POLYTECHNIC OF TURIN

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING (DAUIN)

Master degree course in Computer Engineering

Master Degree Thesis

# Tele-operated car control using Brain Computer Interface

**Academic advisor**
Prof. Massimo Violante

**Candidates**
Filippo Cupolo
Gianluca D'Alleo

**Internship Tutor**
Dott. Vincenzo Campanale

A.A. 2019/2020

# Contents

## Abstract

In this paper, it is presented an approach to control a car remotely with a Brain Computer Interface (BCI). To achieve this goal, it is used a commercial BCI that needed to be connected to a teleoperated car. The accelerator/brake pedals and the steering wheel of the teleoperated car can be remotely controlled through a UDP/IP connection. The BCI is composed by an Emotiv EPOC+ EEG headset that is able to read the brainwaves of the user and train a statistical classifier to classify mental states. Then, these mental states or facial expressions can be translated into commands for the car. Brain-computer interfaces represent a range of acknowledged technologies that translate brain activity into computer commands. The brain is the main processor in giving orders to the human body to perform physical activities. With technological advances, today's brain signals can be used as commands to control electronic devices. For example, disabled people can use their brain signals to give commands to move a wheelchair or operate a mobile device or even to drive a car. This research aims to develop and evaluate a BCI control application for certain assistive technologies that can be used for remote telepresence or remote driving. It has been implemented a scenario to test the usability of the BCI for controlling a tele-operated car. In the scenario, the car is completely brain controlled, using different brain patterns for steering and throttle/brake.

### Main Phases

- Create a dataset of facial expressions/mental commands and train the Brain-Computer Interface in order to get clearer and more significant information.
- Use of Cortex API to communicate with the Emotiv SW and get information about facial expressions/mental commands.
- Creation of a simulation scenario using ROS and some tools as Autoware or CarDemo.
- Translate the mental commands into significant directives to create the possibility to send/receive
- UDP packages for remote driving.
- Develop a SW application (C++, Qt framework) that receivs data from the Emotiv SW, converts these data into commands for the tele-operated car and sends the commands.

# Abstract - Italian version

In questo documento, viene presentato un tentativo di controllo remoto di un autoveicolo con un'interfaccia neurale (BCI). Per raggiungere questo scopo, viene usata un'interfaccia neurale commerciale considerata a basso costo da connettere al veicolo teleoperato. I commandi di accelerazione, frenata, rotazione dello sterzo del veicolo possono essere contrallati remotamente grazie all'utilizzo di una connessione UDP/IP. L'interfaccia neurale è composta dall'headset Emotiv EPOC+ EEG, capace di leggere e interpretare le onde cerebrali dell'utente e allenare un classificatore statistico per classificare gli stati mentali. Inoltre, questi stati mentali o queste espressioni facciali possono essere tradotte in comandi reali per il veicolo. Le interfacce neurali rappresentano una gamma di tecnologie conosciute per tradurre l'attività cerebrale in commandi informatici. Il cervello è il processore principale per il corpo umano nel dare ordini per compiere attività fisiche. Grazie allo sviluppo tecnologico in continua crescita, oggi è possibile utilizzare i segnali del cervello umano per controllare dispotivi elettronici. Per esempio, le persone che soffrono di una disabilità posso stimolare le loro onde cerebrali per impartire comandi al fine di muovere una sedie a rotelle, utilizzare un telefonino o addirittura guidare un'auto.

Lo scopo della ricerca, di seguito presentata, è sviluppare e valutare un'applicazione di controllo basata su un'interfaccia neurale per specifiche tecnologie assistive che possono essere usate per una telepresenza remota o per guida remota. Si andrà a realizzare uno scenario per testare l'usabilità di un'interfaccia neurale nel controllo di un'autoveicolo teleoperato; in questo scenario il veicolo è completamente controllato dai segnali ricevuti dal cervello dell'utente con l'utilizzo di differenti schemi per quelli che sono i comandi base di una comune guida di un veicolo.

## Fasi principali

- Creazione di un dataset di espressioni facciali e/o commandi mentali e allenare la BCI per ottenere un'informazione più pulita e più significativa.
- Uso di Cortex API per comunicare con il software Emotiv e ottenere le informazioni sulle espressioni facciali e sui comandi mentali.
- Creazione di uno scenario di simulazione basato su Ros e altri software come Autoware o CarDemo.

- Traduzione di espressioni facciali e/o commandi mentali in istruzioni significanti.
- Pacchetti UDP per la guida remota.
- Sviluppo di un'applicazione (C++, Qt framework) che riceve i dati dal software Emotiv, li converte in comandi codificati adeguatamente e li invia al veicolo teleoperato.

# Chapter 1

# Introduction

Systems based on electroencephalogram (EEG) signals have been the most widely used in the field of brain-computer interface (BCI) in the last years. Several applications have been proposed in literature, in particular in the study of emotions, from games to rehabilitation systems, in an attempt to address the new users' needs. At the same time, it is possible to record brain activity, in real time and to discover patterns to connect it with emotional states. Until recently, devices for recording EEG signals were too expensive for an end user, as there are currently several low-cost alternatives on the market. The most sophisticated of these low-cost devices is the Emotiv EPOC headset. Some studies have reported that this device is suitable for customers in terms of performance. Will drivers also be able to spark the ignition, turn left and right, and accelerate or decelerate simply using their brain waves? This is the goal of the thesis.

In this work, the introductive chapter presents a brief state of art of BCI and few examples of real test cases. In the second chapter, it is described the goal of the thesis and the first approach with the BCI and with the system, understanding how they work. In the third chapter, softwares used are described and the reasons why they are chosen; there is also a description of the first concept of the application developed mainly based on facial expressions. The fourth chapter concerns to the real development of the application but also to the phases of training of the BCI introducing the mental commands; tests on simulation and on track are described. In the fifth chapter, it is described the last version of the application with new user interface and new features introduced; tests on simulation and on track of the last application are described. In the sixth and last chapter, the conclusions and new ideas for future works are analysed.

## 1.1 State of art

Non-invasive brain-computer interfaces (BCI) have undergone enormous development in less than two decades in terms of performance and variety of applications. In reality, a non-invasive BCI can be defined as a device that allows communication without movement: a direct communication path between a human being (or an animal) and an external device.

Given its portability, its relative low cost and its high temporal resolution, compared to other non-invasive methods (MEG, fMRI, fNIRS), electroencephalography (EEG) is widely used for BCIs [1]. As reviewed by [2], since the beginning, a lot of applications have emerged about BCI. They were used for communication and control, in particular they allowed to control a mouse or to use a web browser just with the thought. Another goal was the study of motor replacement or motor recovery whose main applications were grip activity [3] and wheelchair control [4]. In addition, BCIs have also been used to increase interactivity in games using the multi - mode between EEG signals and standard controls [5]. Finally, as regards emotions, their study in human-computer interaction has increased in recent years, due to the growing need for IT applications able to detect the mood of users [7]. Motivated by the everyday interaction between human beings, much of the research in this field has analyzed the detection of emotions from facial and vocal information.

In controlled situations, today's computer systems for sensing emotions based on this information are able to classify emotions with considerable accuracy [8]. However, emotions do not always manifest themselves through facial expressions or vocal information. Psychologists distinguish between physiological activation, behavioral expression and the conscious experience of emotions. Facial and vocal information is related only to behavioral expression, which can be consciously controlled and modified, and whose interpretation is often subjective. Thus, other approaches have been proposed to identify emotions that focus on different physiological information such as heart rate, skin conductance and pupil dilation [8][6]. A new field of research in the brain-computer affective interaction attempts to detect emotions with electroencephalograms (EEG) [9][10]. There have been several attempts to detect emotions based on EEGs but there is still little human consensus about their validity. Undoubtedly, one of the most important fields for BCI is the application related to rehabilitation. The systems to allow you to grasp objects with your hand were among the first rehabilitation devices. In 2006, a BCI commanded the activation and deactivation of the intake [3]. This approach allows the patient to grab objects thanks to the Electro Functional Stimulation (FES). From this first experiment, the system has been improved to increase the accuracy of the socket position based on the Evoked Potential in a Stationary State (SSVEP) [12].

At the same time, great attention has been paid to controlling a wheelchair. As described in [4], various strategies have been proposed: predefined positions or direct control. In a predefined position strategy, the patient chooses the place where he wants to go with a BCI. Then, thanks to the sensors and the shared control [11], all the low level controls are performed by the wheelchair control system to reach the desired position. On the other hand, in direct control, the orientation of the wheelchair and the choice to move forward is controlled directly and fairly continuously by the patient. Obviously, the second approach leads to more general problems being more flexible.

Recently, an original approach has been proposed regarding BCI based on gait rehabilitation [18]. Since the non-invasive BCIs are able to produce only high-level commands, the shared control is performed by a Central Scheme Generator (CPG), which generates a perfectly periodic step pattern whose speed-equivalent pace is controllable. In this theoretical test, a P300 paradigm was used on a treadmill to include an uncontrolled state detection, ie when the subject does not want to change his state avoiding looking at the screen, and the results on four subjects demonstrated the feasibility of this application.

In [18], a possible extension of the prosthesis / orthosis of the lower limbs has been proposed using specific VUZIX glasses, emerging and well designed, for augmented reality (Vuzix, Rochester, NY, USA). This can circumvent the practical problems arising from the use of the P300 paradigm in terms of screen portability by displaying stimuli on a semi-transparent module containing all the essential hardware elements.

Although all these aspects seem promising, one of the main drawbacks of the EEG system is its high cost to customers. This is why several commercial EEG devices are now available as NeuroSky, Mindflex, Emotiv EPOC, etc. [14]. Based on usability [14], the best low-cost EEG devices are the Emotiv EPOC headphones. However, a scientific study of their performance is rarely available. As far as is known, no scientific comparison was made between this headset and a medical system. In the Neurophone project [15], an Emotiv EPOC P300 system is used on a PDA without showing comparative results. In [16], based on mental activities (relaxation and imagination of two types of images), it was reported that the ActiCap medical system was decidedly better than the Emotiv EPOC cuff. However, the authors did not compare the performance of both systems under the same experimental conditions. For example, the number of electrodes and their position were significantly different. Consequently, the conclusion of this study is possibly spurious. In a qualitative study [17], the authors suggested that the data provided by both systems are the same in general, but the signal is cleaner and stronger in the medical system (BCI device of G-TEC). Furthermore, none of those studies analyzed the impact of gait-related movement artifacts.

In this study, it is described an approach to detecting emotions from electroencephalogram signals measured with an Emotiv EPOC headset.

### 1.1.1 Real test cases

Man Drives F1 Car With the Power of His Mind: In 2017 in Brazil, the CEO of a Brazilian non-profit has become the first person to drive a Formula 1 racing car using only the power of one's mind. Rodrigo Hübner Mendes, Founder and the CEO of the Rodrigo Mendes Institute, used the brain interface technology, which was developed by fellow Young Global Leader Tan Le, Founder of EMOTIV Inc, to pilot the vehicle by thought alone. "The car, it doesn't have pedals, it doesn't have a steering wheel, it doesn't have anything – it's just him and his mind, driving it forward. It blew my mind," Le explained. [22]

Human Mind Control of Rat Cyborg's Continuous Locomotion with Wireless Brain-to-Brain Interface. It is developed a BBI from the human brain to a rat implanted with microelectrodes (i.e., rat cyborg), which integrated electroencephalogram-based motor imagery and brain stimulation to realize human mind control of the rat's continuous locomotion. Control instructions were transferred from continuous motor imagery decoding results with the proposed control models and were wirelessly sent to the rat cyborg through brain micro-electrical stimulation. [23]

Emotiv, the company that has commercialised mind-controlled interfaces with its Epoc headset, has developed a road safety system that automatically slows acceleration when a driver exhibits signs of distraction. Complex is the information gathered to recognise neural patterns of distraction as cognitive processing slides, and those that show if a person is “task switching” i.e. going from focusing on the road and driving your car, to sending a text. The goal was to recognise the distraction and activate a safety mechanism to awake the driver or stop the car. [24]

## 1.2 Background

The ignition of neurons in the brain triggers voltage variations. The electrical activity measured by the electrodes in an EEG headset corresponds to the field potentials deriving from the combined activity of many neuronal cells in the cerebral cortex. However, the measured cortical activity is altered by the tissue and skull between the electrodes and neurons. This introduces noise and reduces the intensity of the recorded signals. Despite this, EEG measurements also provide important information on the electrical activity of the cerebral cortex. The frequency of EEG measurements varies from 1 to 80Hz, with amplitudes of 10 to 100 microvolts. The signal frequencies have been divided into different bands, since certain frequency waves are normally more prominent in particular moods. The two most important frequency waves are Alpha (8-12Hz) and Beta (12-30Hz). Alpha waves are predominantly present during mental states of awake relaxation and are more visible on the parietal and occipital lobes. Intense alpha wave activity has also been related to the inactive brain. The activity of Beta waves, on the other hand, concerns an active state of the mind, more prominent in the frontal cortex during intense mental activities of concentration [20].

Alpha and Beta wave activities can be used in different ways to detect emotional moods (activation and valence) in humans.

Choppin [10] proposes to use EEG signals for the classification of six emotions using neural networks. Choppin’s approach is based on valence and emotional activation characterizing valence, activation and domain by EEG signals. Choppin characterizes positive emotions from a high frontal coherence in Alfa and from a high right parietal power Beta. A higher activation (excitation) is characterized by a higher beta power, a consistency in the parietal lobe and a lower alpha activity, while the domain (strength) of an emotion is characterized by an increase in the beta / alpha ratio in the activity in the frontal lobe, plus an increase in Beta activity in the parietal lobe

Oude [21] describes an approach for the recognition of emotions from EEG signals measured with the BraInquiry EEG PET device. Oude uses a limited number of electrodes and trains a linear classifier based on Fisher's discriminant analysis. He considers audio, visual and audiovisual stimuli and trains the classifier for positive / negative, activated / calm and audio / visual / audiovisual.

Takahashi [8] uses a band of three dry electrodes to classify five emotions (joy, anger, sadness, fear and relaxation) based on multiple biopotential signals (EEG, pulse and skin conductance). Takahashi trains the classifiers using machines with support vectors and reports the consequent classification of accuracy both using the entire series of biopotential signals, and relying exclusively on EEG signals.

Lin [19] applies machine learning techniques to classify EEG signals according to the subject's emotional states referred to by him while listening to music. They propose a structure for the systematic search for specific emotional characteristics of the EEG and examine the accuracy of the classifiers. In particular, they apply machines to support vectors to classify four emotional states: joy, anger, sadness and pleasure.

# Chapter 2

# Starting point

This work, thanks to the opportunity given by Luxoft, focuses on the possibility to work with a car that can be controlled by UDP packets, the specification of a protocol to govern the car, and a commercial BCI.

In the following chapters, are going to be described the characteristics of the Brain-Computer Interface (short form: BCI) that is an EPOC+ produced by Emotiv, the Cortex API provided by Emotiv to control the BCI and further Emotiv software that is used to analyze and understand the functioning of the BCI. The description of these programs is indispensable as they build the basis of this work.

## 2.1 Teleoperated car

The car adopted is an electric vehicle that has been customized to be controlled by using internet communication. It means that by sending messages through UDP packets, the angle of the steering wheel and the pressure on two pedals can be set: the gas pedal and the brake pedal. At this point, it's important to add that these messages follow a specific protocol. With the first pedal, the gas pedal, the acceleration of the vehicle can be controlled. It means with zero pressure on the pedal no acceleration is given to the car. By putting the maximum pressure on the pedal, the maximum acceleration is carried out by the car. The second pedal, however, regulates and sets the braking process. In the same way as the gas pedal, the brake pedal is controlled by the intensity of pressure that is given to it.

It's not possible to specify more detailed the characteristics of the car and the protocol. It concerns the Intellectual Property of the Luxoft company.

## 2.2 Brain Computer Interface

The BCI device enables the connection between the human brain and a computer. This device captures and analyses biological EEG signals to use them for controlling external devices and measuring performance metrics, but also to make rehabilitations much easier etc. At first, the brain activity is measured using electrodes. Then, the measured signal needs to get amplified (Figure 2.1). Although the BCI device is not very known yet, it does have a future of helping people who deal with poor health conditions. There are some other devices based on a similar system, however, none of them are as simple to use as the BCI. The other devices usually require physical movement, whereas the BCI devices works with neural activity only, making it much easier for the users [27][28].

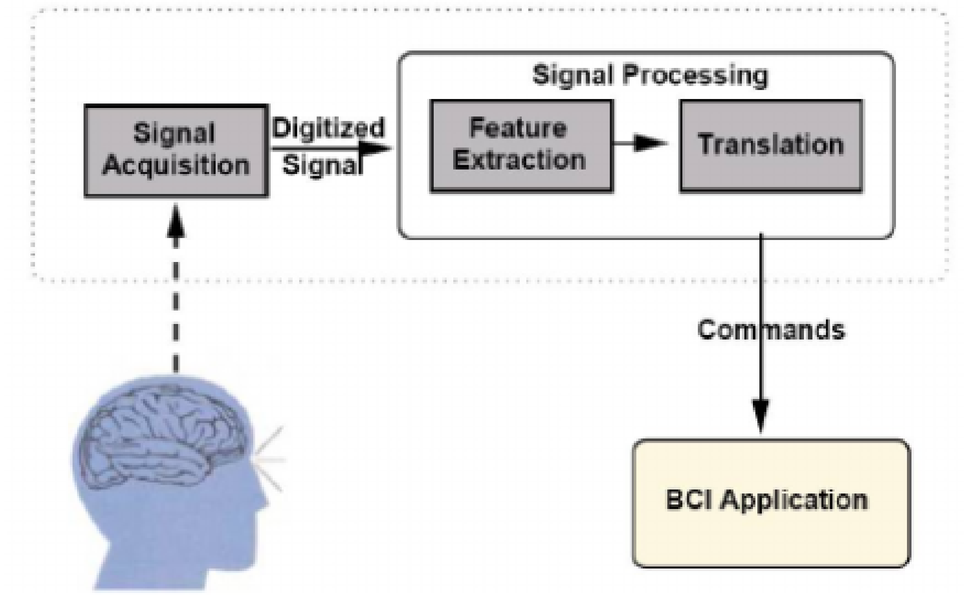


Figure 2.1. The main principle of how the BCI device works.

## 2.3 EmotivEpoc

The BCI used is an Emotiv EPOC+ 2.4. The Emotiv Software Development Kit for research mainly includes 14 channels (plus CMS / DRL references, in positions P3 / P4) each based on saline sensors.

The Emotiv EPOC headset uses 14 different electrodes plus two references and the available channels (also based on international positions 10-20) are illustrated in figure 2.2. The multiple independent sensors that consist of felt pads with gold connections to increase the sensitivity of the pickups. These felt sensors need to be moist at all time to reduce their impedance and to conduct the potential difference across the skull, this is done by using a saline solution until the level required by the software was reached. The headset is completely wireless and has a great autonomy of 12 hours, as announced by the company. The sampling rate can reach 128 Hz. In this experiment, all the standard available electrodes of the Emotiv EPOC headset were used.

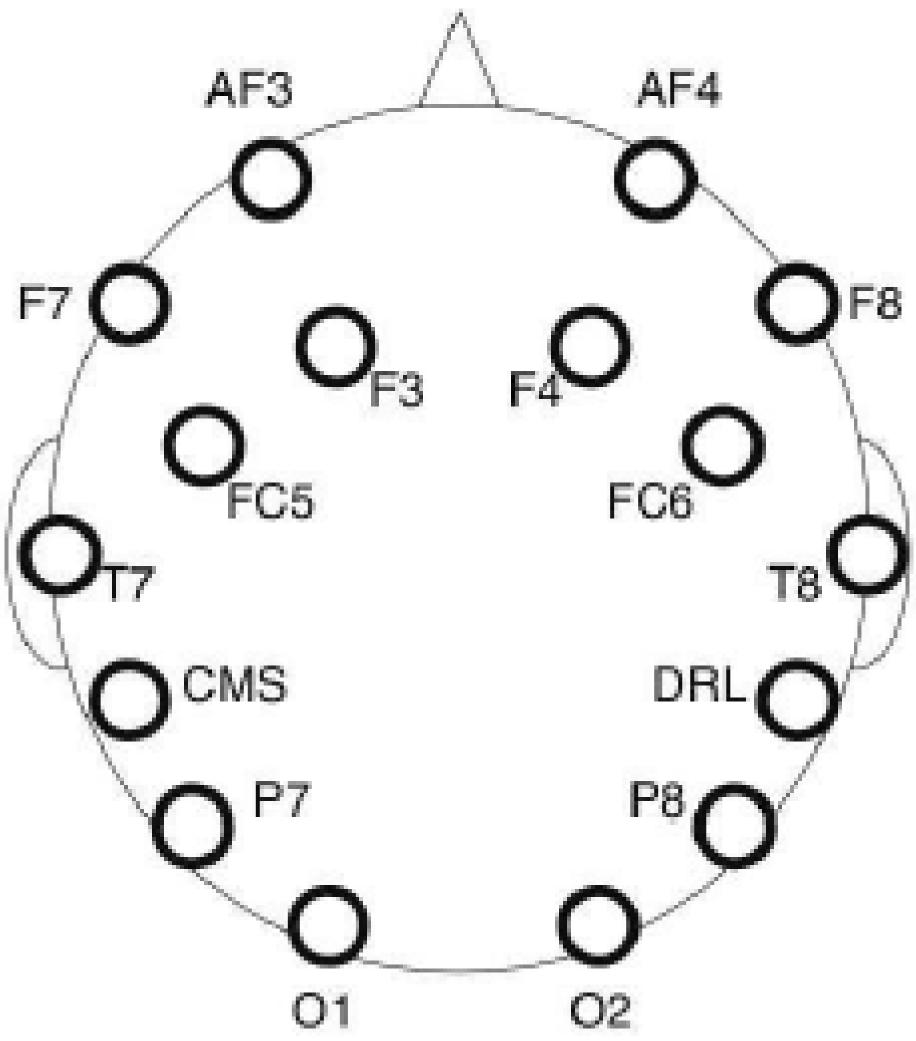


Figure 2.2. Emotiv EPOC headset: 14 different electrodes plus two references.

