and stressed states using a training software (that we will develop) Classify stressed states between two different actions using a training software (that we will develop) Construct a classifier with the classified data to predict future instances Construct a game-like environment where the two previously classified actions can be tested

2.2 Alpha waves acquisition

Alpha waves will be acquired via Starstim EEG device with 14 acquisition probes. The users are subjected to a training interface that iterates between relaxation and stress stages in a periodic manner, in each stage the user is requested the effect the respective action. In Figure 2 you can see the overall process for data acquisition. Each stage is repeated a large number of times, so that their contribution is statistically significant to the classifier.

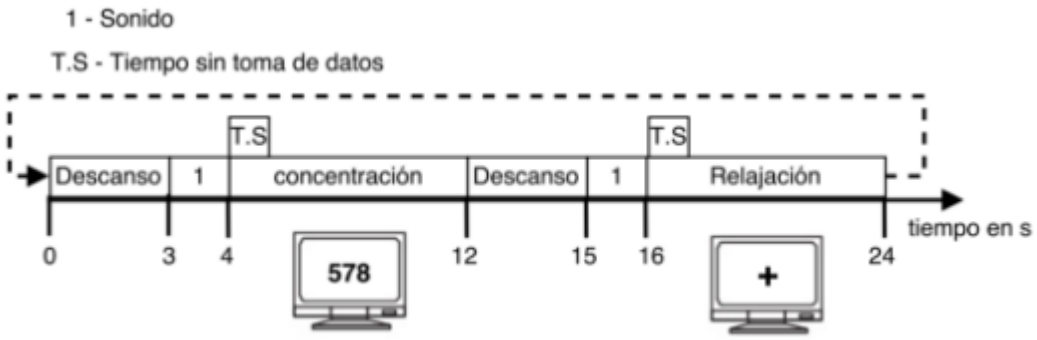

Fig. 2.2: Figure 1. Process overview

Each probe samples the voltage at a rate of 500Hz (good enough, given the frequency of the alpha waves). Thus we have 14 channels of voltage-time information, however, to characterize the alpha waves per channel the spectral information is more useful (this spectrum show the relation of relative amplitude against frequency value). Non desired frequency may be present so, a low-pass filter is applied. Some transforms to acquire the spectrum information are available: most popular one is the Fourier transform, and the other is the Hilbert transform (the main difference is that the Hilbert transform relays less data, as it only takes into account the envelope of the wave). Thus we have, power-vs-frequency data in a time interval (for each probe); we then generate a vector per probe where the power per frequency is averaged throughout the interval of acquisition (5 seconds), which results, as a whole, in a two dimensional data array. Adding all the repetitions together results in a tridimensional data array, as shown in Figure 3

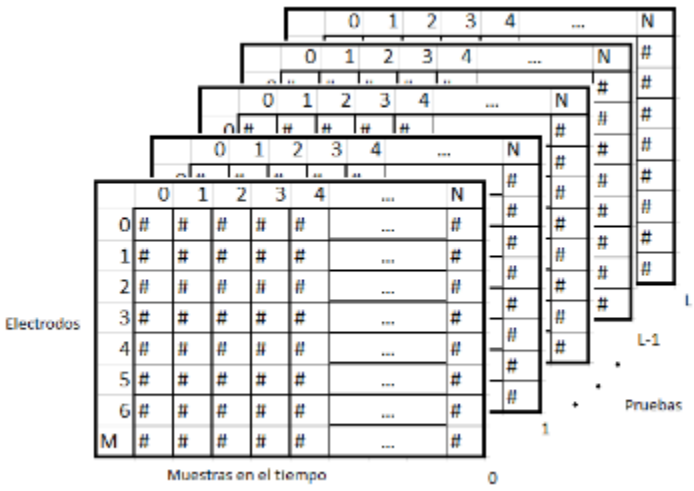

Fig. 2.3: Figure 3. Matrix array for classification

CHAPTER 3

VIP-BCI Component Description

The application structure is really simple and can be generalized as a classifing service coupled with an interactive user interface. Given that the classifier is trained with a different data set according to the user, a user management entity should be implemented, so that the correct data files are loaded when training. The interaction section is logically independant from the core section of classification, the main link is a set of functions that are:

```
receiveClassifierDecision()
move(decision)
```

The first function links the game and the classifier, retreiving the decision, 0 or 1, as the decision is binary. The second function implements the movement according to the specific game. In this manner both, display and logic are separated. A system that implements such a structure is shown in Figure 1.