The Emotiv EPOC headset is an affordable easy to use marketed EEG recording device. Targeted at the emerging BCI video games market, Emotiv aims to enhance the gaming experience. Emotiv's design also has attracted the interest of neuroscientists due to the setups low price, running inexpensive experiments from one's own computer. The Emotiv headset is the key to the entire project, being what obtains and transmits the neuro-signals. The placement of the headset on ones scalp is also an integral part to the acquisition of signals. As the headset is carefully slipped on it is key that to place the sensors with the black rubber insert on the bone just behind the ear lobe, as shown below in figure 2.3.

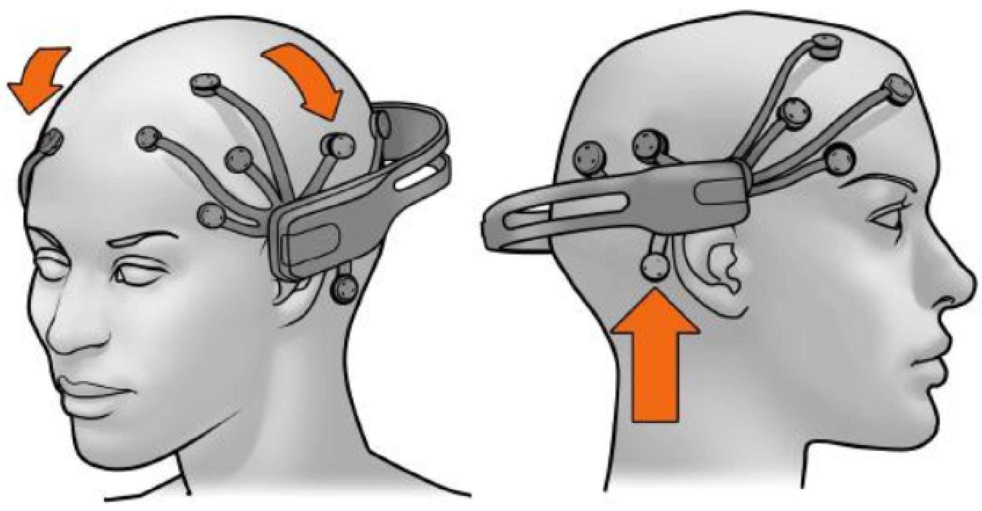


Figure 2.3. Correct placement of the headset.

It should be noted that the two front sensors should be approximately at the hairline or three fingers above the eyebrows. After the headset is in position check to see that there is good contact by the reference nodes, this is essential because a bad connection with these sensors will not produce any readings. Once the headset is connected via the wireless USB receiver the headset setup panel is displayed. The main function of this panel is to display the contact quality feedback for the neuroheadset's EEG sensors.The EPOC headset has sixteen electrodes to measure the potential difference across the skull. However there is no official reference for the user wearing the headset so the electrodes are actually paired up, and the difference between a pair is used as the measured signal. So when the user is training a certain action to manipulate a 3D cube with the Cognitiv suite, it is comparing how the values of a pair of electrodes change. Therefore whenever it sees a similar change, the software recognizes that you are trying to perform a specific action on the cube [25] [26].

In summary, the headset Emotiv EPOC+ 2.4 has 14 EEG channels, 2 reference channels, 256 EEG sampling per second, 16 bits EEG resolution, an accelerometer and a magnetometer to detect motions. The EEG sensors have to be soaked with saline solution instead with gel to make the usage of the BCI more comfortable. It has a battery and communicates the status of the brain via Bluetooth.

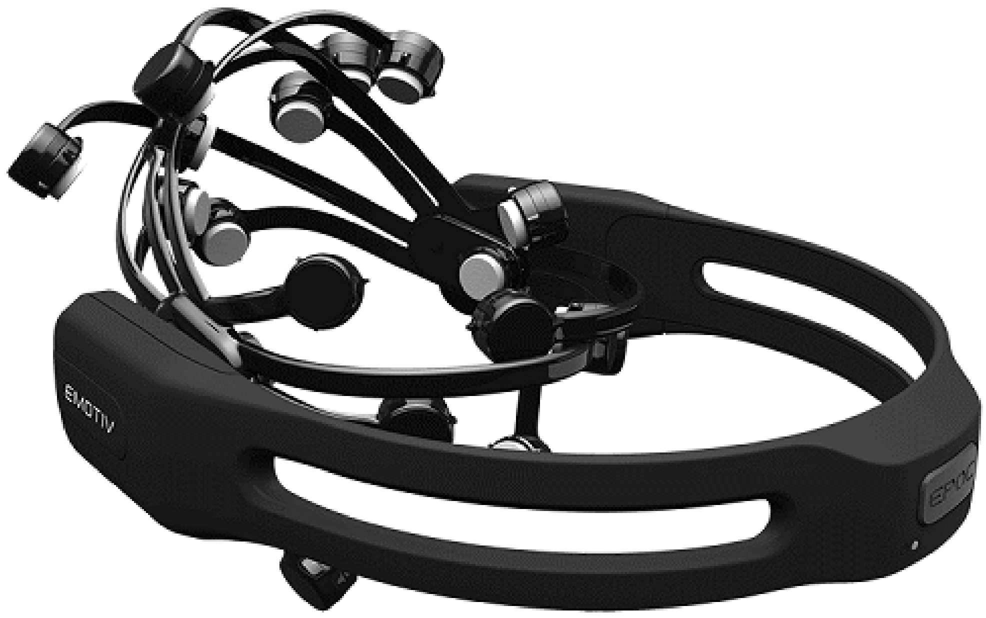


Figure 2.4. Headset Emotiv Epoc+

The headset offers a good tradeoff between price, ease of use and performances. The setup time of this headset is quantified, by the company that sells it, with 3 to 5 minutes. That is remarkable because other BCI models of the same brand have a setup time that can be 30 minutes or even more. It is defined later what is meant by setup. The ease of usage, of course, reduces the performance of the device but after all, this solution is considered the best for the purposes of this work.

To receive information from the brain by Emotiv EPOC+ it is necessary to use one of following the software that are offered by Emotiv:

- **MyEmotiv** provides a real time detection of cognitive states. These can be for example stress, focus, interest, excitement and many others.
- **BrainViz** provides a 3D model of the brain by showing the projections of frequency band information (alpha, beta, theta, gamma).
- **EmotivBCI** offers the possibility to train mental commands and facial expressions and then to use them. In particular, once you trained the “push” mental command you can try it pushing a cube. The cube will be pushed every time the BCI recognize the “push” command. You can also get information about the Performance Metrics and the Motion sensors.
- **EmotivPRO** offers apart from slight improvements almost the same features as EmotivBCI. The license of the software is not free though.
- **CortexUI** is a tool for developers that provides API to train the BCI and gets information about the state of the brain, the mental commands and the facial expressions.

EmotivBCI is used to get familiar with the tools and the potentialities of the BCI. It already has implemented all the functions and features wanted and besides that, it is free, unlike EmotivPRO. Then, CortexUI is used to develop our application. So, now the focus is on the description of these two software.

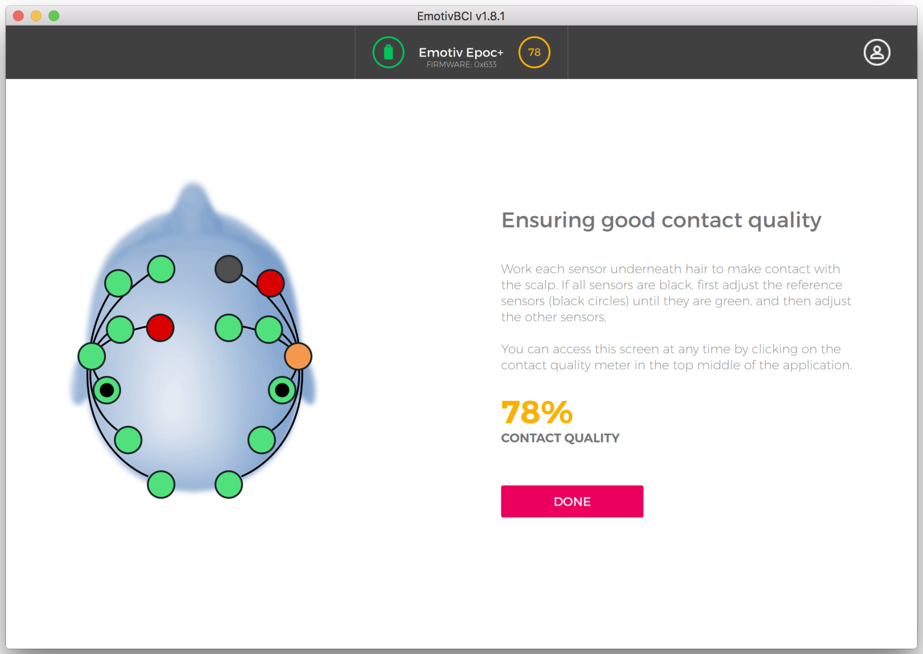
## 2.4 EmotivBCI

This suite detects and evaluates a user’s real time brainwave activity to discern the user’s conscious intent to perform distinct physical actions on a real or virtual object. The detection so designed to work with up to 13 different actions including directional movements and rotational movements and an extra action that exists only in the users imagination which is to make something disappear. The suite allows the user to choose up to four actions that can be recognized at any given time. The detection reports a single action or neutral, which would be no activity at a time, along with an action power which represents the detections certainty that the user has entered the cognitive state associated with that action. The tricky part being that increasing the number of concurrent actions increases the difficulty in maintaining conscious control over the Cognitiv detection results. This is where training would come into play. New users gain control over a single action quite quickly but learning to control multiple actions requires practice and adding more actions quickly increases the difficulty. The Cognitiv suite control panel uses a virtual n-D cube to display an animated representation of the detection output; this cube is used to assist the user in visualizing the intendedaction during the training process. In order to enable the detection, each chosen action, plus the neutral action, must be trained. The suite enables the EmoEngine to analyze the uses brainwaves and develop a personalized signature which corresponds to each particular action as well as the background state of neutral. As the engine refines the signatures for each of the actions, detections become more precise and easier to perform.

The first thing to do, when the program opens, is to log in if an account already has been created or to register if not. The second step is to set up the headset, as shown in Figure 2.5. There is an interactive image of a cranium in which are placed some dots that indicate the sensors. In the picture there are sixteen normal dots that represent the normal sensors and two dots with a black point inside that represent the reference sensor. These dots can have different colors: grey, red, orange or green.

If the dot, that represents each sensor, is grey that means that the quality of the contact is very poor or absent. Probably the headset is not in the correct position or the sensor is not touching the cutis.

If the dot is red there is a poor quality, the orange colour indicates better quality and the green light very good contact quality.



Ensuring good contact quality

Work each sensor underneath hair to make contact with the scalp. If all sensors are black, first adjust the reference sensors (black circles) until they are green, and then adjust the other sensors.

You can access this screen at any time by clicking on the contact quality meter in the top middle of the application.

78%
CONTACT QUALITY

Figure 2.5. Graphic representation of the cranium and the sensors. The color of the sensors depends on the contact quality.

To set up correctly the BCI, the first thing to do is place it in the correct position. To do that, it is needed to check the reference sensors: when they are green the position is correct, and it is enough to adjust the other sensors. The hardest part of the operation is to set the correct position. Afterwards, it is very easy and quick to obtain good contact quality. When the setup is completed the main window of EmotivBCI appears and it is divided in four sections:

- **Motion sensors**: is a section where the output of many motion sensor is shown: four Quaternions, three Accelerometer and three Magnetometer (one for each axis).
- **Performance metrics**: in this section, the emotional states of the brain are described also thanks to graphics (Stress, Focus, Interest, Relaxation, etc.).
- **Facial expressions**: in this section is to possible to train some expressions like smile, smirk left/right, frown etc. By choosing the live mode a face model reproduces the facial expressions done in reality.
- **Mental commands**: the functions of this section are similar to

those of the section facial expressions but instead of training expressions is possible to train mental commands as push, pull, left/right, rotate left/right etc. Even in this section there is a live mode that moves a cube according to the commands given.

For the Mental command and Facial expression sections there are two windows (Figure 2.6): one is for training, the other is for live mode.

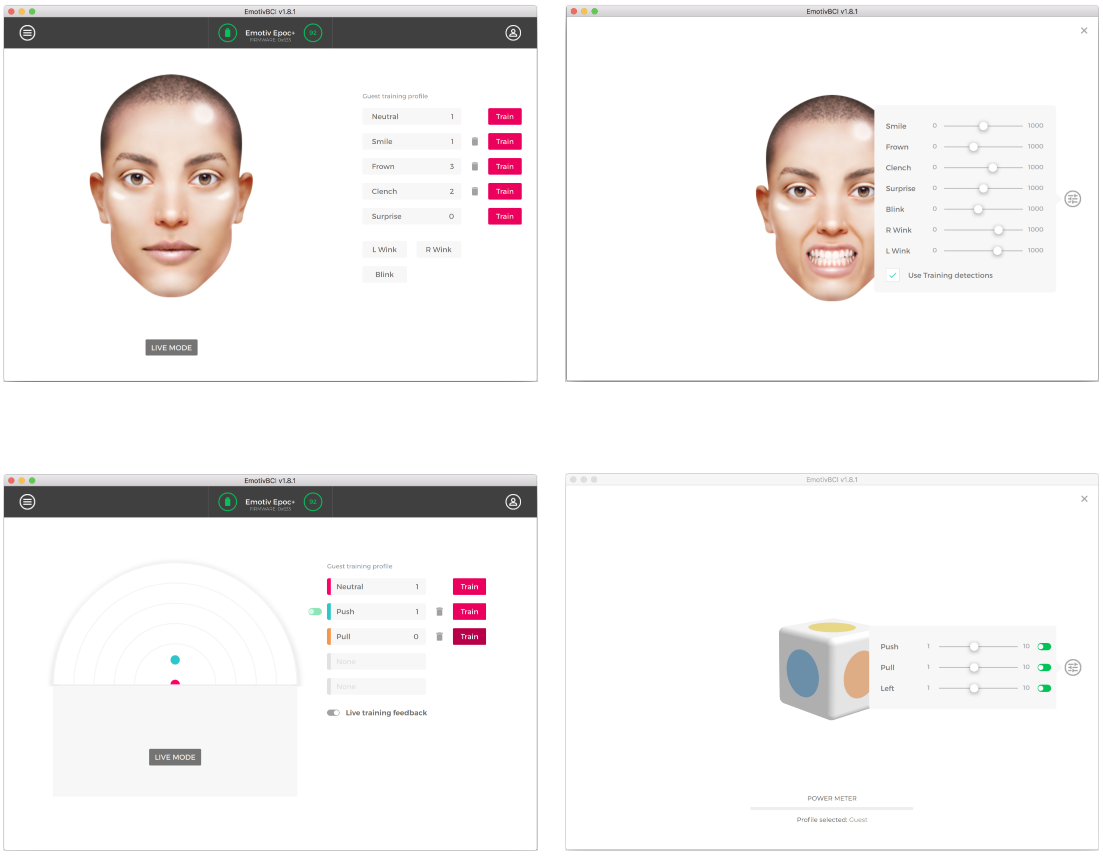


Figure 2.6. the top left image shows the training window for the facial expression, the image below is the training window for the mental command, the top left image is the live window of the facial (the sliders change the sensibility for the recognition of an expression), finally the image on the bottom right corner shows the live window for the mental command. As in the facial expression the sliders increase or decrease the sensibility for a certain command.

Therefore, a user should set up correctly the BCI and start training an expression in the training window. As for the eye movement, the expressions do not have to be trained. After the training, a user can switch to the live mode window, where the model of a face is displayed. It reproduces the same expression of the subject that wear the headset. All the expressions are showed by the face model, not only the trained ones. It is also possible to set the sensitivity with which it recognizes an expression.

The mental command section is very similar to the facial expression one. There are the same two windows, with the first you can choose a mental command and train it. The initial step in creating Mental Commands is to train the system to recognize your background mental state, the so-called NEUTRAL condition, by recording a brief period of your brain patterns while you are not trying to execute any commands. Training a new mental command is as simple as selecting the desired command label in the training mode, then imagining the consequences of the command for 8 seconds (for example, imagine the target object floating up into the air for the LIFT command) while the system records the mental patterns you want to associate the command. During the training, is it possible to see a cube that executes the mental command trained.

For example, if the training concerns the push command, the cube moves away. If the left command is trained, instead, it will move to the left. In this window, it is shown a semicircle (Figure 2.7) where the centre is the neutral state. After the training of a new command, a coloured dot appears in the semicircle. The goal is to train the commands in a way that the dots are as far as possible from each other, that means that the difference between the states is high enough so that in live mode the BCI does not confuse one command with another.

After the training, it is possible to go into live mode where the cube moves according to own mental command: if the right command is trained and sent, the cube moves right. As for facial expressions even here, it is possible to set the sensibility for each command. If the sensibility for a command is high, it is more probable that the BCI will recognize its state.

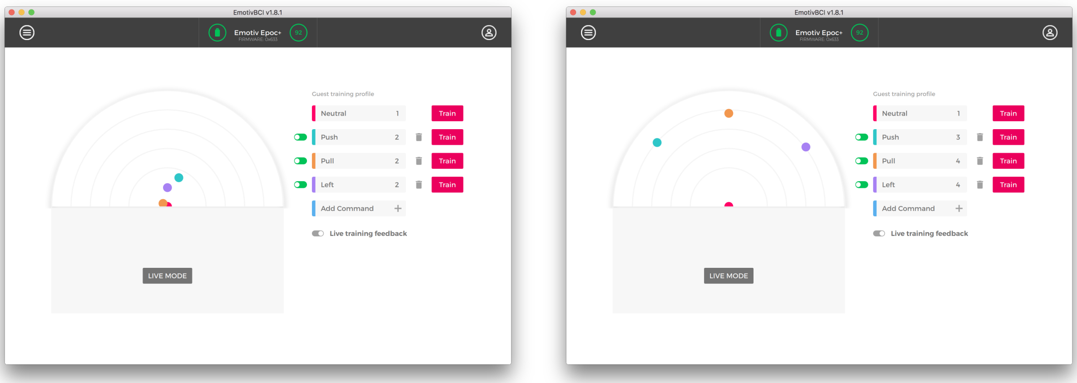


Figure 2.7. the first image shows a bad training indeed the dots the represent the commands are close. The second image shows a good training with the dots distant from each other and from the center of the semicircle (were the neutral is always placed).

In the Table 2.1, all the mental commands and the facial expressions that can be trained are shown:

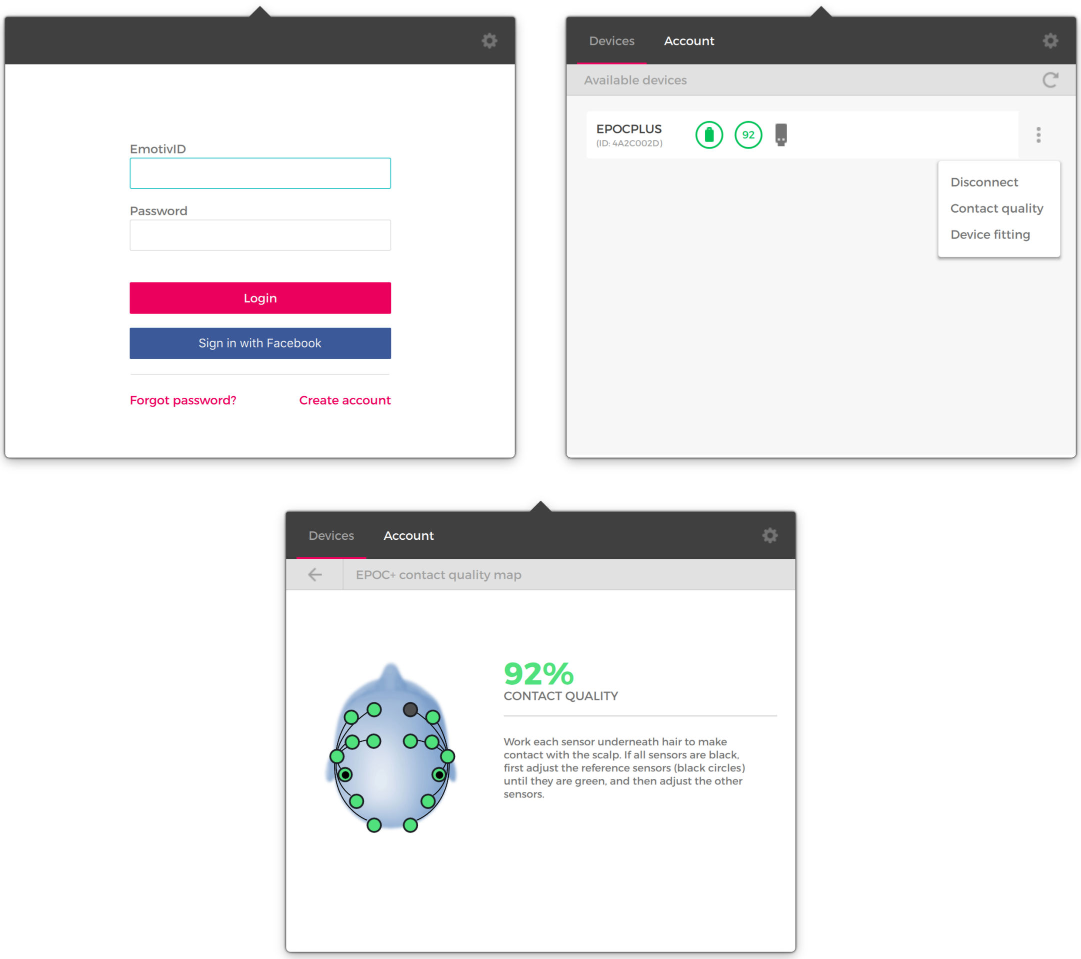
Table 2.1. Commands

| **Mental commands** | **Facial expression** |
|---|---|
| Neutral | Neutral |
| Push | Blink |
| Pull | Wink Left |
| Lift | Wink Right |
| Drop | Hori Eye |
| Left | Surprise |
| Right | Frown |
| Rotate Left | Smile |
| Rotate Right | Clench |
| Rotate Clockwise | Laugh |
| Rotate Counterclockwise | Smirk Left |
| Rotate Forward | Smirk Right |
| Rotate Reverse | |
| Disappear | |

## 2.5 CortexUI

Cortex UI is a software released by Emotiv for developers who want to use the potentiality of their BCI in an application. This software is very minimal. It only has four windows. One is for login, one is to connect the BCI through Bluetooth, one is to check the contact quality, and the last one is to set the account settings.

As in EmotivBCI after the connection of the headset, the model of a skull with the sensors to setup the BCI is shown. The convention is the same as before: if the sensor is green the contact quality is good. If it is red, the quality of the connection is bad. (Figure 2.8).



Work each sensor underneath hair to make contact with the scalp. If all sensors are black, first adjust the reference sensors (black circles) until they are green, and then adjust the other sensors.

Figure 2.8. Three Cortex UI windows. The first on the top-left is to login, the second to connect/disconnect/setup the headset and the third shows the contact quality (as in Emotiv BCI).

After the connection of the headset to Cortex UI and its setup, it is possible to use internet API to get information from BCI. To use these API first, you have to log in to the Emotiv server, with the login method, using the same Emotiv credential used to log in CortexUI. Later, it is explained what is a method and in particular how the login method works.

Without a PRO license (subject to a fee), the access to API is limited. Otherwise, it is possible to use more functions and features. For example, with the PRO license, the raw EEG data are included but without it, only a set of commands/expressions can be trained as in EmotivBCI and then receive the current command/expression the headset gives.

Below, it is illustrated how the communication through these API (Figure 2.9), between a developed program, Cortex UI and Emotiv EPOC+ works. Note that the image refers to Cortex API 1.0 that was the most recent version during the developing of the program. Since the end of 2019, there is a new update available (Cortex API 2.0).

An application with this API communicates actually with an Emotiv server and not directly with the headset. The Emotiv server then forwards the command to the CortexUI that communicates with the Headset. As imagined, this architecture is not very efficient and Internet latency is very important.

Unfortunately, due to the lack of good documentation, it is unknown how effectively these APIs and architecture work, but a few examples are presented to show how most likely the system works. First, it is explained how the training works and then how to read and extract the mental commands.

Training:

- Application asks Emotiv server to train mental command “push”.
- Emotiv server sends to application a message to notify that the training is starting, in the meantime the server asks CortexUI for the EEG.
- CortexUI asks EEG to headset and forwards it to EmotivServer.
- Emotiv server records EGG for a certain amount of time and trains a neural net.
- Emotiv server notifies application that the train is complete.

Reading mental command:

- Application asks Emotiv server to read the mental commands.
- Emotiv server asks CortexUI to send itself the EEG.

- CortexUI gets the EEG from headset and forwards it to EmotivServer.
- Emotiv server reads EGG and sends a previous trained command that has the most similar EEG pattern of the last EEG sampling.

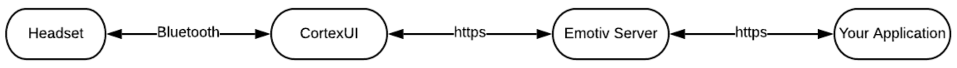


Figure 2.9. This scheme shows the communication between our application, the BCI, CortexUI and the Emotiv server.

## 2.6 Cortex API

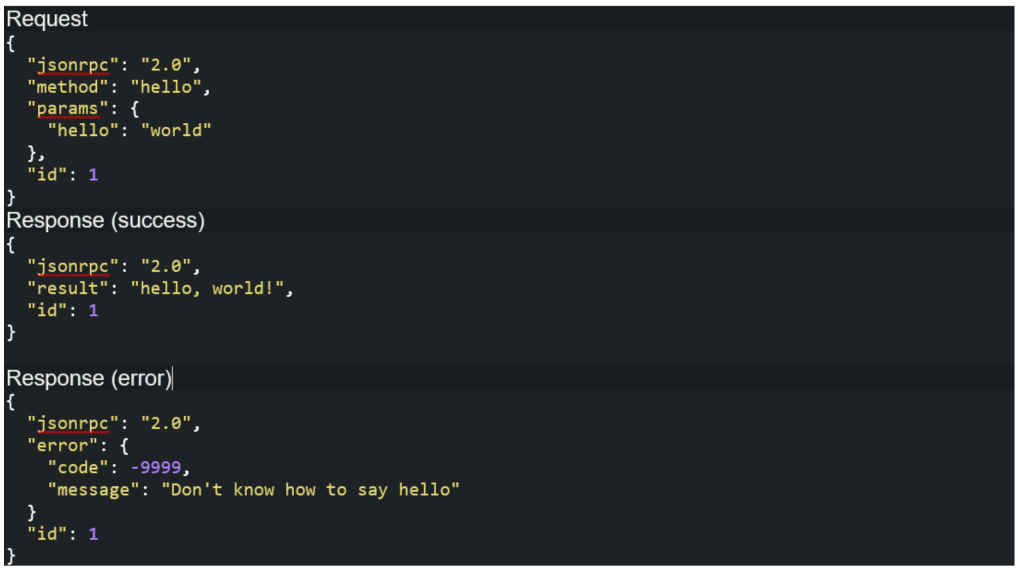
The Cortex protocol is made of 3 building blocks: WebSockets, JSON, and JSON-RPC. WebSockets provide a real-time connection to the underlying Cortex service, designed to be easy to use in both desktop and web-based applications. JSON is a widely supported format used by Cortex to send and receive data, and JSON-RPC is a standard way of using JSON to make requests and get back the results.

The Cortex service listens on port 54321. Depending on the WebSocket client used, this means is possible to connect by using the URL wss://emotivcortex.com:54321. JSON-RPC builds on top of JSON by adding a few standard keys to track requests. An example, of request and response are shown in Figure 2.10. The ”jsonrpc” key is always 2.0, indicating the protocol version. ”method” indicates which of the API Methods is invoked and ”params” contains the wanted keys and values to use for that method. To find the parameters for a given method, look it up in the Method reference.

It is given a response with a ”result” or an ”error” if the request was unsuccessful. “id” is a parameter used to track the association between requests and responses. If it is sent 1, the response is 1. If it is sent ”hello”, the response is ”hello”.

Currently, the request in Cortex is synchronous. They are planning to support also the asynchronous request in the next version. The exception to this is streams of data that received from the headset, which do not follow a request-response pattern and do not follow JSON-RPC. They are described below, under Subscriptions. [29]

When it comes to explaining and understanding the communication between CortexUI, Emotiv Server and Cortex API an important aspect to consider is the login to own account. The first things to do, when the API are included, is to log in with the same credential used in Cortex UI. In this way the server recognizes the account to which he forwards the messages.



```
Request
{
  "jsonrpc": "2.0",
  "method": "hello",
  "params": {
    "hello": "world"
  },
  "id": 1
}
Response (success)
{
  "jsonrpc": "2.0",
  "result": "hello, world!",
  "id": 1
}

Response (error)
{
  "jsonrpc": "2.0",
  "error": {
    "code": -9999,
    "message": "Don't know how to say hello"
  }
  "id": 1
}
```

Figure 2.10. Example of request and responses wit JSON-RPC

Before presenting the most important API methods used for this project, it is relevant to point out that the documentation of the application of the API is very scarce. Therefore, empiric work is done to understand how this API work. For that reason, the following description is a mix between official documentation [29] and the assumptions done.

### 2.6.1 login

This method is used to provide the Emotiv credentials. As mentioned earlier, the same credentials in CortexUI have to be used. Furthermore, this method returns also the client id and the secret. To use Cortex API in an application, it is requested to register the app on the Emotiv website and create the pair of client id and secret.

### 2.6.2 logout

By providing the username the user can be logged out. Afterwards a new login is possible.

### 2.6.3 authorize

The function of this method is almost the same as the login method and serves to authenticate and get an authentication token needed for every further method that is called. It caused a lot of problems. In the documentation is written: "Authenticates a user. You can authenticate as an anonymous user, but to get access to Raw EEG data or high-resolution performance metrics or both, you need to provide a username and password of user from the Emotiv Cloud, and the client id/secret for your application.".

Since the Raw EEG data or high-resolution performance metrics are not needed (it is not deemed to buy the license to access these information), at the beginning the authentication is considered as an anonymous user. Unfortunately, malfunctions in the API are found, for example, the creation of a new profile did not work. After checking on the internet, to solve the problems it is requested to authenticate with the credential even if only the free API is needed.

### 2.6.4 setupProfile

This method takes as parameters a string that is the profile name and another string that is the "status". The status can be "create" to create a new profile, "save" to save the profile and all the training on Emotiv Server, "load" to load a profile and all the training previously saved, "unload" to remove all the saved training of a profile, "rename" to rename a profile and "delete" to delete it. So potentially a person can train some mental commands one day and the day after loading the profile and get all the trainings already done. However, this is not suggested because the shapes of the brain can change from day to day or even from hour to hour. This method is not mandatory and the creation of a profile is not requested, but it is used because it is needed to change the sensibility of facial expressions or mental commands.

### 2.6.5 queryHeadsets

This call returns an array of the connected headsets to CortexUI. In this case, the array has always one element due to the possession of a single headset then a connection of just one BCI. This method is necessary because the headset is needed to get a session and a session-id (it is explain later what a session is).

### 2.6.6 createSession

A session is an object that creates the link between your application and an EMOTIV headset. When the user wants to work with a headset, the application should create a session first. Then you can:

- subscribe to the data stream of the headset
- create a record and add markers
- use BCI

The application can open only one session at a time with a given headset. But it can open multiple sessions with multiple headsets. A session is created by “createSession”, closed by “updateSession” and linked to an application. All the sessions of an application are automatically closed when the application is disconnected from the Cortex service. A session is also closed if the headset is disconnected. A session is a temporary in-memory object; it is not persistent. After a session is closed, it is destroyed by Cortex.

### 2.6.7 subscribe

After opening a session with a headset, it is possible to subscribe to one or more data streams. Each data stream gives real-time access to data from the headset (EEG, motion, etc.) or data calculated by Cortex (band powers, mental command, etc.). After successfully subscribed to a data stream, Cortex keeps sending data sample objects. A subscription is linked to a session. All the subscriptions of a session are automatically cancelled when the session is closed. Call unsubscribe to cancel a subscription. The “subscribe” method uses as parameter a string called “stream” that defines the type of subscription demanded. For example, for the training, the stream field has to be set to “sys” to subscribe to it. All the stream types are shown in the table 2.2.

Table 2.2. Streams

| Stream | Description |
|---|---|
| mot | Motion data from the accelerometer/gyroscope |
| eeg | Raw EEG data |
| com | Mental Command Event |
| fac | Facial Expression Event |
| met | Performance Metrics data |
| dev | Device data include battery level, signal strength, and signal quality all of channel headset |
| pow | Band Power data |
| sys | System event (for set up training) |

### 2.6.8 unsubscribe

This method is used to cancel a subscription that was previously created by the subscribe method.

### 2.6.9 training

This method is used to control the training of the mental commands and facial expressions detections. Before start training, the subscription is set to the ”sys” data stream. This call has three main parameters:

- “detection” is a string that can be “mentalCommand” if train mental commands or “facialExpression” if train facial expressions.
- “action” is a string in which defines the wanted action to train. It can be a mental command as “push” if the detection is “mentalCommand” or a facial expression as “smirkLeft” if the detection is “facialExpression”. All the actions for both mental command and facial expression can be found in table 2.1.
- “status” defines what to do with the train. The possible states are:
  - “start” to start to train a certain action.
  - “accept” to accept a training after completion and if the result is satisfactory.
  - “reject” to reject a training after it has already started and then get a result that is not satisfactory.

- "reset" to cancel the current training.
- "erase" to erase all the training data for the specified action.

A possible workflow could be the following: start the training of the mental command "pull" [detection = "mentalCommand", action = "pull", status = "start"], so for a few seconds the user has to give the mental command, then a message is received to show that the training has been successfully and afterwards accept the training [detection = "mentalCommand", action = "pull", status = "accept"].

### 2.6.10 mentalCommandGetSkillRating

This method returns the skill rating of a mental command action if it is specified as parameter, or the overall mental command skill rating if not specified any. In other words, it returns a number from zero to five that tells how good the training is. Zero is the worst training value and five is the best rate.

### 2.6.11 mentalCommandActionSensitivity

This method gets or sets the sensitivity of the four active mental command actions. It gets one main parameter that is an array of four elements that contains the sensitivity of each active action. The order of the values must follow the order of the active actions, as returned by the "mentalCommandActiveAction" method. The value can go from one to ten: with one the relative mental command becomes insensitive but with ten it gets very sensitive. This is a useful regulation to fix the false positive and the true negative results. For example, a mental command is read even if it is not given by the user or vice versa it is not read even if it is given.

### 2.6.12 facialExpressionThreshold

This method can get or set the threshold of a facial expression action for a specific profile. Actions with a low threshold are less likely to be detected. Actions with a high threshold will be detected more often. This method is similar to "mentalCommandActionSensitivity", but instead of sending an array it gets as parameters the name of the action (whose threshold has to be change) and the threshold value that goes from zero to one thousand.

## 2.7 Cortex API 2.X

The main difference between Cortex API 1.0 and 2.0 is that in the first version the communication has to be instantiated with a remote server with the second version the communication is made with a local host. With all probabilities, the second version will communicate directly with CortexUI. Since in this work the new version is not tested, it is not known if this improves the management of the headset but probably it will improve the latency of the communication.

With this new version, the login is no more needed avoiding a bit of redundancy and simplifying the communication with the server, since with the old and current API there is login and authentication. Some other methods change their names but their functionality stays the same. The 2.0 version has been released on July 2019 and in October 2019 the current version is 2.2.1. Probably besides to have improved the latency and simplified the communication these new APIs have brought more efficiency in the management of the BCI.

## 2.8 Goal of the thesis

To sum up, there is a car that can be driven remotely with UDP packets, a defined protocol and Emotiv EPOC+ that is a BCI. By using Cortex API and a software called CortexUI, it is possible to develop a program that can train the headset and then "read" mental commands or facial expressions.

The final goal is to develop an application that can manage the training of mental commands and then possibly drive the teleoperated car by following the mental commands that are read from BCI. To be clear, this application should train these mental commands and each of them corresponds to a movement of the car. In this phase, it is not clear how the commands work but for example when the user gives the mental command "left" the car steers left. The process is similar for the acceleration. When the "push" command is received, the gas pedal has to be pressed. After the training phase the BCI "reads the brain" and sends every 200 milliseconds the mental command that it thinks is the most probable.

## 2.9 First tests with EmotivBCI

At the beginning without any idea how a BCI in general works, and in particular Emotiv EPOC+, the main focus is to start to “play” with it by using EmotivBCI, the application described above. The connection between the BCI and the program resulted very easy, much more complex was the setup of the headset. In fact, the correct positioning of the reference sensors was hard to find and after that the contact quality of the sensors where not so good (the colours on the cranium map were grey or red). The instructions of the BCI did not help so much to resolve these firsts problems.

Knowing that the training and then the recognising of the facial expressions would have been easier than the training and the recognizing of the mental commands so, it has been decided to start to this test with the firsts. Without much success at the beginning, indeed just a few times the application could really recognize the expressions that were done. It has been worse during the training of the mental commands, almost helpless and lost: How a mental command can be given? What should be thought? Those are the first questions that went out.

It is really hard to give a command without actually doing something. A command to lift the arm and lift an arm can be given to ourselves, but without moving a body part it is pretty hard. It is indeed not recommended to move the body while giving a mental command, because the movement has an effect on the command and it is hard to reproduce the same physical movement and so the same mental command. Trying to give those commands it makes a funny situation.

## 2.10 First tests with CortexUI and Cortex API

CortexUI, as already explained, is a very minimal program in the sense that it just has two main functions: find and connect the headset via Bluetooth and check the connection quality. Even if there are few functions, they do not work so well. In fact, it is really hard to establish a connection to the headset and sometimes this connection is really unstable. The window which shows the contact quality is not reloading in real-time, and often it is helpful to go back to the previous windows to see the changes of the connection.

Consulting the Cortex API documentation to see which functions are available for the goal of this work, but unfortunately the documentation is not very clear. At least Emotive provides some example programs to show the main services. A lot of time is spent to try many functions to understand empirically how they really work.

## 2.11 Problems and first solutions

In this section, the problems faced with these first tests and the attempts to solve them are summerized below.

The first problem is the connection (via Bluetooth): not so much could be done about it since the problem seems to be in the software (EmotivBCI, CortexUI) or in the hardware (Emotiv EPOC+). The only aspect noted is that lower the battery of BCI more the connection problems are present. Therefore, the headset is always kept full charged.

Another issue found concerns the setup problem and the contact quality of sensors. It is partially resolved by trying out different solutions: learning the exact position of the reference sensors, and by reading the setup tip on the Emotiv web site [30] it has found out that by soaking the sensor with the saline solution the quality of the contact will improve. At the beginning, only few drops are put on them.

EmotivBCI caused many problems with the training of both, facial expressions and mental commands. By reading the instructions on the internet, primarily on the Emotiv web site, has been understood that it is needed to keep the facial expression for all the duration of the whole training. However, in the beginning, the movement of the facial expression is repeated various times. It can be clarified with an example: training a clench expression, to keep clenching the teeth and then to relax the face muscles during the whole training. By keeping clenched the teeth for all the time the training, instead, improved a lot the quality.

For what concerns mental commands the situation is more complicated. Emotiv gives many advices but a mental training is mandatory as well. The first step is to understand how the BCI recognizes a mental command. The first mental command that Emotiv suggests to train is the neutral command, that is the state of rest of the brain when the subject does not think about anything. The training is a record of the EEG for a certain timeslot (few seconds). This record is filtered and then processed with some machine learning techniques. Then the user can train other mental commands. The important point is that every command must produce an EEG that is unique and easily distinguishable from the others.

The second step is to decide what to think about. Emotiv gives the following answer:

> ”The thought that you train on and use for your Mental Commands can be anything. They can be literal (i.e. you can try and focus on pushing the virtual box) or they can be as abstract as you like (i.e. where push is associated with visualizing a scene or counting backwards from 500 in steps of 7). The possibilities are endless. Different strategies work best for different people, so try a few out. If you are training a profile with one command, you want to make that one command as strong and distinct as possible. One way to achieve this is to use something that is multi-modal - i.e. something that contains different sensory and kinematic (related to movements of your muscles) components all together. If you have a strong disposition toward any of these modalities (e.g. you are a musician and so can easily imagine auditory sounds), you may find focusing on this single modality works best for you. Given the difficulty of train and use mental commands, it has been decided that Xavier, the program implemented in this work, had to works both with facial expressions and mental commands because the firsts were much easier than the latter. explain better maybe. If you are training a profile with multiple commands, you may find you get best results if each of your commands uses a single and different sensory or kinematic modality (e.g. one that is visual, one that is auditory and one that is kinematic). What is most important is that they are distinct from each other and you are able to recreate them accurately in your mind repeatedly. You may also find that associating different hand gestures or postures with a Command can help to better reproduce them.” [31]

So, the starting point is to try to control the thoughts to train mental commands. It is suggested to train well a command before introducing a new one. It has been noticed immediately that it is essential to give good knowledge and good tips to the user to train and be able to provide these type of commands.

A developing problem found is how to test the software: it is not so easy to use the teleoperated car because it can be dangerous, and it requires to setup a safe environment which takes a lot of time and money. Then, it has been decided to create a simulated environment with a car that can be controlled in the same way as the real one, in order to test the functioning of Xavier. Of course, the simulation could not replace the real car completely, indeed some real tests with the car are still needed, but at least it is possible to save a lot of time by first using software trials and different scenario.

Another problem is, as already anticipated, the poor value of the Cortex API documentation, indeed there are just no satisfying examples. Therefore, a lot a time is spent on trying these APIs and understand what could really be done. This fact had a negative impact even on the planning and on the design of Xavier since it is not known what are the possibilities and which information is possible to get from the headset. Most probably EmotivBCI uses the same APIs and this can be helpful to understand how those really work. For example, very often an error message occurred when trying to change the sensibility of the eye movements. Only by checking EmotivBCI, it has been understood that this function is not supported as well as for other movements.

The main idea of Xavier at the beginning was to train four commands and each of them corresponds to a command which is sent to the car. The command “push” presses the accelerator, the “pull” command presses the brake of the car and “left” and “right” commands turn the vehicle respectively left or right. After some tests with EmotivBCI, it is clear how difficult it is to manage multiple mental commands, so the “pull” command has been removed and replaced by the “neutral” one which is mandatory for the training. When driving, not only a pedal is pressed or the steering wheel is turned, it can also vary the intensity of the push or the spin: it is possible to press the gas pedal slowly or hardly and turn the steering wheel of 10 degrees or 90 or 180. So this is another problem to think about because the given commands just expressed a movement (turn, accelerate or brake) and not the intensity of it. It is decided to give a fixed value for the acceleration and the braking so every time a “neutral” or “push” command is received, a certain level of pressure is sent to the corresponding pedal. Unfortunately, it is not possible to apply the same for the steering. The system cannot give a sudden 90 degrees turn command because that could create at least three main problems:

- The car and the mechanical system, that turns the steering, can break.
- Such sharp curves are not comfortable for the passenger.
- It is impossible to do a precise turn.