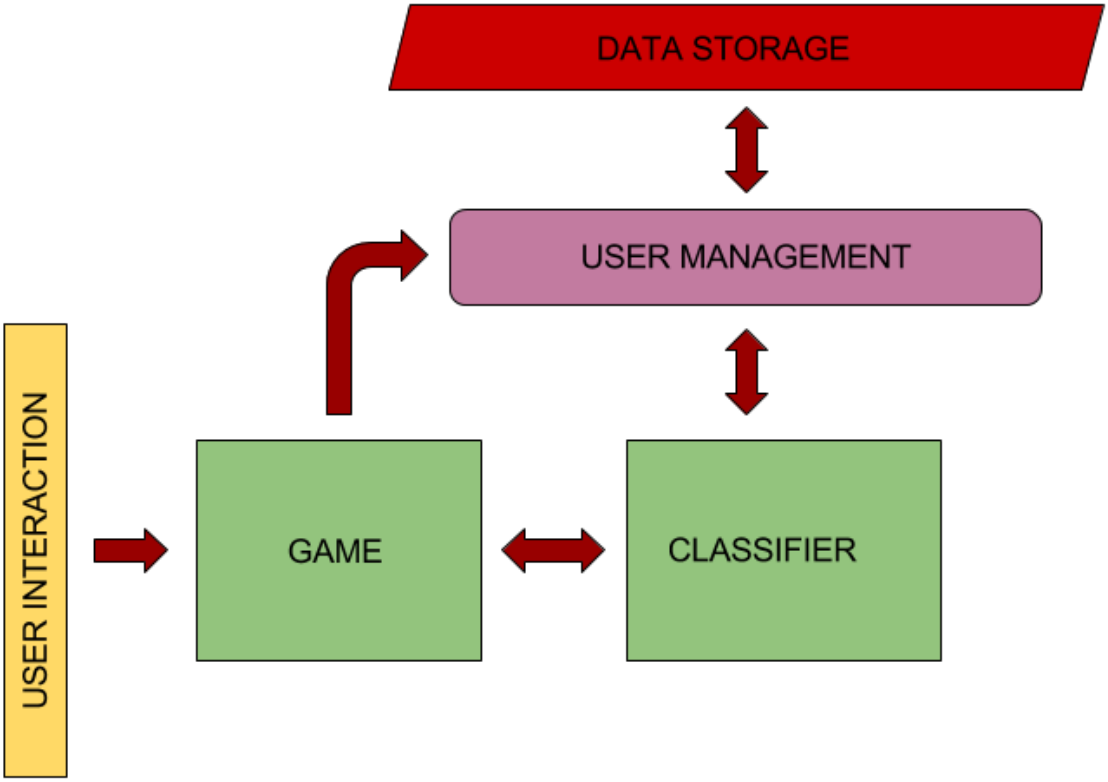

Fig. 3.1: Figure 1. Components of the application

CHAPTER 4

VIP-BCI Tutorials

CREA UNA SESIÓN DE ENTRENAMIENTO

CREA UNA SESIÓN DE TESTING

CHAPTER 5

VIP-BCI Code Explorer

This Code Explorer is distributed in sections, each section corresponds to a game type; only relatively relevant functions are explained, the way they interact with other functions is also specified.

5.1 Car Game

The car game is shown in Figure 1. It is supposed to allow horizontal movements to right and left.