For these reasons, a linear function (Figure 2.11) has been introduced.

On the x-axis there is the frequency of sending packets to the car and on the y-axis there is degree values of the steering. The maximum steering value is 440 and it is reached after about 5 seconds. If received a “left” command, it starts to turn the steering of 17.6 degrees, if received the same command for twenty times, it keeps increasing the steering of the wheel of 17.6 degrees for 20 times. So, since the frequency is set to 5 Hz after 3 seconds, the car turns of $5 * 3 * 17.6$ degrees that is 264.

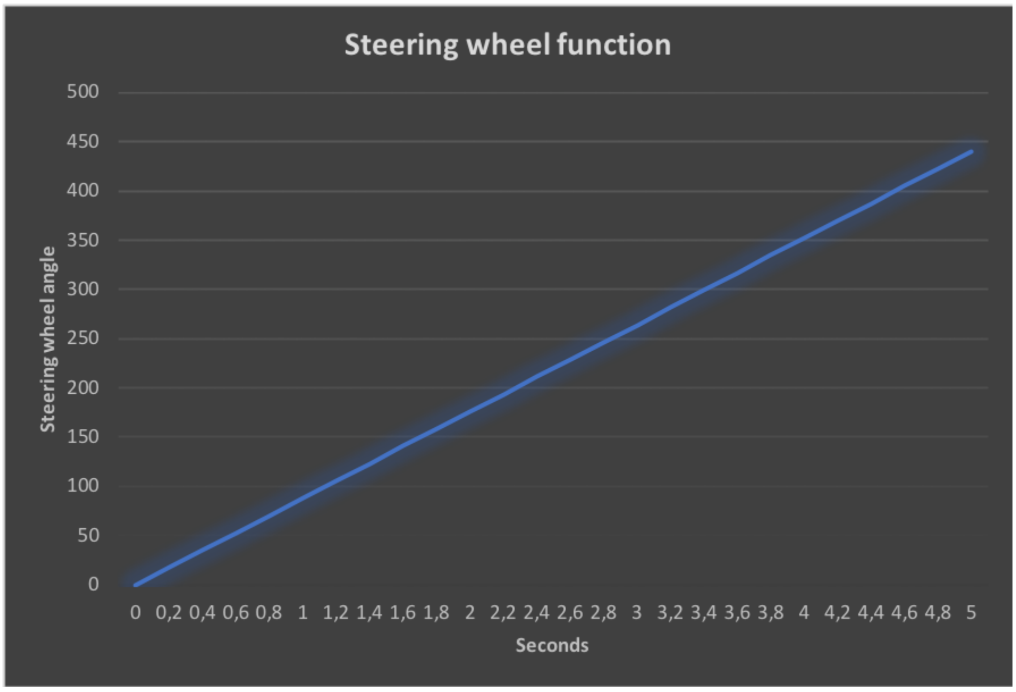


Figure 2.11. Linear function: on the x axis the seconds and on the y axis the angle of the steering wheel (max angle value is 440°)

This frequency has been chosen because is more or less the same frequencies of the streaming of the headset in other words the frequency in which the headset tells which is the facial expression or the mental command the user is doing or thinking.

# Chapter 3

# Development of Xavier and of the simulation

It has been decided to split the work in two parts:

1. The development of a software application that allows to drive the car with the BCI that is called Xavier.
2. The creation of a simulation with a simulated car that can be controlled by UDP packets to test Xavier before the final tests on the real car.

## 3.1 Simulation

### 3.1.1 ROS Robot Operating System

The first question is which is the best tool to simulate a world and a teleoperated car? It has been chosen ROS because is Open Source, thus free, is largely used, there is a big community, many worlds and robots (and cars) can be found on the internet (free license).

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

The ROS runtime ”graph” is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the

ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. ROS is not a realtime framework, though it is possible to integrate ROS with realtime code.

ROS has two levels of concepts: the Filesystem level, the Computation Graph level. The filesystem level concepts mainly cover ROS resources that you encounter on disk, such as:

- **Packages**: Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most atomic build item and release item in ROS. Meaning that the most granular thing you can build and release is a package.
- **Metapackages**: Metapackages are specialized Packages which only serve to represent a group of related other packages. Most commonly metapackages are used as a backwards compatible place holder for converted rosbuild Stacks.
- **Package Manifests**: Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages. The package.xml package manifest is defined in REP-0127.
- **Repositories**: A collection of packages which share a common VCS system. Packages which share a VCS share the same version and can be released together using the catkin release automation tool bloom. Often these repositories will map to converted rosbuild Stacks. Repositories can also contain only one package.
- **Message (msg) types**: Message descriptions define the data structures for messages sent in ROS.
- **Service (srv) types**: Service descriptions define the request and response data structures for services in ROS.

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways.

These concepts are implemented in the ros_comm repository.

- **Nodes**: Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as roscpp or rospy.
- **Master**: The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- **Parameter Server**: The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.
- **Messages**: Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).
- **Topics**: Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other's existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type.
- **Services**: The publish / subscribe model is a very flexible communication paradigm (Figure 3.1), but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done

via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.

- **Bags**: Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

The ROS Master acts as a nameservice in the ROS Computation Graph. It stores topics and services registration information for ROS nodes. Nodes communicate with the Master to report their registration information. As these nodes communicate with the Master, they can receive information about other registered nodes and make connections as appropriate. The Master will also make callbacks to these nodes when this registration information changes, which allows nodes to dynamically create connections as new nodes are run.

Nodes connect to other nodes directly; the Master only provides lookup information, much like a DNS server. Nodes that subscribe to a topic will request connections from nodes that publish that topic and will establish that connection over an agreed upon connection protocol. The most common protocol used in a ROS is called TCPROS, which uses standard TCP/IP sockets. This architecture allows for decoupled operation, where the names are the primary means by which larger and more complex systems can be built. Names have a very important role in ROS: nodes, topics, services, and parameters all have names. Every ROS client library supports command-line remapping of names, which means a compiled program can be reconfigured at runtime to operate in a different Computation Graph topology.

For example, to control a Hokuyo laser range-finder, it has started the hokuyo_node driver, which talks to the laser and publishes sensor_msgs/LaserScan messages on the scan topic. To process that data, it should be written a node using laser_filters that subscribes to messages on the scan topic. After subscription, the filter would automatically start receiving messages from the laser.

Note how the two sides are decoupled. All the hokuyo_node node does is publish scans, without knowledge of whether anyone is subscribed. All the filter does is subscribe to scans, without knowledge of whether anyone is publishing them. The two nodes can be started, killed, and restarted, in any order, without inducing any error conditions.

Later it could be possible to add another laser to our robot, so it's needed to reconfigure the system. What is needed is remap the names that are used. When the first hokuyo_node starts, it is told instead to remap scan to base_scan, and do the same with the filter node. Now, both of these nodes will communicate using the base_scan topic instead and not hear messages on the scan topic. Then just start another hokuyo_node for the new laser range finder. [36]

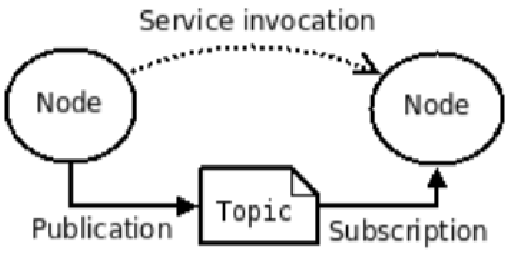


Figure 3.1. Publish/Subscribe paradigm.

### 3.1.2 Gazebo

Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. While similar to game engines, Gazebo offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs. [37]

Typical uses of Gazebo include:

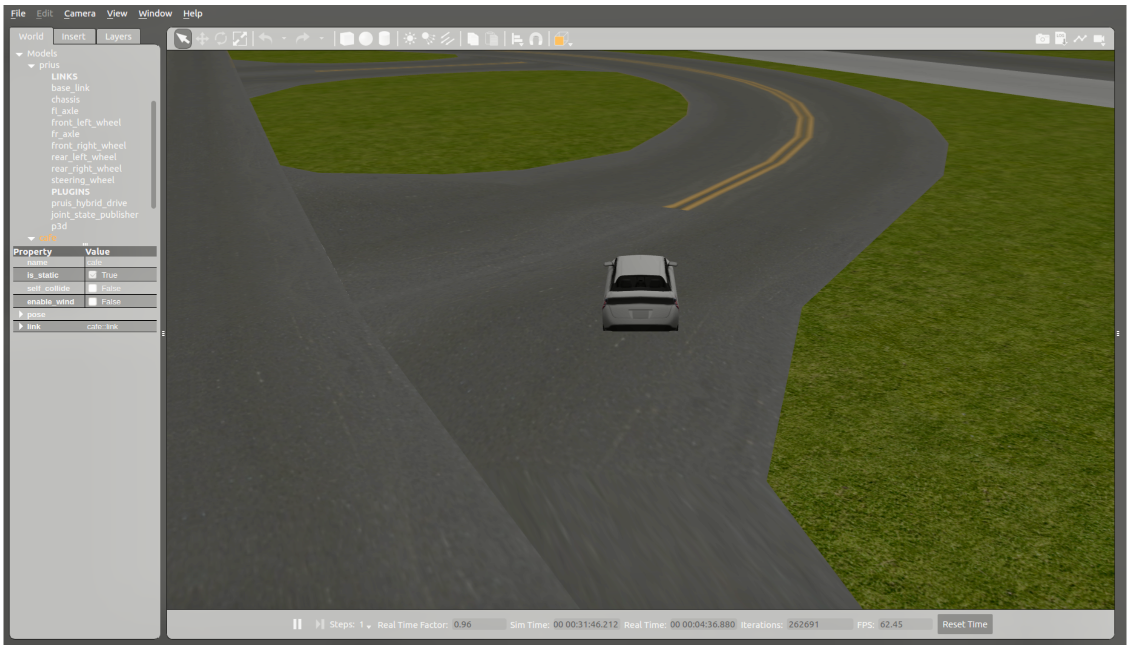
- testing robotics algorithms.
- designing robots.
- performing regression testing with realistic scenarios.

A few key features of Gazebo include:

- multiple physics engines.
- a rich library of robot models and environments.
- a wide variety of sensors.
- convenient programmatic and graphical interfaces.

### 3.1.3 CarDemo

On Open Source Robotics Foundation, it has been found a demo that simulate a Prius in gazebo 9 with sensor data being published using ROS kinetic (ROS distro). The car's throttle, brake, steering, and gear shifting are controlled by publishing a ROS message. A ROS node allows driving with a gamepad or joystick. To run this demo, it is used Linux Xenial (16.04), gazebo 9, and ROS kinetic. [38]



| Property | Value |
|---|---|
| name | cafe |
| is_static | ✓ True |
| self_collide | False |
| enable_wind | False |
| pose | |
| link | cafe::link |

```
Steps: 1  Real Time Factor: 0.96  Sim Time: 00 00:31:46.212  Real Time: 00 00:04:36.880  Iterations: 262691  FPS: 62.45
```

Figure 3.2. A screen shot from the car demo simulation.

### 3.1.4 Developing of simulation

In Car Demo, a simulated world and a simulated Prius is created. Furthermore is present a node that get information from a joy-pad and move according the car. To control the Prius a custom ROS message id used. This ROS messages is called **Control** and has 4 parameters:

- **throttle**: it is a float64 type with a range that goes from 0 to 1 where 1 is the maximum pressure on gas pedal.
- **brake**: it is a float64 type with a range that goes from 0 to 1 where 1 is the maximum pressure on brake pedal.
- **steer**: it is a float64 type with a range that goes from -1 to 1 where 1 is the maximum turn on left, -1 is the maximum turn on right.
- **shift_gears**: it is a uint8 type with a range that goes from 0 to

3. Zero gives no command, one puts the shift to neutral so if the gear is in this position and the throttle is set to maximum the car actually does not accelerate, two puts the shift to forward so if accelerate the car goes forward, three puts the shift to reverse so if accelerate the car goes backwards.

The goal of this work is not to control the car with a controller but through UDP packets and a defined protocol. So, what is needed to do is a ROS node which listens on an UDP socket, receives a packet from the BCI with mentioned protocol, translates the command in the Control ROS message and publishes it. The address and the port on which the socket is listening are printed in order to know to which IP the packets are sent. Since, in the real car, is not possible to control the gears, the node always sends “shift_gears = 2” which makes the car go forward. This node is created with C++ because is a well-supported programming language by ROS and due to the big experience with it. In this way, it has been created a simulation that waits for commands over UDP and moves the simulated Prius accordingly (Figure 3.3).

Once developed this node, to understand if it really works (Xavier, the application had yet to be implemented), is created a really simple Android application that sends an UDP packets every 300 milliseconds. This application can drive the car with a couple of sliders: the first slide adjusts the acceleration (on the left: no acceleration, on the right: maximum acceleration), the second adjusts the steering (on the left: maximum left turn, at centre: no turn, on the right: maximum right turn). After some simulations has been understood that often, because of wrong commands given from the BCI, the car gets stuck against walls and side-walls. Then, a second node is create to move the car forward, backward, left and right through the keyboard arrows to quickly reset the position of the vehicle.

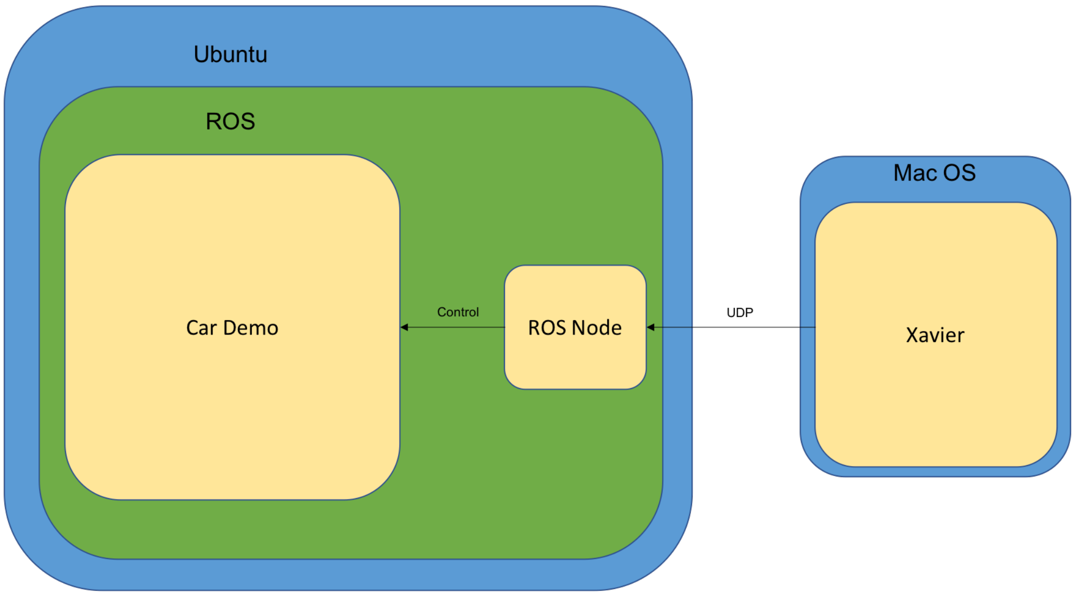


Figure 3.3. Simulation workflow. Able to control the simulated car with UDP packets.

## 3.2 Qt

Qt is a modular cross-platform framework, written in C++ and release in 1995, designed for developing applications for desktop and mobile devices, as well as embedded systems. Qt combines a cross-platform software development application, graphical user interface (GUI) framework, and a toolkit for developing apps using C++ standard.

This object-oriented cross-platform framework comes with Qt Creator – its own Integrated Development Environment (IDE) that runs on Linux, Android, Windows, OS X, and other systems. It provides developers with all the functionality required to build cross-platform software. Considering the fact that Qt uses C++, it is entirely object-oriented and supports extensible and true component programming. GUIs can be written directly in C++ using its Widgets module, as it comes with Qt Designer – an interactive graphical tool that is integrated into Qt Creator. Qt cross-platform software development network uses C++ (on a server side) along with QML (on the front-end side). Qt QML is a specific programming language (Qt Meta Language) which is similar to HTML. It is used to create cross-platform libraries and applications. Qt QML provides both QML and C++ Application Program Interfaces (APIs), which enables app developers to integrate QML code with JavaScript and C++.

Qt QML is designed to be simply extensible via C++ code. The classes in this module make it possible for QML objects to be directed from C++, meanwhile, the nature of QML engine's integration enables C++ functionality directly from Qt QML. This supports the development of hybrid apps implemented with a blend of C++, QML, and JavaScript code. [32]

### 3.2.1 Advantages of using Qt app development platform

- Qt app development allows porting an application to multiple platforms through simple recompilation.
- It increases development productivity and decreases time to market, making applications future-proof.
- Developing with Qt simplifies technology strategy and, ultimately, reduces costs.
- Saves time, through one code deployed across all screens and platforms.
- Coding in C++ gives developer a greater control and the possibility to work with numerous existing libraries.
- The code is compiled to native binaries that run at full speed (no need to use a virtual machine).
- Qt also has a very capable IDE in Qt Creator which works on all platforms and gives the same development environment wherever it's used.
- Qt cross-platform software development makes it easy to create intuitive experiences for all users, no matter what system is used.

However, there are two properties of Qt that has been the subjects of discussions over the years. The first one is its usage of a pre-processing step called the "Meta Object Compiler". It has sometimes been described as not standard C++, but this is a misunderstanding. What MOC does is parsing the header files for some predefined macros and generating C++ code to support features like signal/slots, introspection and property system. This makes Qt a strong candidate for new projects of medium to large sizes and all variations, as demonstrated by the examples from automotive, entertainment, automation, medical and other industries. [33]

## 3.2.2 Signals and Slots

Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks. Signals and slots are made possible by Qt's meta-object system. In Qt, there is an alternative to the callback technique: using signals and slots.

A signal is emitted when a particular event occurs. Qt's widgets have many predefined signals, but it is possible always subclass widgets to add own signals to them.

A slot is a function that is called in response to a particular signal. Qt's widgets have many pre-defined slots, but it is common practice to subclass widgets and add own slots so that is possible to handle the signals to be considered (Figure 3.4).

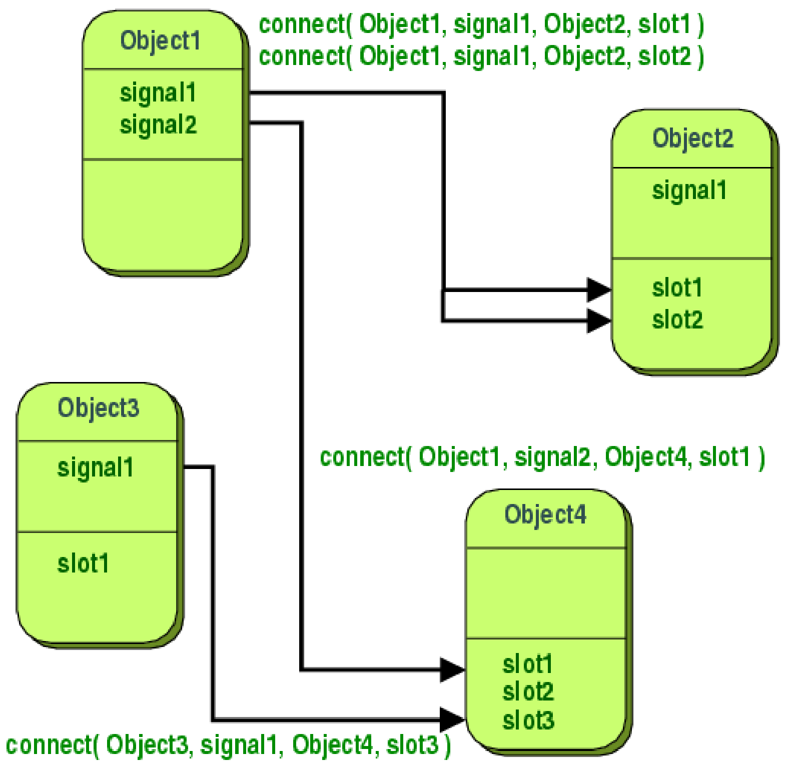

Object1
signal1
signal2
connect( Object1, signal1, Object2, slot1 )
connect( Object1, signal1, Object2, slot2 )
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect( Object1, signal2, Object4, slot1 )
Object4
slot1
slot2
slot3
connect( Object3, signal1, Object4, slot3 )

Figure 3.4. Qt - Signals and Slots mechanism

The signals and slots mechanism is type safe: The signature of a signal must match the signature of the receiving slot. (In fact a slot may have a shorter signature than the signal it receives because it can ignore extra arguments.) Since the signatures are compatible, the compiler can help to detect type mismatches when using the function pointer-based syntax. The string-based SIGNAL and SLOT syntax will detect type mismatches at runtime. Signals and slots are loosely coupled: a class which emits a signal neither knows nor cares which slots receive the signal. Qt's signals and slots mechanism ensures that if a signal is connected to a slot, the slot will be called with the signal's parameters at the right time. Signals and slots can take any number of arguments of any type. They are completely type safe.

All classes that inherit from QObject or one of its subclasses (e.g., QWidget) can contain signals and slots. Signals are emitted by objects when they change their state in a way that may be interesting to other objects. This is all the object does to communicate. It does not know or care whether anything is receiving the signals it emits. This is true information encapsulation, and ensures that the object can be used as a software component.

Slots can be used for receiving signals, but they are also normal member functions. Just as an object does not know if anything receives its signals, a slot does not know if it has any signals connected to it. This ensures that truly independent components can be created with Qt. It is possible to connect as many signals as required to a single slot, and a signal can be connected to as many slots as needed. It is even possible to connect a signal directly to another signal. (This will emit the second signal immediately whenever the first is emitted.)

Together, signals and slots make up a powerful component programming mechanism. [34]

### 3.2.3 MOC vs. template

Templates are a builtin mechanism in C++ that allows the compiler to generate code on the fly, depending on the type of the arguments passed. As such, templates are highly interesting to framework creators, and advanced templates are used in many places in Qt. However, there are limitations: There are things that you can easily express with templates, and there are things that are impossible to express with templates. A generic vector container class is easily expressible, even with partial specialisation for pointer types, while a function that sets up a graphical user interface based on an XML description given as a string is not expressible as a template. And then there is a gray area in between. Things that you can hack with templates at the cost of code size, readability, portability, usability, extensability, robustness and ultimately design beauty. Both templates and the C preprocessor can be stretched to do incredibility smart and mind boggling things. But just because those things can be done, it does not necessarily mean doing them is the right design choice. Code, unfortunately, is not meant to be published in books, but compiled with real-world compilers on real-world operating systems. [35]

- The syntax matters: The syntax used for Qt's signals and slots is intuitive, simple to use and easy to read.
- Code generators are good: Qt's moc (Meta Object Compiler) provides a clean way to go beyond the compiled language's facilities. It does so by generating additional C++ code which can be compiled by any standard C++ compiler. The moc reads C++ source files. If it finds one or more class declarations that contain the Q_OBJECT macro, it produces another C++ source file which contains the meta object code for those classes. The C++ source file generated by the moc must be compiled and linked with the implementation of the class. Typically moc is not called manually, but automatically by the build system, so it requires no additional effort by the programmer.
- GUIs are dynamic: C++ is a standarized, powerful and elaborate general-purpose language. It's the only language that is exploited on such a wide range of software projects, spanning every kind of application from entire operating systems, database servers and high end graphics applications to common desktop applications. One of the keys to C++'s success is its scalable language design that focuses on maximum performance and minimal memory consumption whilst still maintaining ANSI C compatibility.
- Performance: Qt's signals and slots implementation is not as fast as a template-based solution. While emitting a signal is approximately the cost of four ordinary function calls with common template implementations, Qt requires effort comparable to about ten function calls. This is not surprising since the Qt mechanism includes a generic marshaller, introspection, queued calls between different threads, and ultimately scriptability. It does not rely on excessive inlining and code expansion and it provides unmatched runtime safety. Qt's iterators are safe while those of faster template-based systems are not. Even during the process of emitting a signal

to several receivers, those receivers can be deleted safely without the program crashing. Without this safety, the applications would eventually crash with a difficult to debug free'd memory read or write error.

- Flexibility: C++ with the moc essentially gives us the flexibility. There are other useful aspects like dynamic qobject_cast<T>() mechanism that does not rely on the system's RTTI and thus does not share its limitations; dynamic meta objects. [35]

## 3.3 Xavier

As already mentioned, the goal of this thesis is a program that can drive a car according to mental commands, which are given by a user through a BCI. Since the tests with Emotiv BCI showed difficulties controlling the mental commands, the first version of Xavier is created using only the facial expression whose training is much easier.

This program had to provide an intuitive UI to manage the login (that is mandatory to use cortex API and CortexUI, as already explained), to manage the training for the facial expressions and have a live mode where the facial expressions are read from BCI and send to the car through UDP packets.

Emotiv provides a sample code without GUI that calls API to train some mental commands and facial expressions. Actually, there are two different codes, because this program is written in two programming languages: one in C# and the other one in Qt C++. It is needed a program that could work in Windows, Mac OS and Linux. The cross-platform feature was essential in order to choose the programming language. Given the good knowledge of C++ and the example program in Qt C++, this is the final choice language, a cross-platform C++ framework.

### 3.3.1 GUI

The GUI of Xavier is very simple. There are just two windows: one for the training and the other one for live mode that has the purpose of reading info from BCI and send the commands to the teleoperated car.

The first window, called train, is divided in two parts: the first part allows to start the training with a new or with an old profile and get a connection with the BCI. The second part manages the training of facial expressions.

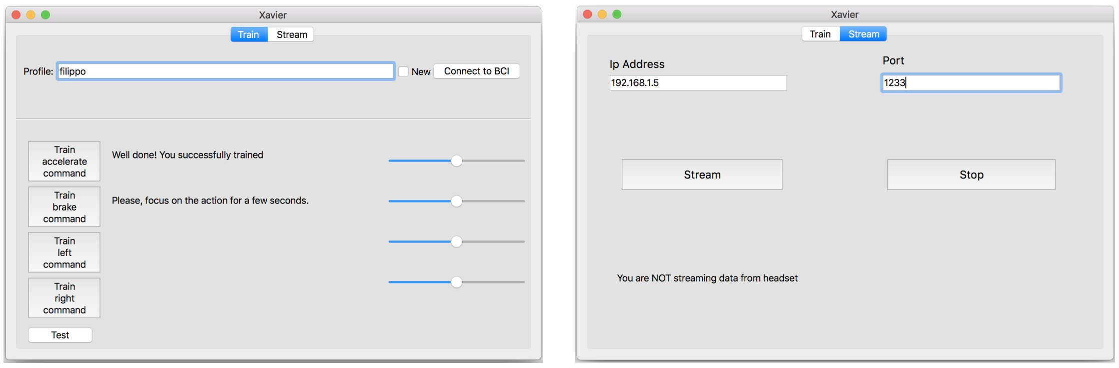
To use this program, it is required to login in CortexUI, connect the headset to the latter and set-up the BCI. Then open Xavier, add a profile name, set a check-box and say if the profile was already existing or if it was new (checking the box means that the profile is new) and at the end press the "Connect to BCI" button to login with the Emotiv server. Afterwards, looking for the connected BCI and set up the new/old profile.

The lower part of the first window is to train the facial expressions. Actually, with this version for each command, there is a fixed expression. There are four train buttons and each of them trains a specific facial expression that corresponds with a command for the car:

- accelerate command: clench
- brake command: neutral
- left command: smirk left
- right command: smirk right

By pressing one of these buttons a label appears which communicates to the user the state of the training like "Please, focus on the action", "The train is successfully completed" or "Train is failed". At the bottom, there is a button called "Test". By pressing this button, the program starts to "read" the facial expression from the BCI and shows the read expression in a label. In this way, the user can evaluate the quality of the training and eventually modify the sensibility for each expression through four sliders. If the user moves the first slider to the right the sensibility of the clench expression increases.

Once the training is completed successfully the user can switch to the "Stream" window that is composed by four main objects: two line-edits where the address and the port, on which the car is listening, should be inserted and two buttons, one called "Stream" to start to send commands to the car and the other one called "Stop" to stop the sending. There is even a label that shows a message in case of error (Figure 3.5).



Profile: filippo

Well done! You successfully trained

Please, focus on the action for a few seconds.

Ip Address: 192.168.1.5

Port: 1233

You are NOT streaming data from headset

Figure 3.5. GUI of Xavier. On the left the training window and on the right the streaming one.

## 3.3.2 Code structure

The program is divided in two threads: the first contains the Mainwindow, an object that handles the GUI and all the user iterations and gives him feedbacks from the BCI. On the other hand, the second thread contains all the login information of the program and connects the user iterations that come from Mainwindow, the BCI and the car. In the second thread operates the Binder, an object that contains two main objects: CortexClient that handles the communication between Xavier and the Emotiv Server and Command that handles the sending of commands to the teleoperated car (Figure 3.6).

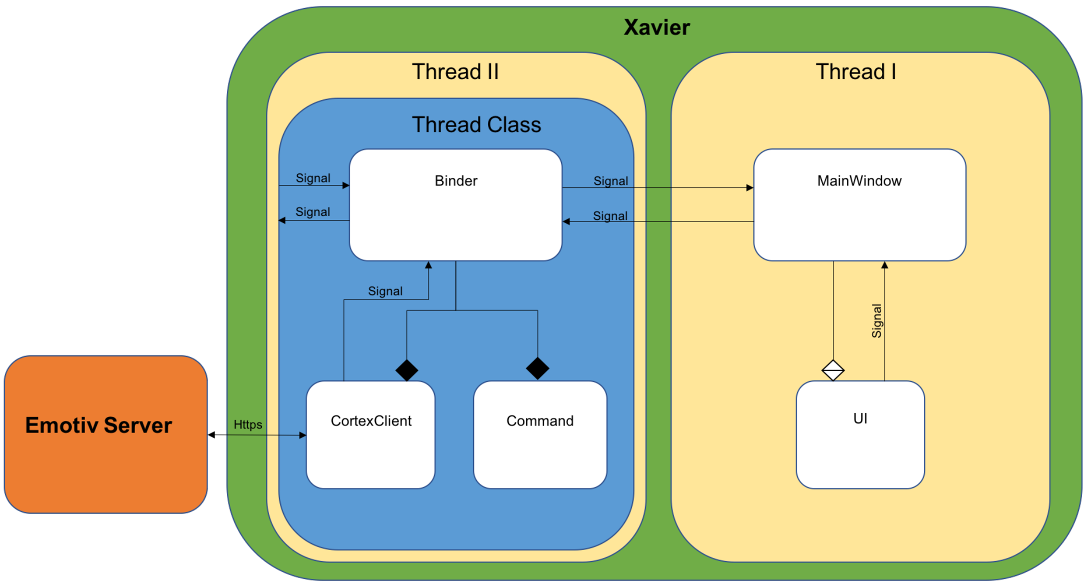


Figure 3.6. Scheme that shows the structure of Xavier, communication between classes and communication with the Emotiv Server.

### Mainwindow

This class is simple, inherits from QMainWindow. It contains, as private member variable, just a pointer to UI::MainWindow, the class that handles the UI elements and a Boolean which is needed for the closure process.

Mainwindow contains many signals and slots, that can be divided in two types: the ones that are triggered by the UI and the ones that are called by Binder's signals. The slots of the first type are called when for example the "Stream" or "Training" buttons are pressed or the sensibility slider has been moved from a position to another. If necessary, these slots get other information like in the case of the "Stream" button. The relative slot reads the values of "IPaddress" and "Port" text forms and then calls a signal to inform the Binder about the new iterations of the user. The second type of slots is the ones which are transmitted by the Binder and which inform the user that the connections with the BCI and Emotiv cloud are ready or that the connection has been closed. With these slots are also received error messages that are displayed in the UI by MainWindow or the facial expression read from BCI that it is displayed as well.

Since Xavier is a program with two threads, both threads need to be closed and so free all the resources before completely closing the program. In other words, it is needed to establish a communication between the two threads where the master informs the slave that it has to close itself and the slave has to say to the master when it is closed. To do so, the MainWindow handle a closure process. When "closeEvent" is received (the user clicked the red cross button to close the program) MainWindow checks if "_closeMainWindow" Boolean is "true" (this variable is initialized to "false"). If the value is true, it closes the program. Otherwise it ignores the event (the program is not closed) and sends the "closeBinder" signal to Binder that closes everything and sends back a signal to inform MainWindow that it is closed. When the "onBinderClosed" slot is called, triggered by "closeBinder", "_closeMainWindow" is set to "true" and "closeEvent" is called and will finally close the program.

### Binder

This class is the most complex. It handles the communications between the user interactions, which are received by the signals of MainWindow and Emotiv Server.

It has many private member variables. Indeed, this object must keep in memory a lot of information such as headset id, Emotiv token authentication, session id, profile name, an UDP socket, the port and the IP address of the car (or of the simulation) the training status of the four commands, a timer to send packets after a certain amount of time, an instance of the Command class and pointers to other classes, like CortexClient, HeadsetFinder, SessionCreator.

This class has many signals, some of them are connected with a thread class to bind the life cycle of the Binder with the thread's life cycle, others are connected with the MainWindow to inform it about its status and still others are connected to the CortexClient object that handles the communication with Emotiv Server. Since it is required an efficient communication within the three main classes that receive Binder's signals, as already described above, this class has many slots that are triggered by signals sent from MainWindow, Thread, or CortexClient.

The behaviour of the Binder can be split into three phases: the setup phase, the training phase and the streaming phase.

The first phase starts when the user inserts the username and the password and presses the "Connect to BCI" button. Mainwindow then checks the information provided from the user and sends the "start" signal that triggers the "onStart" Binder's slot. This callback saves the profile information and starts the connection with the Emotiv cloud by sending signals to the CortexClient. When the connection has been established, CortexClient triggers a new Binder's slot which searches with the HeadsetFinder object for a connected BCI. HeadsetFinder is a class which wraps inside itself a series of communications with CortexClient to find the headset. When the headset has found another Binder's slot, it is called by the HeadsetFinder, the resources of the latter are freed and a session has to be created. For that, it is used the SessionCreator object that is similar to the HeadsetFinder and handles a set of communications with CortexClient to establish a communication. After the latter is set, a signal from SessionCreator is sent and another Binder's slot is triggered. This slot is called "onSessionCreated". It saves just received information as the authentication token and the ID of the session and sets up a profile by sending a signal to the CortexClient. When the profile has been set up on Emotiv server the "onLogged" Binder's slot is triggered that sends the "ready" signal to the MainWindow to say that the BCI is ready to be trained. Of course, during this communication many errors can happen and in this case, at first, an error message is sent through Qt signal to the Binder and then forwarded to MainWindow which displays it. With the "ready" signal the setup phase is completed.

The second phase, the training phase, starts when the user presses the train button of one of the four commands. The Binder receives through a slot called by the MainWindow, which manages the iterations of the user, the information that a command must be trained and which is the specific command. The first task the Binder has to do is to subscribe, if this has not already happened, to one of the data streams. In the case of training this is the "sys" data streams. When the subscription is completed, the Binder calls the training signal that triggers a slot of the CortexClient. In this signal, some information about the type of command ("mental command" or "facial expression") and the command itself are coded. The CortexClient forwards this request to the Emotiv Server. After this request CortexClient receives some stream data about the status of the training. These data are sent to the Binder and then displayed in the UI by the MainWindow to show the user the status of the training. The statuses are for example: "please focus on the action for a few seconds", "Training completed" or "Training is failed. Try again". After the end of the training, either it was successful or not, the second phase is completed.

The third and the last phase is the so called "streaming phase" and it starts when the user switches to the "Stream" tab and presses the "Start Stream" button. In this case, a slot of Binder, triggered by the MainWindow, is called and information, such as the port and the IP address of the controlled car, are passed to the Binder. So, the Binder calls the subscribe signal with the "facial expression" parameter to receive stream data from the headset about the type of facial expression that the user is doing. Then the Binder sets a timer that every 200 milliseconds triggers a Binder slot called "onTimeout". In this way, the Binder will be awakened up cyclically by the "onStreamDataReceived" and "onTimeout" slots. The first is called when a new data is available from the headset and the second one when the timer counted 200 milliseconds. With "onStreamDataReceived" the Binder reads which is the current facial expression and saves it in the Command object. The functionality of this class will be explain later. When the "onTimeout" slot is called, the Binder calls a function of the Command to get a byte array that contains the command, which is coded with the protocol given by Luxoft, that will be send to the car. These bytes will be sent, using a socket, with a UDP packet to the IP address of the car. This phase keeps going on until the user presses the "Stop Stream" button.

Then the subscription is closed and the timer is stopped.

### CortexClient

This class manages the communication between Xavier and the Emotiv servers. The work of this class can be divided in two: the sending of requests and the receiving of the answers from Emotiv. The initialization of this class starts when in the first phase of the Binder the open slot is called and CortexClient instantiates a Web Secure Socket connection with the servers.

All the other slots of this class are slots which are used to send requests to Emotiv such as "authorize", "createSession", "subscribe", "training", etc. When one of these slots is called, CortexClient creates a JsonObject that is filled with the information which is required for the request. CortexClient then adds information regarding the JSON-RPC protocol and sends it to Emotiv. One important aspect of the JSON-RPC protocol is the ID of the request: the ID has to be increasing and for each request and there will be a response with the same id. For example, if it is sent an "authorize" request in the json file, there will be a key called "id" with the value x. Then, a response will be received for that request with the same id value (x). So, for every response received, it will be known the corresponding request.

When CortexClient receives a response from the servers, it first checks if an id is present or not. If it is not present, that means that there is no JSON-RPC communication and that the message is a stream message. Therefore, the signal "streamDataReceived" is called. These messages are received when there is a subscription to train or to stream the recognized facial expression. Otherwise, if there is an id, CortexClient checks which is the corresponding request and acts accordingly.

For example, the json file received has "id 14". Before a request of authorization with the id 14 has been sent, this json file must have a key called "_auth" whose value is the authentication token. CortexClient so, sends a signal called "authorizeOk" and it informs the Binder that the authorization is valid, and which is the authentication token. These are the functions of this class which does the processing of the input and the output lightening the work of the logic of the Binder.

### Command

As already said, the Binder receives with the “onStreamDataReceived” a facial expression, which is before “read” by the headset, it saves the expression in the command object. This class has a public method called “setCommand”, with which the Binder passes as parameter the facial expression, and then checks if that expression corresponds to a command for the car. If so, it sets a value for the corresponding command. If the command is “accelerate”, it sets the gas pedal to a value of half pressure of the pedal. If the command is “brake” the brake pedal value is set to the max value (maximum pressure). However, when the command is “left” or “right” a liner function is used. At the beginning, the angle value is set to approximately 17 degrees. If the same command is received a second time, the angle is set to 34 degrees and so on. This increment follows a linear function (Figure 2.11), until the maximum value, which is 440 degrees, is reached.

The second public method of Command is a function that receives the value of the commands, previously set with the “setCommand” method, and returns them coded with the protocol that is necessary to communicate with the teleoperated car. Theses coded commands will be sent by the Binder with a socket to the car (or to the simulation).

## 3.4 Test with simulation

This test is done with two computers. The first computer works with Linux Xenial, runs ROS, launches Car Demo and the node that receives the UDP packet to control the car. The second computer, with Mac OS (in this case actually the OS is not relevant since Xavier has been written with a cross-platform language), is connected via Bluetooth with the headset and runs CortexUI and Xavier. To communicate, these two PC must be in the same intranet and the second one (the one which runs Xavier) must have access to Internet to communicate with the Emotiv servers (Figure 3.7).

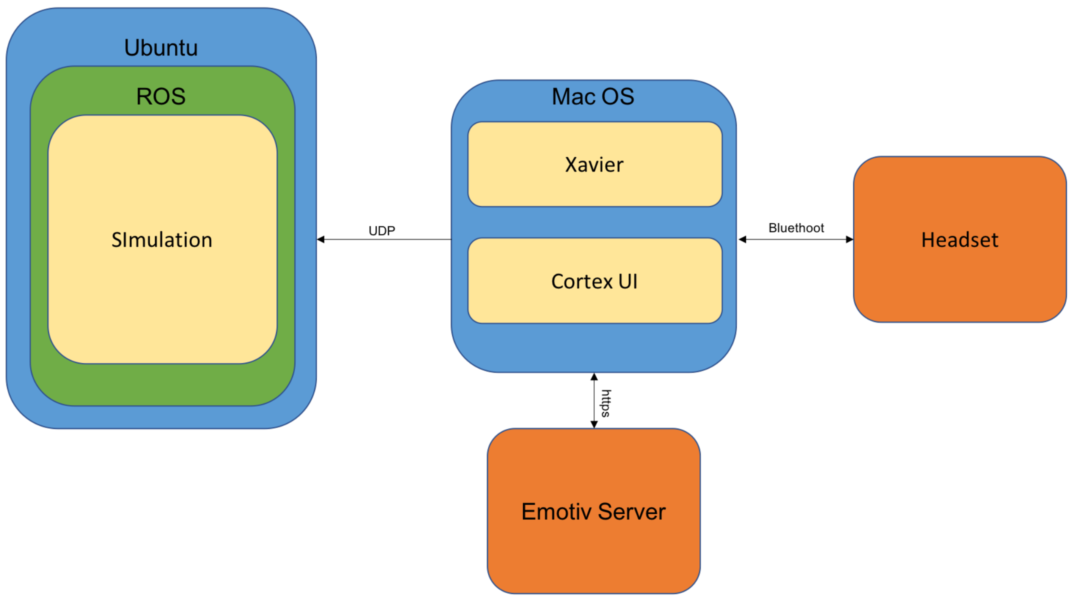


Figure 3.7. Architecture of the communications for the ROS simulation.

A lot of problems are discovered and encountered in the first test because of many bugs, but that is exactly the reason why a simulation has been created and not used directly the real car. Some bugs are fixed very quickly and then Xavier is re-launched.

The first conceptual error found was that in a normal driving, the pedal to accelerate is pressed and, in the meantime, the steering wheel is turned or straightened. It means two commands at the same time to the car. With the BCI and then with Xavier, in this moment it is not possible to give two commands simultaneously. For example, if from the BCI is received that the user is smirking left, so can be sent to the car to turn the steering wheel to the left. If is received the signal clenching so can be sent the accelerate command, but the two information together are never received. What happens is that the car turns the wheels but without an acceleration, then it does not actually make a curve.

## 3.5 Test with teleoperated car

The first tests with the car are done in a park. The aim is just to test if the program uses the communication protocol correctly and how intense the gas pedal pressure should be kept to maintain the speed of the car between ten and fifteen kilometres per hour.