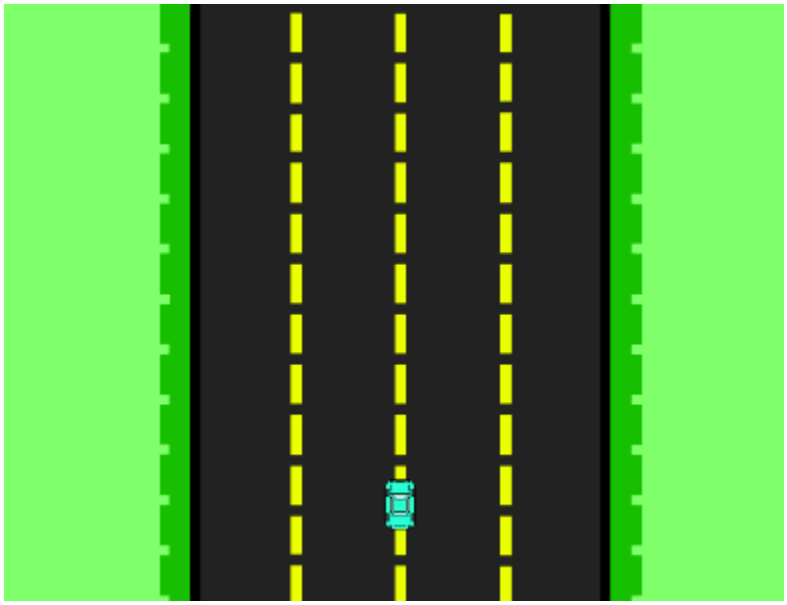

Fig. 5.1: Figure 1. Components of the application

The entry point of the application is through a class named *App* and is composed as follows. The code is ilustrative and minimized, actual code is much larger.

```
App():
    def __init__(ID=nombre, trials=n, fps=60, **args):
        # Init UI components
        # Init thread instances
        ...

    def run():
        # Main loop of the application
        # the self.fps variable controls the speed of the motion
        ...

    def training():
        # Init the retreiving data cycle
        ...

    def game():
        # Init the game instance, start classification thread to guess the car
        # direction
        ...

    def draw_text():
        # display notification message
        ...

    def getData():
        # Check if the Starstim device is available
        # Start buffering data
        # This is executed in a separated thread
        # This runs in the training mode
        ...

    def AutoUpdateBuffer():
        # Check if the Starstim device is available
        # Start buffering data
        # This is executed in a separated thread
        # This runs in the game mode
        ...

    def saveData(ID):
        # Stores the retreived data
        ...

    def processing():
        # Transforms the retreived data to a format that can be analyzed
        ...
```

As can be seen, the *App* class is self contained, as it implements functions of both classification and game interaction. In the following, the functions will be explained in detail.

The *init* function set ups the class variables and calls the *run* function that starts the main application, that is the menu selection.

```
def __init__(self, ID=nombre, trials=n, fps=60, **args):
    # Init UI components
    # Init thread instances
    self.fps = fps
    self.ID = ID
    self.trials = trials
```

```
    self.run()
```

The *run* function determines whether the user wants to train or to play the game, the decision is interactive via the application menu.

```
def run():
    # Main loop of the application
    # the self.fps variable controls the speed of the motion

    while(1):
        if self.method == 'training':
            self.training()
        elif self.method == 'game':
            self.game()

    # User interaction
    ...

    # Car position updating
    self.updateCar()
```

The training function triggers the data acquisition process explained in the Theory section, this in turn means that a separate thread of execution is launched were the connection with Starstim is tested; that involves *process* and *getData* functions.

```
def training():
    # Init the retreiving data cycle
    self.initAcquisition()

    count = 0
    while count < self.trials:
        self.cycle()
        count += 1
```

Hello, navigating this documentation is fairly straight foward, you can see the component tree in the navigation bar at the left. Each description unit serves a specific porpuse, perhaps you will find particularly interesting the *Installation* and *Tutorials* sections, as the information there is enough to give you working knowledge of the platform (installing, training, testing and extending it for your personal use cases). This is, however, surface knowledge; to delve into the advanced tutorials you will need to look into the *Component Description* section, to have a notion of the organization of the application. Further more, if you want to modify, remove or add functionality, exploring the source code is mandatory. *Theory* section and *Statistics and Performance* section is not required, but are valuable assets if you ever want to replicate some of the things we do here.

CHAPTER 6

General Objective

Our principal purpose is to build brain interfaces that can be used by general people in two applications: gaming and robotic hand control. The brain-computer interface has to establish a communication with the application without the employment muscles. To use the software knowledge about the usage of electroencephalographic (EEG) sensors, signal processing, machine learning and programming skills is required.

CHAPTER 7

Limitations

This project is subject to several limitations, the first one comes from the motivation itself, and it is that the system must be non-invasive and should rely only brain signals that reach the cerebral cortex. Also, due to the limited amount of sensors for data acquisition only 8 sections of said cortex can be explored at the same time. The type of signal that can be extracted with this limitations enables us to differentiate between two states: whether the region is consideration is under stress or it is relaxed. See the *Theory* section for further information.

CHAPTER 8

Objectives

So, what can you expect from our work, well here are the objectives we set our minds in:

- Use the data acquired with each sensor, determine the stress state of each region of the brain in a given time (by measuring power in frequency ranges)
- Train a classifier that takes the said data and determines the action the user is thinking about (only two actions are possible)
- Build different gaming platforms to test the classifying algorithm

CHAPTER 9

Specific Capabilities

With this application you can do basically the following things:

- Train the system to recognize LEFT and RIGHT brain signals of a specific individual
- Train the system to recognize whether the user is under mental concentration or is relaxed
- Play to different games: a box moving game and a car race game with your brain (using the data acquired in the training stages)