A personal computer is connected with Xavier to the computer that controls the car through an Ethernet cable and to a smartphone with Wi-Fi, since CortexUI and Cortex, as already explained, need an internet connection and a connection to the headset via Bluetooth.

Since it was not possible to get the speed value from the vehicle and adjust the pressure on the pedal accordingly, it is empirically understood which value of pressure has to be given to the gas pedal to keep the car at the desired speed. To calculate this value, just the accelerate command is trained with Xavier and the source code is changed manually to set the pressure. After a few tries, the most appropriate value is approximately ten percent of pressure on the pedal.

After this acceleration test, the test of the steering starts. For safety reason, the “accelerate” command is disabled from the car side. The left command is trained and the application starts to send commands to the car.

The first small issues was the wrong direction of the commands: by giving the command “left” the car turned right and vice versa by giving the command “right” the car turned left. That was a very easy bug to fix, fixed even during the simulation, indeed with the simulated car this issue was solved, since both software were affected with the same bug. A bigger problem was the setting of the steering angle: as explained earlier, the sent steering value is based on a linear function (Figure 2.11). The first time a left or right command is received, it is sent to the car to turn the steering wheel for 17.6 degrees, which is a too high value. In fact, such a big delta on the angle value made the driving uncomfortable and noisy. Furthermore, on could potentially break the mechanic system that turns the steering. Therefore, it is needed to send messages more frequently but with less delta angle (Figure 3.8).

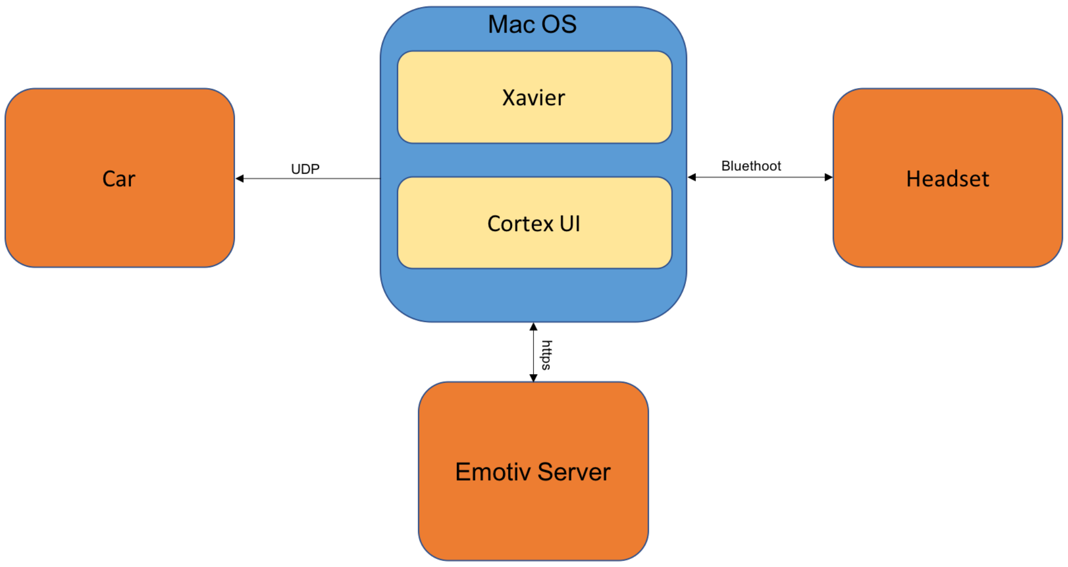


Figure 3.8. Architecture of the communications for driving the car with the headset.

# Chapter 4

# Mental commands and Xavier update

In this chapter, it has presented the steps taken to create a new version of Xavier that could drive a car also with mental commands. The first part of the chapter explains all the studies and the attempts done to master these type of commands and their training. In the second part, it has described the functionalities and how to develop Xavier 1.0. Besides, how to solve the problems encountered in the tests with the previous version of the software.

## 4.1 Training patterns

It can be normal to not get success fastly and shortly; a lot of difficulties have been recognized and for that in this experiment, several training patterns have been analysed and tested in order to improve the performance of the training and the quality of the commands. Below it is showed which training patterns can be followed and what has to be chosen to get better performances.

### 4.1.1 Focus on the actions

The first and immediate pattern for the user is to try to be focus on the action that wants to achieve. It is the easier way but not always the best one.

Key things about mental commands training are relax, so the algorithms don't have to cope with variable muscle signals from straining and keep the visualisations as consistent and distinct as possible.

Novices tend to strain hard for all actions, which is a challenge for the signal extraction systems, and they also tend to think “DO SOMETHING pull” or “DO SOMETHING push”, which makes the signatures difficult to distinguish. More experienced users tend to be more comfortable and confident and can invoke the visualisations almost unconsciously. An example could be to imagine a scenario in which your action represents the command.

The graph below in the Figure 4.1, the results got are not very perfomant. This is caused by the stimulation of identical zone of the human brain and then, very often, it recognises different thougts as equals.

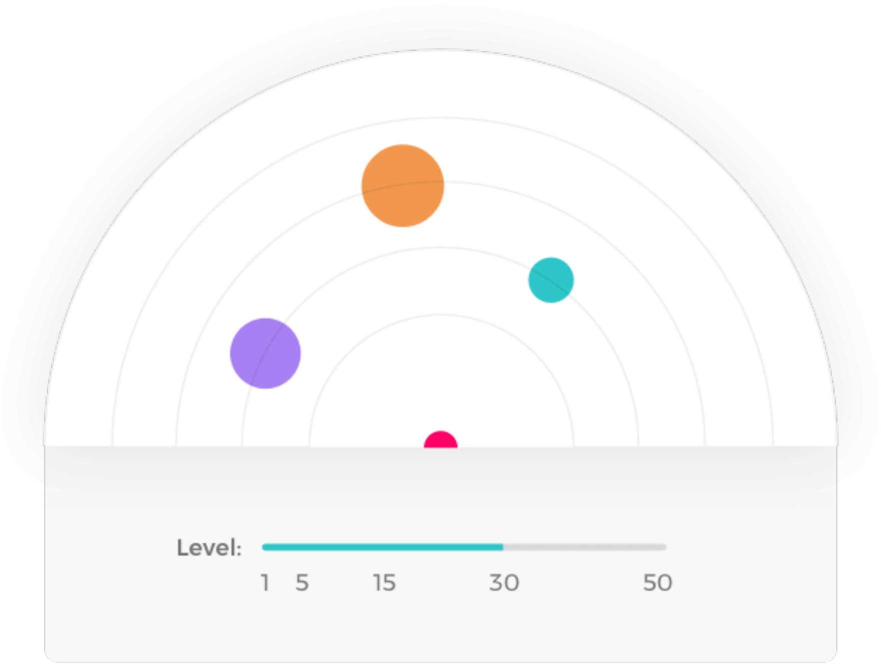


Figure 4.1. Output of data for trained commands based on actions.

## 4.1.2 Focus on the colours

Another surprising pattern is to try to be focus on colours. Colours have a direct impact on the brain and our thought process. Different colors are known to trigger different emotions in us and hence, they are associated with different human attributes affecting brain waves. Colours evoke a person’s emotional quotient in different ways. The color is a kind of an energy that affects both the functions of human body as well as the mind and the emotions.

During the training, thinking only about a command, the green colour is set for push and it's easy to notice how much is different and far from the neutral state as showed in the Figure 4.2.

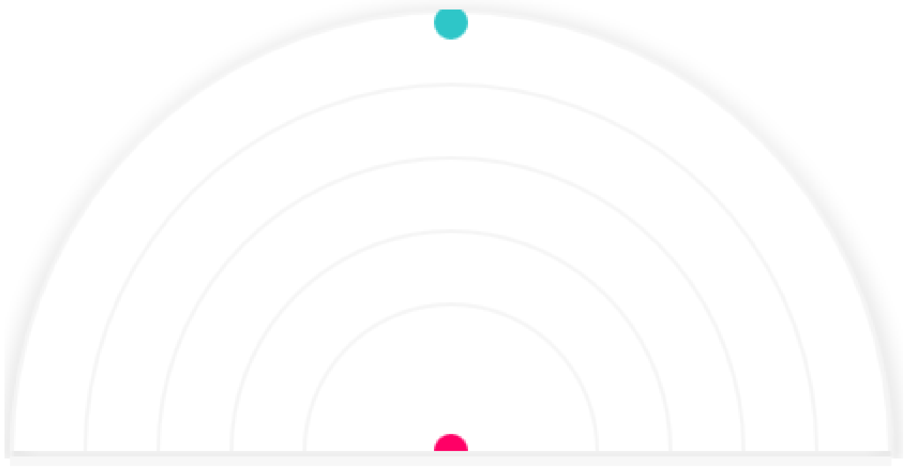


Figure 4.2. Output of data for a trained command based on colour.

Then, the full training have started adding more commands; the neutral is still set for brake, the yellow is set for left and the orange for the right; it has tried but avoided the red colour because very often it has been associated to stop or danger situations.

After several trainings it has been noticed that two colours still can be recognized in a pretty way but more colours start to homogenise all the training. The dots on the map (Figure 4.3) become closer together, so some relevant information about the different commands are lost. It is relevant to underline that very often the strict link between thought of colours and the the related emotions can influence the training patterns.

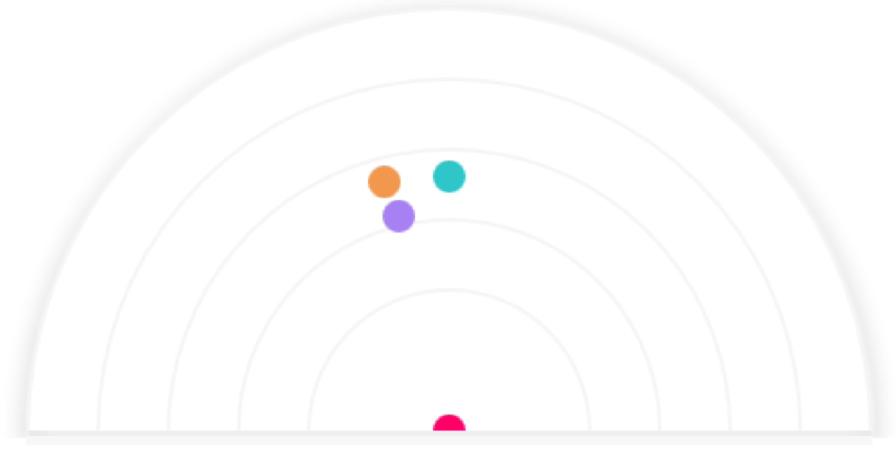


Figure 4.3. Output of data for trained commands based on colour.

### 4.1.3 Focus on the Math/Logic/Memory

Human people are born with the ability to count: Shortly after birth, babies can estimate the number of events and even perform simple calculations. But what exactly happens in the brain? Studies were able to demonstrate that some brain cells fire mainly for quantities of three, others for quantities of four and others for other quantities. A similar effect can be observed for digits: In humans, the neurons activated in response to a ”2” are for instance not the same as the neurons activated for a ”5”.[39]

Results show activity in parietal and frontal cortices, core areas related to mental-arithmetic and a very fast response of our brain to those signals. (Figure 4.4)

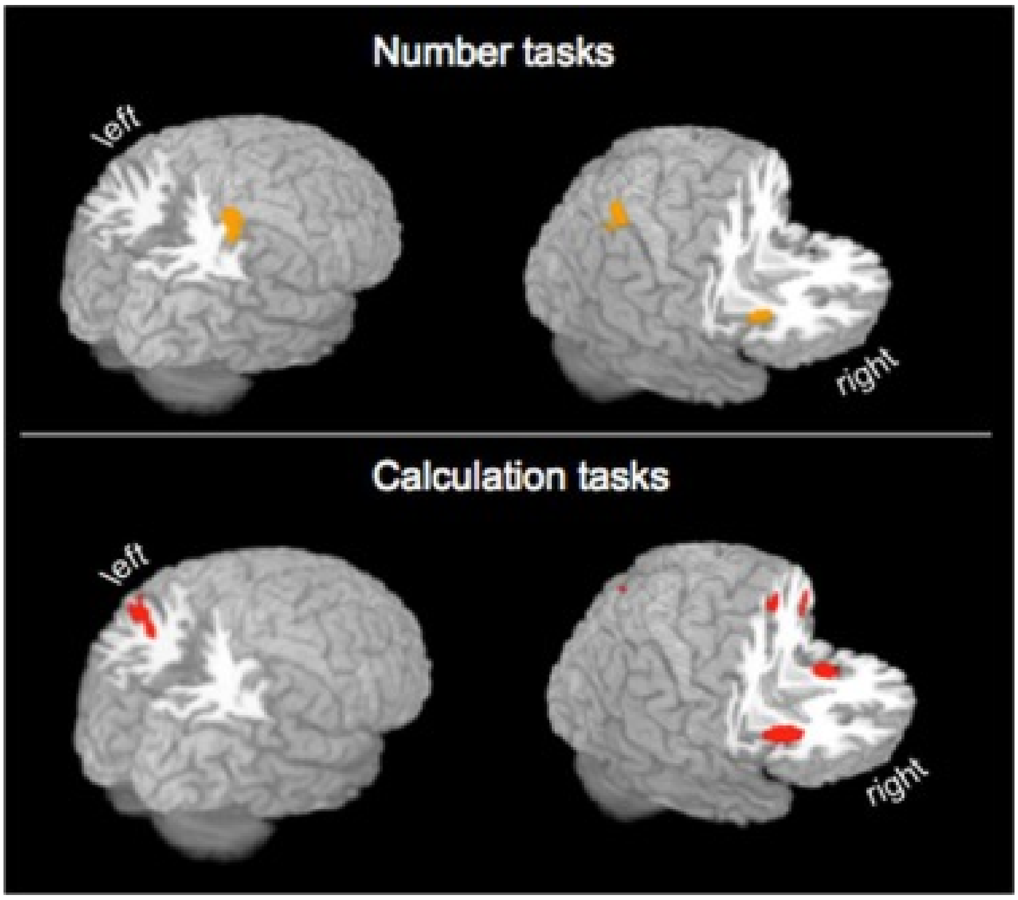


Figure 4.4. Involved brain area for mathematical or logical inputs.

The first exercise is to set the first command (here the push one) as a random number and consequently set the others commands as different ones; the results of a single cycle training are good but several tests show that they are only pretended because repeating this kind of training brings to the same results, so to an unusable dataset.

The second exercise is to set the first command as a linear counting operation from zero and the second command as a counting backwards from one-hundred. This brings to a good improvement on the dataset but it decreases as start to act mechanically using the right hemisphere (responsible for the associative heuristic to cope fast with easy task when there is a high level of familiarity) and not stimulating the arithmetic area of the brain (the left hemisphere) used for problem solving.

In the third exercise, the experiment involves the multiplication tables, in the first moment thought as a mnemonic operation and later as a real mathematical calculation. In this case, two commands are very good recognized, three are still keeping an usable rate but again for four commands the signals become a bit messy (Figure 4.5).

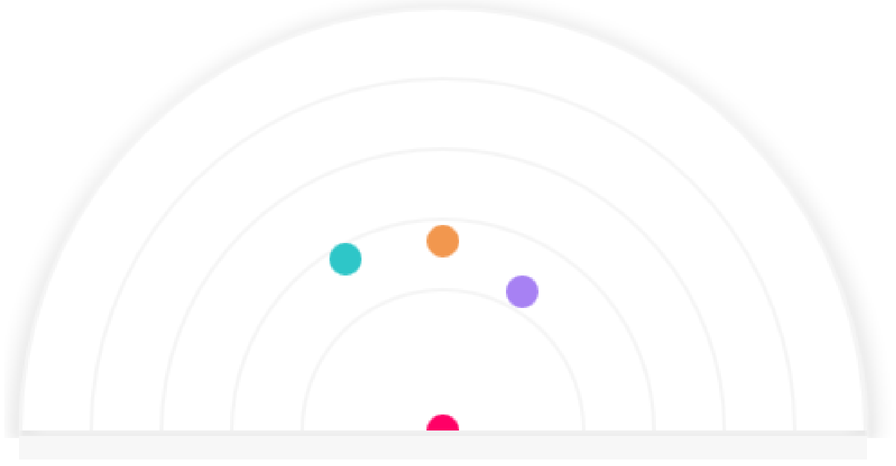


Figure 4.5. Output of data for trained commands based on mathematical concepts.

### 4.1.4 Focus on mixed patterns

Training for an action can take up to 8 hours of training before the system recognizes your specific pattern. From a biological point of view, it is relevant to highlight that even inputs from different zones of the brain can be translated as the same signals. For this pattern, it has been chosen the following commands: the green colour is set for push, the neutral state is set for brake, the counting backwards is set for turn left and a repetition of the chorus of a song. This is done to try to stimulate several parts of the brain (Figure 4.6) and extracting signals very far from each other and then get a clean and very productive training.

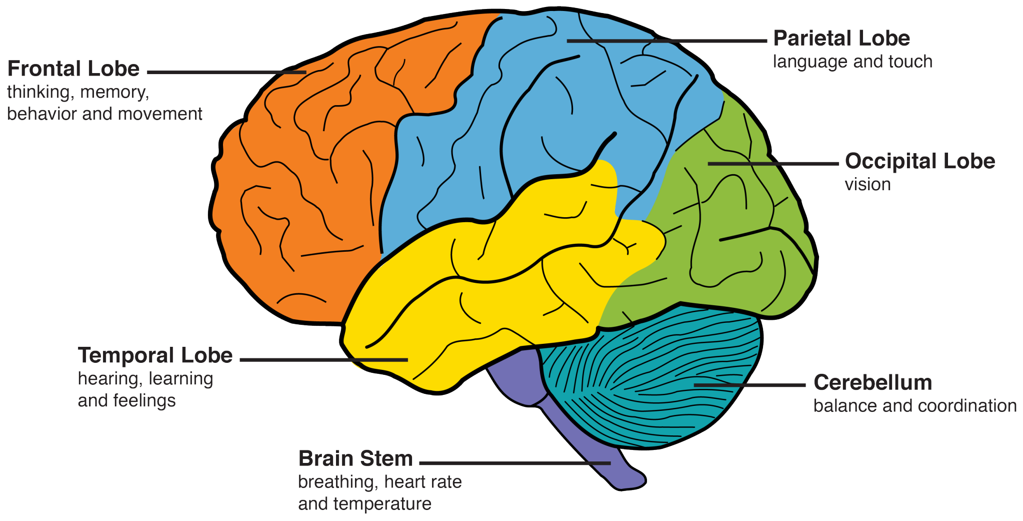


Figure 4.6. Brain lobes of human brain.

In this case, the training took several hours and it was tested even in different days. It has been considered as the slowest and the most complex training due to physical and mental fatigue. An alternation between neutral training and the command training every time, it brings a big improvement to the quality. In addition, as first step can be helpful to check the daily normal state of the tester brain in a focused and relaxed moments (Figure 4.7).

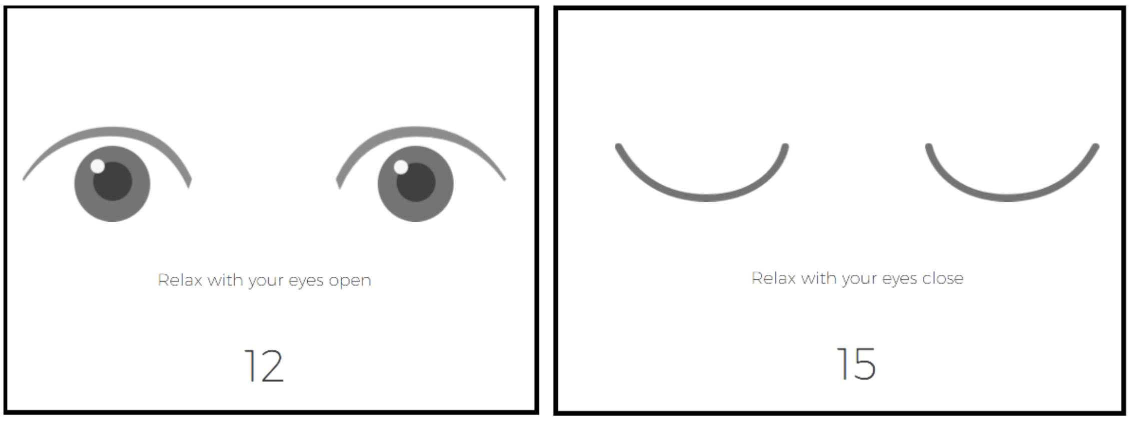


Figure 4.7. Daily normal state of the user brain.

In the Figure 4.8 it is showed the higher performances achieved with this kind of strategy.

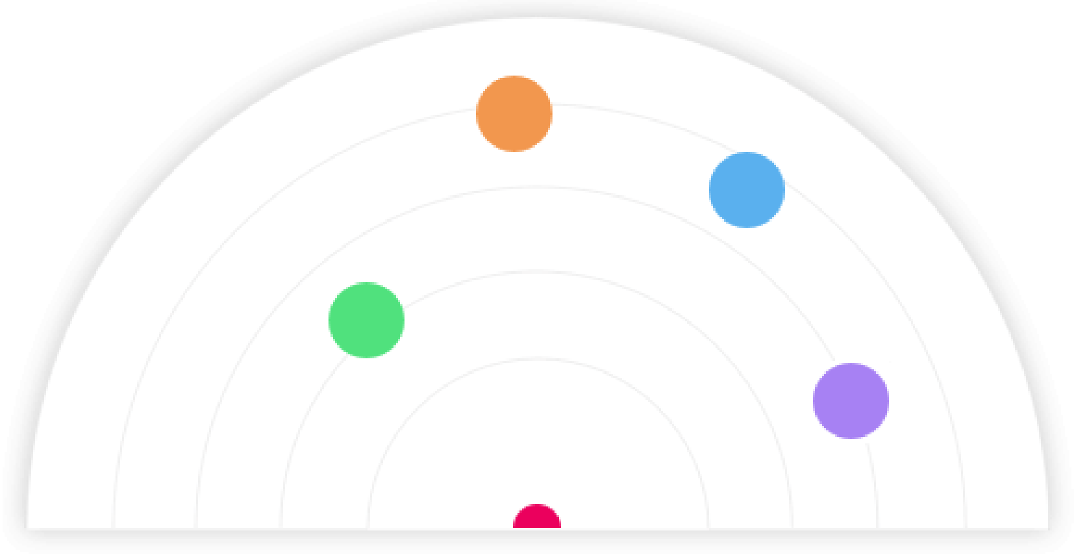


Figure 4.8. Output of data for trained commands based on mixed patterns.

### 4.1.5 Various training patterns

Use of movements: the use of physical movements to stimulate the brain was used. In particular, movements of the hand were chosen, due to its electrical activity. This kind of stimulus was applied in different ways. Dynamic movements were used, on which the user kept moving a limb following certain patterns or holding an item as a small ball in the hands. Even if the results can be much higher, it has been decided to not cover this pattern due to the eventual impossibility for disable people.

Emotional pattern: the use of feelings as fear, rage, sadness, surprise, etc. was used too. Also in this case, the translation from emotional feelings, stimulating different areas of the human brain, to commands brings to very good results but this pattern has been avoided because prone to the environmental context in the meaning that the command detected by the BCI is not the real intended one but it is due to external sources.

Mental deftness is a skill that will improve over time. As you learn to train distinct, reproducible mental states for each action, the detection becomes increasingly precise. The best results after training each action several times. It is important to not overstress the brain because can only get worse the quality of the training. The decision of what is the correct training pattern is strictly personal and it can take up to months to become an expert but it is important to not give up and never be discouraged.

## 4.2 Xavier v1.0

Version 1.0 is used as a major milestone, indicating that the software is ”complete”, that it has all major features, and is considered reliable enough for general release. The requested features start to be complete and the problems of the initial version are partially or completely solved. Small details are added and a completely revisited user interface has been created. The major feature introduced is the possibility to train and then to control the car with the mental commands.

### 4.2.1 Mental commands

Emotiv offers the opportunity for the user to create and execute several mental commands; in order to provide consistency and a simple range of possible actions, the Emotiv Cortex API offers up to 15 different kinds of commands.

Training a mental command is not easy at all so, to help the user in his training, in the application, right now, the Emotiv mental commands are chosen by default and the in the GUI are not selectable; there are four train buttons each of them trains a specific label that correspond with a command for the car:

- brake command: neutral
- accelerate command: push
- left command: left
- right command: right

The initial step in creating mental commands is to train the system to recognise your background mental state, the so-called NEUTRAL condition, by recording a brief period of your brain patterns while you are not trying to execute any commands.

Our brain changes frequently and the position of the headset could change how its shape is read, so this step is mandatory in order to get know the system how the brain patterns work in that time. To train a new mental command is enough to click the button and then imagining the consequences of the command given for 8 seconds (for example, imagine to push far an object for the PUSH command or try to make closer it for the PULL command) while the system records the mental patterns you want to associate with the command.

The user cannot test and practise the commands in the live mode if the neutral command and at least another command is trained. After a few repeated trials and as many training updates as the user wishes, the command is ready to be used and the training data summary is stored in the user profile. It is suggested to master an action and have good control before adding a the second action and so on.

Thanks to the graph below (Figure 4.9) is it possible to understand visually the results of the training. Different coloured dots represent the different commands and their distance to the center of the semicircle; the center represents the neutral state and more distance there is between the dots more accurate is the training.

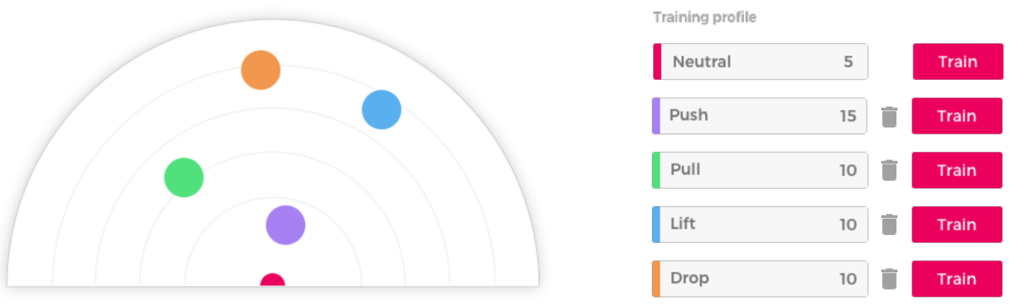


Figure 4.9. Output of data for trained commands respect to neutral state.

### 4.2.2 Frequency of the communication

In the version 0.1, the BCI, after reading the brain, sends every 200 milliseconds (5 Hz) the commands recognised. As a consequence of the real test on the track, it is considered appropriate to increase the sending frequency of the packets to 50 milliseconds (20 Hz) in order to decrease and minimise the latency and the delay of the communication to the car system.

### 4.2.3 GUI

The GUI of the program remains pretty simple but it's fully changed to make nicer and more user-friendly. It is similar to both the kind of the commands. So, also, for the mental commands, there are still only two windows, one for the training and one for the live mode. In this case, the training mode window has two tab widgets for Facial expressions and Mental commands, respectively.

In the main screen of the first window, there are four rows with the command signals mapped into the BCI signals. Every row is a QWidget that includes a QGroupBox of items containing the relevant information about the training: the selection of the command, a progress label showing what is happening (to keep focus on the training or if the operation is still or not more available), a label for counting the times a training is executed and successful, a label for the rate of the training shows a value in percentage of the quality of the training, a button allows to delete the training and discard all saved information. In the top-right corner, it is introduced the button "Logout" to close the session, delete the current profile and back to the login form. At the end of the commands list, there is a single button to enter the live mode and test the training done.

The selection of the command is one of the most important taken in consideration. Now it is possible to choose the command from a QComboBox among a range of available commands. The box becomes inactive only when at least a training exists already. In this case, for the mental commands the states by default are set up but not enabled for change the value (Figure 4.10). For the facial commands, instead, you can pick your favourite one from the list (Figure 4.11).

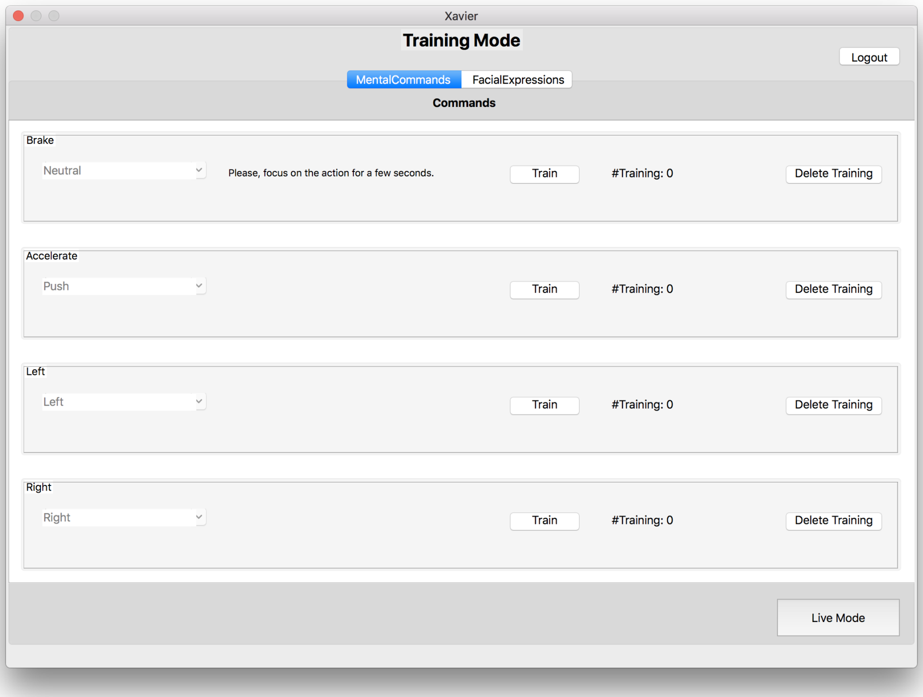


Figure 4.10. Xavier v1.0 - Mental commands: Training mode window.

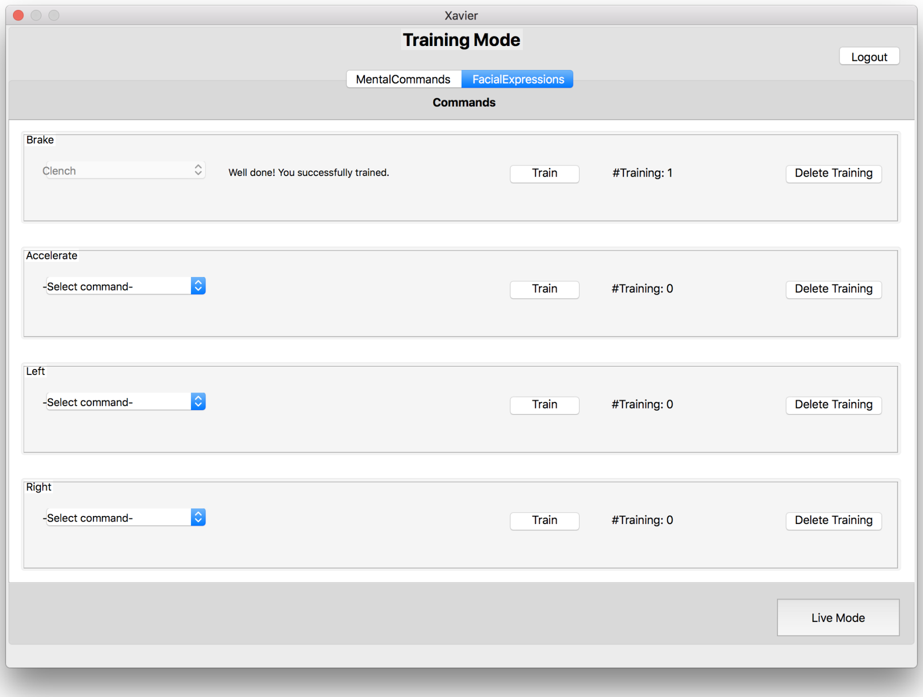


Figure 4.11. Xavier v1.0 - Facial expression: Training mode window.

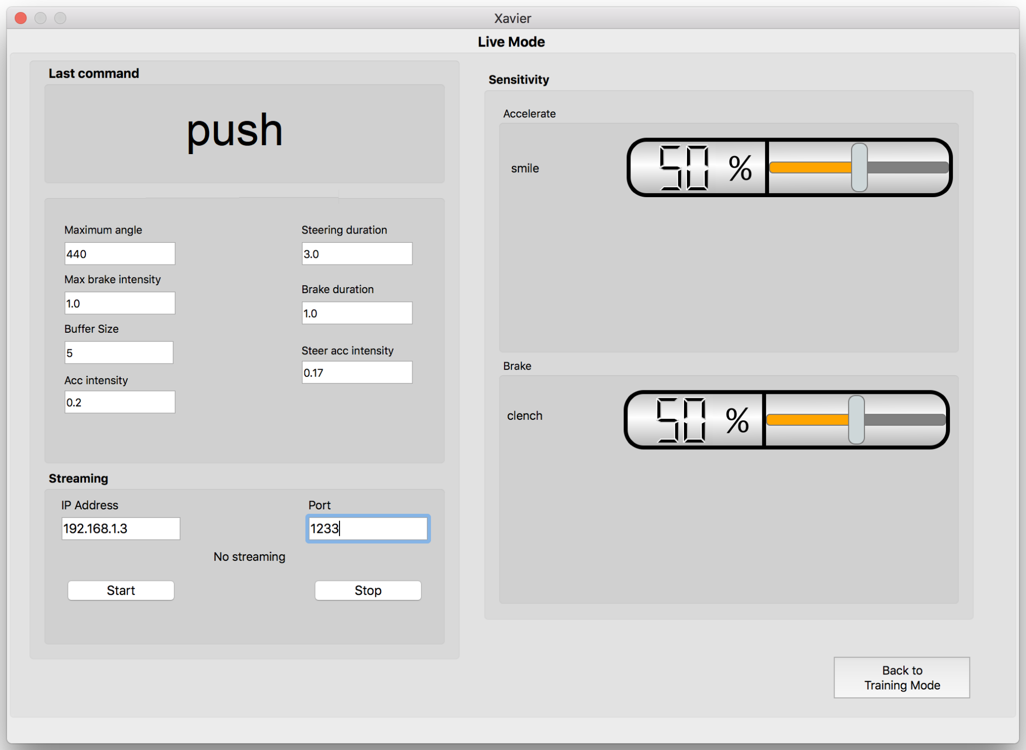
Also the live mode window has changed its aspect completely. The screen is divided in two parts. On the left side, there is an area showing the command read by the BCI (if any is detected or recognized it remains empty), an area to set the configuration parameters, an area to set the network parameters (IP address and port). At the bottom, a label shows the status of the live mode (if the streaming is active or not), two buttons to start and stop the live mode (Figure 4.12).

The parameters possible to set dynamically (but always before to start the live mode) are: the maximum angle of the steering wheel possible to reach, the steering and the brake duration in seconds, the acceleration and the the steering acceleration intensity (value between 0.00 and 1.00), the max brake intensity (still as a bool value 0 or 1), the buffer size is an integer value. On the right side, there is the list of QWidgets for the sensitivity range, in order to help and to understand better the training done and its quality.

As for the Facial expression commands, for the Mental commands, thanks to "mentalCommandActionSensitivity" method, it is possible to set the sensitivity of activated actions. In the GUI, this is done with a slider on the right part in the Live Mode. The user can set dynamically a value in percentage (0-100%) and it appears only related to a trained command. If the slider is set to 0 the sensibility for that mental command / facial expression will be null, in other words, it will never be recognized by the BCI. Instead, if the slider is set to 100, the probability it is identified increase a lot.

The main change of this new version of Xavier is to shift in live mode the label that shows the current command sends and the sensibility slider. In this way, the user during streaming, looking the last command sent, can modify the sensibility to correct some wrong behaviour of the BCI. Another feature introduced is the dynamic setting of parameters to change the type of steering acceleration or braking.

In both windows, dynamic labels are shown in case of changes or if errors are encountered.



Last command

push

Maximum angle: 440
Max brake intensity: 1.0
Buffer Size: 5
Acc intensity: 0.2
Steering duration: 3.0
Brake duration: 1.0
Steer acc intensity: 0.17

Streaming

IP Address: 192.168.1.3
Port: 1233
No streaming

Sensitivity

Accelerate
smile 50 %

Brake
clench 50 %

Figure 4.12. Xavier v1.0 - Live mode window.

### 4.2.4 Improvements

During the test of the previous version, some problems had found and they have to be fixed. In particular, a part of many bugs that have been fixed, these problems are faced:

- Acceleration during the steering is needed.
- Packet frequency was not enough.
- Linear steering is not comfortable.
- Commands are often misinterpreted by the application.
- Hard to train mental commands.

In detail, the BCI observes one command at a time and in the first version of Xavier, only the command received by the headset was sent. So, if the BCI sent a command corresponding to left, only the turning left command was sent to the car and happened that it turned the wheel but without acceleration, then it does not move. The fix to this problem is pretty simple: any time a turn command (left or right) is sent, also the accelerate command is needed or at least a non-zero-value of the acceleration.

The second and the third point are related and they cause malfunction with the steering. What happened is that giving steering command the car reacted turning brusquely and doing big delta of steering for each command. For example, with the frequency of 5 Hz (frequency of the previous version), to go from 0 to 440 degrees of steering in 3 seconds, the steering wheel will turn fifteen times of almost 30 degrees. Such high delta of steering could brake the car steering system.

As a consequence of the real test on the track, it is considered appropriate to increase the sending frequency of the packets to 50 milliseconds (20 Hz) in order to decrease and minimise the turning delta and the delay of the communication to the car system. The solution for the first point will be explained in the Steering function section. The solution for fourth and the fifth was to learn how the training works find the best training patters and practise. All the sets have been explained in the previous section. Another improvement has been the addition of section where is possible to change some parameters for the steering and the acceleration.

### 4.2.5 Steering functions

Polynomial functions of the following kind can be considered:

$$q(t) = a_0 + a_1 t + a_2 t^2 + ... + a_n * t^n$$

The higher the degree n of the polynomial, the larger the number of boundary conditions that can be satisfied and the smoother the trajectory will be. A polynomial function is very often used in the trajectory planning for automatic machines and Robots.

In the initial version of the software, the values of the angle for the steering wheel followed a mathematical linear function. After testing, it has been noticed that the speed of the steering wheel returning back should be slower and gradual in order to avoid breakages in the system. It has been tested to double the previous value set but still the movement is not enough acceptable. Then, it has been tried an exponential function with the formula:

$$angle = multiplier * (base^{tick})$$

and it is implement setSteeringAngleExponential() method;

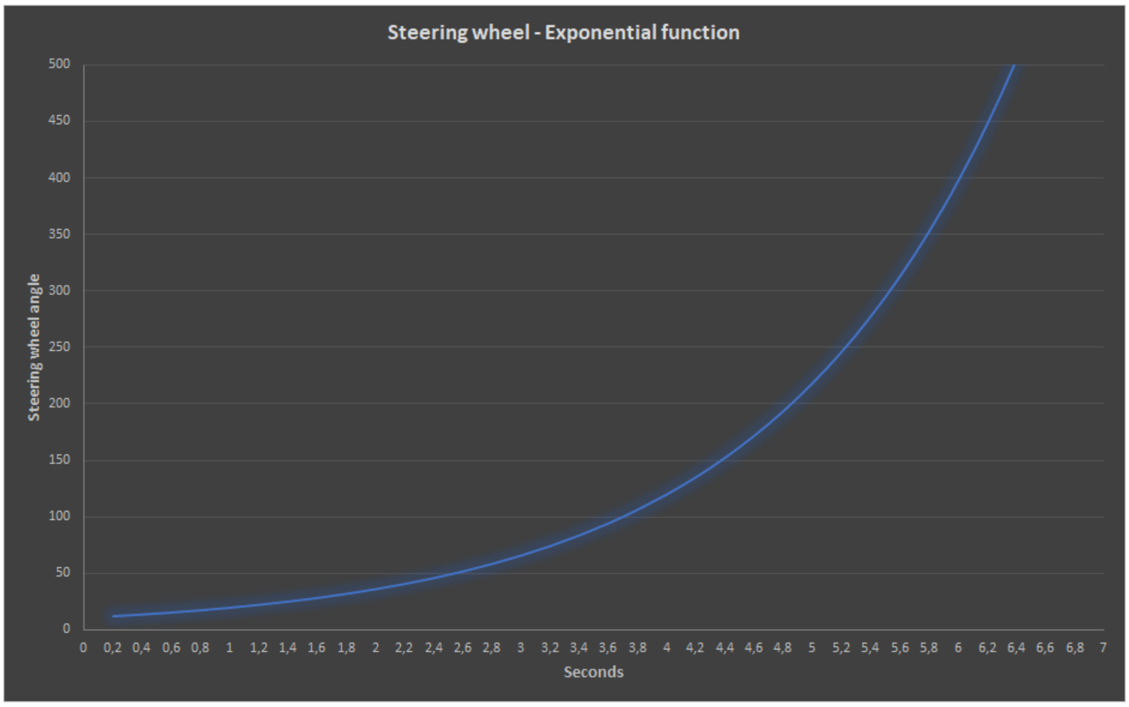


Figure 4.13. Exponential function for steering.

Moreover, there is also a trigonometric function with the formula

$$angle = (maxAngle/(pi/2)) * (Arcsin(tick/divisor))$$

implemented in the setSteeringAngleASin() method. Both solutions give a slight improvement with respect to the previous linear function but it is still not what is looked for.

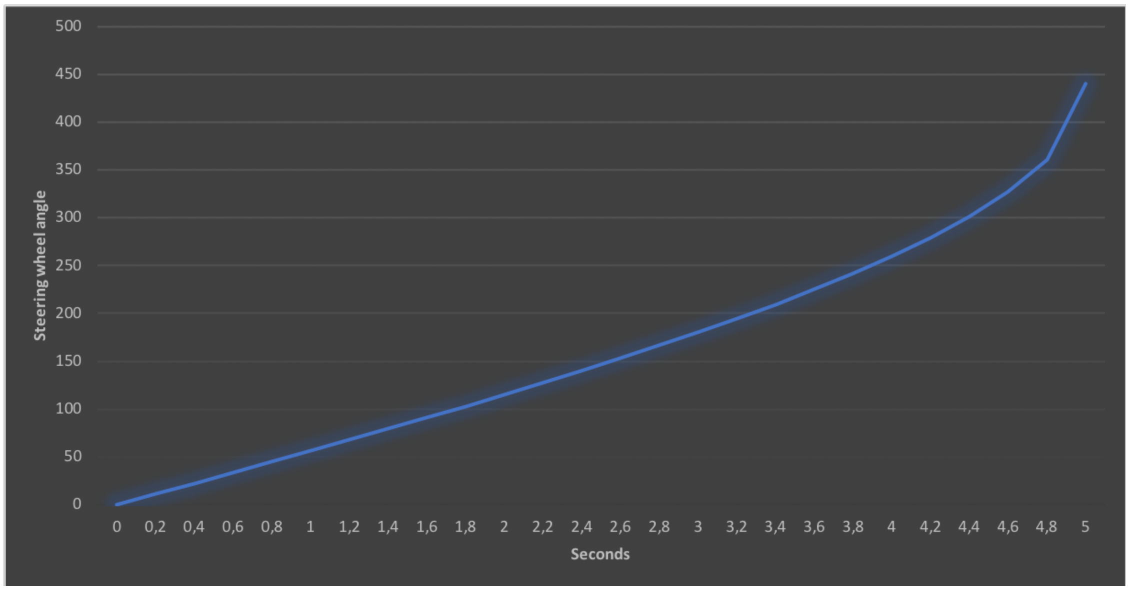


Figure 4.14. Arcosine function for steering.

In the end, it is introduced a cubic polynomial function able to make smoother the values of the angle sent to the car. This is implemented in the setSteeringAngleCubic() method of the Command class.

In the case considered, the following restrictions exist: $\alpha(t_i) = \alpha_i$; (the initial angle position) $\alpha(t_f) = \alpha_f$; (the final angle position) $t_i = 0$; (the initial time) $t_f$; (the final time):

These restrictions can be considered by using of the third order polynomial. The cubic polynomial is the third-order function. The basic form can be expressed as:

$$\alpha = f(t) = at^3 + bt^2 + ct + d \tag{4.1}$$

Considering the restrictions and equation (4.1), formulas for the calculation of coefficients a,b,c,d can be found:

$$\begin{cases} d = \alpha_i \\ c = 0, \\ at_f^3 + bt_f^2 = \alpha_f - \alpha_i \\ 3at_f^2 + 2bt_f = 0 \qquad (1) \end{cases}$$

(1) derivative equals to zero in order to consider a constant angle during fixed time.

Solving of the equation above gives the next values for coefficients a,b,c,d:

$$\begin{cases} d = \alpha_i \\ c = 0, \\ b = 3\left(\frac{\alpha_f - \alpha_i}{t_f^2}\right) \\ a = -2\left(\frac{\alpha_f - \alpha_i}{t_f^3}\right) \end{cases}$$

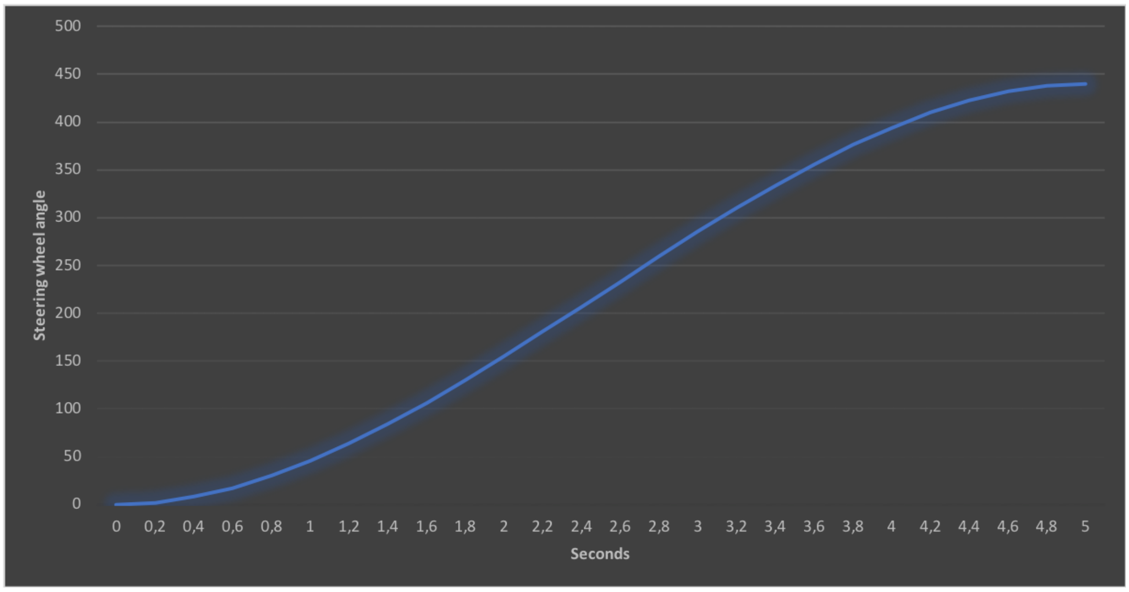


Figure 4.15. Cubic function for steering.

### Dynamic parameters

Adding a new function for steering introduced many variables to choose and to set, for example, the maximum angle of the steering or the time for the steering wheel to go from 0 degrees to the maximum value. This variable could be hard-coded but since the best value it's unknown, it is preferred to put a section in the GUI where is possible to set these parameters. Variable as acceleration and brake intensity are been added to give the possibility to modify the pressure value during the acceleration or the braking.

## 4.3 Code Structure

The architecture of Xavier in this new version did not change completely. Indeed, there are still two threads one that handles the GUI and the user interaction and one that handles the communication with the Headset and Emotiv Server through a SecureWebSocket and the communication with the car. The main change is the new GUI that it is not any more handle by a single class, MainWindow, but since the complexity increased, four additional classes are needed (Figure **??**). The Binder, the Command and the CortexClient classes are been modified to add the new features already described.

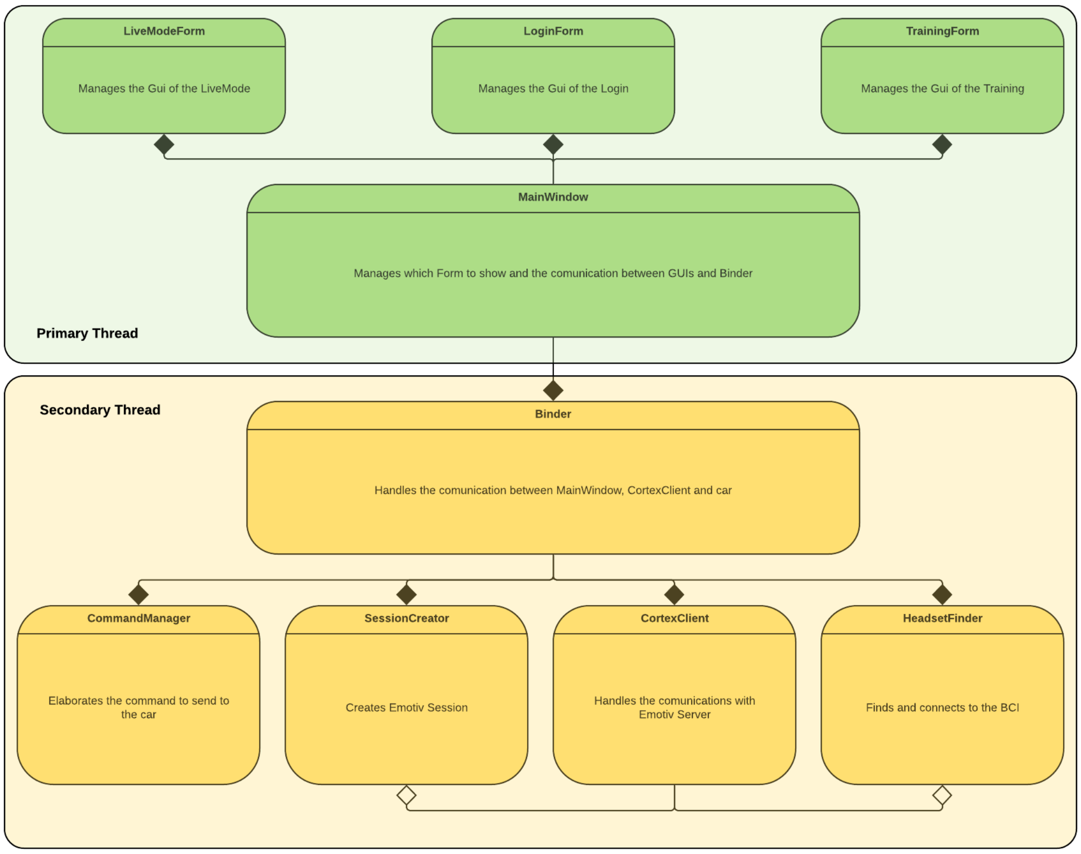

LiveModeForm
Manages the Gui of the LiveMode
LoginForm
Manages the Gui of the Login
TrainingForm
Manages the Gui of the Training
MainWindow
Manages which Form to show and the comunication between GUIs and Binder
Primary Thread
Secondary Thread
Binder
Handles the comunication between MainWindow, CortexClient and car
CommandManager
Elaborates the command to send to the car
SessionCreator
Creates Emotiv Session
CortexClient
Handles the comunications with Emotiv Server
HeadsetFinder
Finds and connects to the BCI

Figure 4.16. UML of Xavier v1.0.

### 4.3.1 GUI classes

The MainWindow class now contains a QStackedWidget that provides a stack of widgets where only one widget is visible at a time. This QStackedWidget contains three widgets LoginForm, TrainingForm and LivemodeForm.

The MainWindow so is the class that is responsible to show the right view (Widget) according to the program workflow and with the User interaction. It is also responsible for the communication between the Binder and the Widget that is shown. Later a couple of examples are presented.

LoginForm is a really simple widget that has two text edits and a button that manages the login. TrainingForm is the widget that shows the view for training (Figure 4.11) and it has a tabWidget that contains two Widget one for mental command and one for facial expression. The last Widget, LivemodeForm, manage the view of the live mode (Figure 4.12).

Let's make a couple of example of the communication between the GUI classes and the Binder: the user wants to train a new mental command, Xavier shows the last command that the BCI returned.

For the first example, the TrainingForm receives the information that the User wants to train a certain command. So it emits a signal called "trainHMI" that has two parameters: the command name and the typology of this command (mental command or facial expression). The MainWindow which receive this signal does nothing, but forwards the signal to the Binder. The latter will manage the communication with the Emotiv Server to train the desired command.

For the second example instead, the Binder receives from CortexClient the information about the command that the BCI thinks the user gives and sends a signal with this info to the MainWindow. This class calls a method of the LivemodeForm to set the last command that after all it is shown to the GUI.

### 4.3.2 Binder

The Binder class did not change its task or its architecture from the previous version. It is the core of Xavier and its job is to manage the interaction of the user and the communication with the Emotiv server and the car.

In this new version is added a slot to remove a training, added an automatic request to Emotiv for the rating for a new trained command, the profile is been hard coded so the user does not have to choose it every time. The main work has been modified with many existing slots to manage a request for both mental commands and facial expressions. For example, the slots "onTrain" wants the command to train and even the command type (mental or facial) because according to it, a different request to CortexClient has to be done.

Furthermore, it has been done a general improvement of stability and it has been developed better management of errors.

### 4.3.3 Command

The task of this class did not change from the previous version of Xavier: it has a setter where the Binder sets the last command read and then it has a function that creates the message to send to the car and formats it according to the protocol used.

The first change is that when the stream button is pressed all the parameters set by the user in the GUI are collected and sent to the command. With this parameter, this class calculate the type of function for steering (max steering value, time to get the maximum value from 0 degrees) and the length of a buffer. Indeed, the command has a circular buffer where puts the command passed by the Binder with a dimension set by the User. After the variable called "_lastCommand" is set with the command that has more occurrences in the circular buffer. In this way, it is applied a sort of filter and the command is considered if has been read for a certain amount of time (depends on the buffer length). Usually, a size of 6 elements is set, so a command to be considered has to be read for 0.5/1 second.

Another change is the type of function for the steering that is not a linear function any more but is a cubic function as already explained.

## 4.4 Test with simulation

The test with the simulation showed immediately a problem not related to Xavier but to the Simulation. Indeed, with the increment of the frequencies in sending packets to control the car, the system results in overload of work and its reaction are not in real-time. It accumulates the received packets that are elaborated after a certain amount of time. For this reason, all the graphical part has been reduced leaving only the relevant components of the scenario considered, lightening its memory usage while running.

It is been created, also, an easy bash script in order to launch automatically all commands to run the simulation.

It has been noticed the difference between simulation and real tests in the meaning of the value of the parameters chosen. For example, in the simulation tests, the speed intensity considered can be enough between 0.04 and 0.07; values that become almost senseless in the real tests where they have to be at least double. Besides, if there is a very small value of the acceleration it is set to the minimum value.

With the facial expressions, it seems to work everything fine. Of course, still are present difficulties to give some commands because sometimes the BCI misunderstand some expressions, but in general, the test is successful. The real problem happens when starting to control the car with the mental commands because it is very hard to train up to four different commands (neutral plus other three).

What happened is that sometimes a command can be given but just for a small amount of time (live half or one second). In this way, the car could not make a turn because the time it receives the command was too small. So, it is needed a new system that could drive the car with a small input. This system, receiving only a command to turn, could make the car do a whole curve of 90 degrees. Receiving an accelerate command, it could make, for example, the car accelerates for ten meters.

Thinking how to implement this system, called Assisted Drive, some questions about that remain still open:

1. how much should the car turns to make a curve of 90 degrees?
2. for how long should the car turns to make a curve of 90 degrees?
3. during the turning how much should the car accelerates to make a curve of 90 degrees?
4. how much should the car accelerates to make 10 meters with the car?

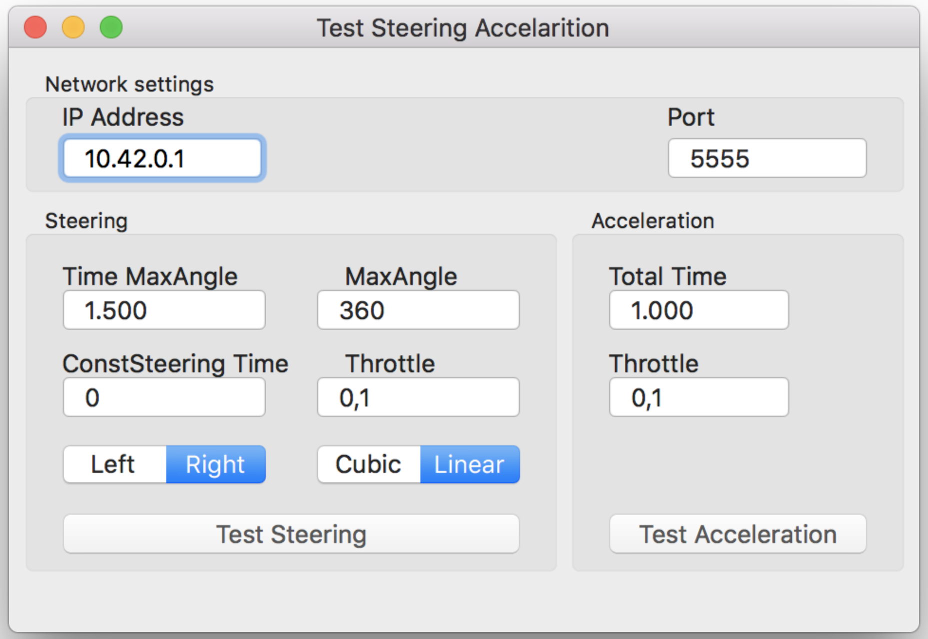
To answer these questions, it is developed a program that sends to the car a sequence of commands corresponding to a set of parameters that can be inserted. So the system can set for how much, how long the car should turn and with which acceleration value and the program sends these sequence of massages. The next section explains more in detail the program.

## 4.5 Assisted Drive Studies

The goal of these part was to find all the parameters to make a turn with the car and a straight line of a couple of meters. It has been developed a simple program with Swift eases the research. The GUI is pretty simple (Figure 4.17): a section for the network settings (address and port where the car is listening), a section with the parameters for the acceleration (pressure intensity and duration) and a section for steering. In this latter section there are six parameters:

1. the toward of the curve (left or right)
2. the type of the steering function (linear or cubic)
3. the acceleration intensity
4. the max angle

5. the time to go from 0 degrees of steering to the max value
6. the time the car remains with the steering wheel to the max value



IP Address: 10.42.0.1
Port: 5555
Time MaxAngle: 1.500
MaxAngle: 360
ConstSteering Time: 0
Throttle: 0,1
Total Time: 1.000
Throttle: 0,1

Figure 4.17. GUI of the test program developed in Swift.

## 4.6 Test on track

The test took place in the Club Des Miles, a circuit based in Moncalieri (TO). In this way, there is the possibility to try the software, controlling the car in a safe environment for testers and the other people. It would be very dangerous indeed to test the program driving the car in a normal street.

Initially, some tests with the Assisted Drive software are done to see which are the parameters to perform well a curve of 90 degrees. The result is that with a max angle of 440°, a pressure of the 15% on the gas pedal, a period of 2 seconds to go from 0 to 440 with a cubic function and 5 seconds with the wheel at the maximum value, the car is able to realise the desired turn. In the chart below (Figure 4.18), it is shown the steering wheel position in relation to the time.

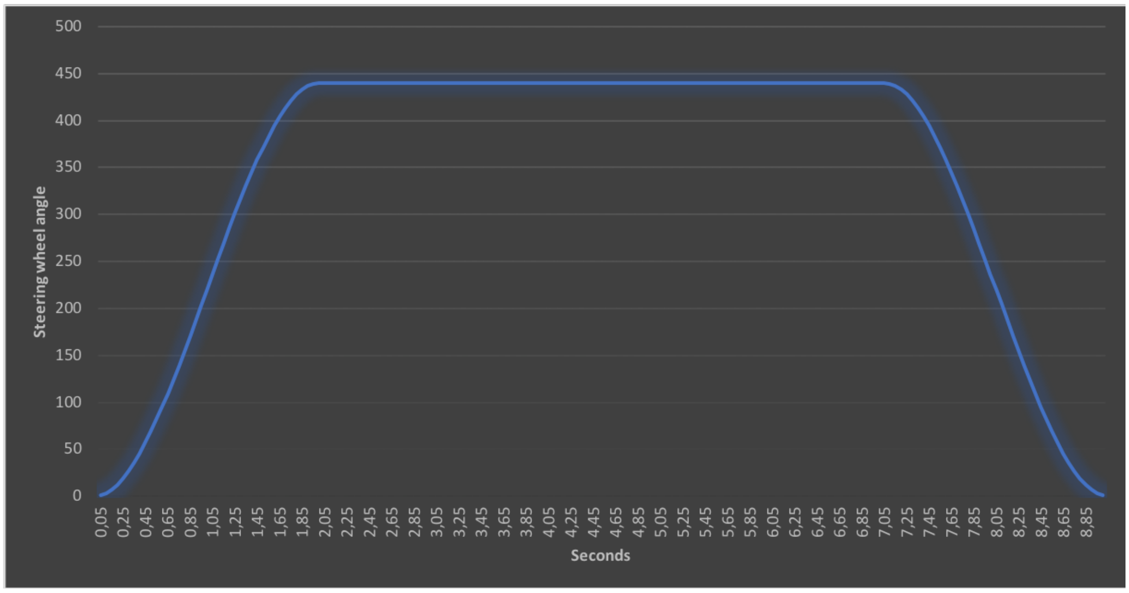


Figure 4.18. Steering wheel position during the Assisted Drive test considering 5 seconds of constant time.

The Xavier test phase starts and the environment of the test is structured in this way: on the driver side a person who, in case of necessity, would have taken the control of the car and, instead, in the passenger side a person who would have controlled the car with the BCI. The connectivity is similar to the previous test: a PC running Xavier and CortexUI connected with the BCI with the Bluetooth, with the car with an Ethernet cable and it had access to internet thanks to the connectivity provided by a mobile hotspot.

At first, the facial commands are tested and the results are overall successfully: the car was driven correctly in the circuit even if the speed is very limited (10-15 Km/h). Sometimes, the BCI did not recognize the command to turn and it was needed to wait or to resend the command, or manually brake to avoid to go outside the circuit.

After these first tests, the control of the car switches to the mental commands. As imagined, it has been really hard to train and have control of four commands. It results almost impossible to drive the car in the circuit.

### 4.6.1 Issues

During the test phase of this version some issues has been encountered. For example, it is needed to go deeper on the connection between the profile and the session when at the beginning the BCI connects to the

Emotiv server to handle the update of the session with all current data. Another aspect is to manage the time out of the headset when it disconnects automatically. It could be relevant to modify the value of the brake introducing also for it a more complex mathematical model.

It is important to still continue to modify the GUI and test if all the configuration can be set dynamically. Another important aspect is the connectivity because the internet connection is always needed in order to get access to the profile and to get the results performed by the Emotiv server and at the end to save all data into the selected profile.

# Chapter 5

# Xavier update and final tests

In this chapter, it is presented an improved version of the software. The first part of this chapter explains the updates and the new features to increase the code quality and the stability of the existing functionalities. The second part describes the results of the tests during the simulation phase and on track.

## 5.1 Xavier v1.1

In the initial version of the software, the values for the brake followed a true/false behaviour. In Xavier 1.0, the brake values are changed according to a mathematical linear function. After testing, it has been noticed that the speed of the braking was still glitchy and stiff. Then, as done for the values of the steering wheel, it is introduced a mathematical cubic function, with the same features explained in the previous chapter (equation 4.1), to get a smoother behaviour.

### 5.1.1 Profile and Session management

The tests, above all in open environments and on track, showed difficulties and implementation problems about connectivity and synchronization among the software, the headset and the Emotiv server.

The headset, very often, loses the connection and after a default time, the session is closed and never saved. It renders useless the training because it is lost. An unexpected closure of the application stores the profile but it closes the session and it doesn't save the training done into the profile.

A session represents a continuous period of data collection from a headset. To get data from a headset, start by creating a session, put it into the appropriate state, then subscribe to receive the results in real-time. The data available from the headset is split into various streams and to receive data from a session, a subscription is required.

It has been analysed the complete workflow of session and profile and the implementation done. It is faced is the exception error "(-32007) - Session does not exist" got in the previous cases and when the session is not more up.

The sessions and the profiles are created and saved on the local machine. After the session is set to "active" successfully (Figure 5.1), the data will be saved and uploaded on the Emotiv's server.

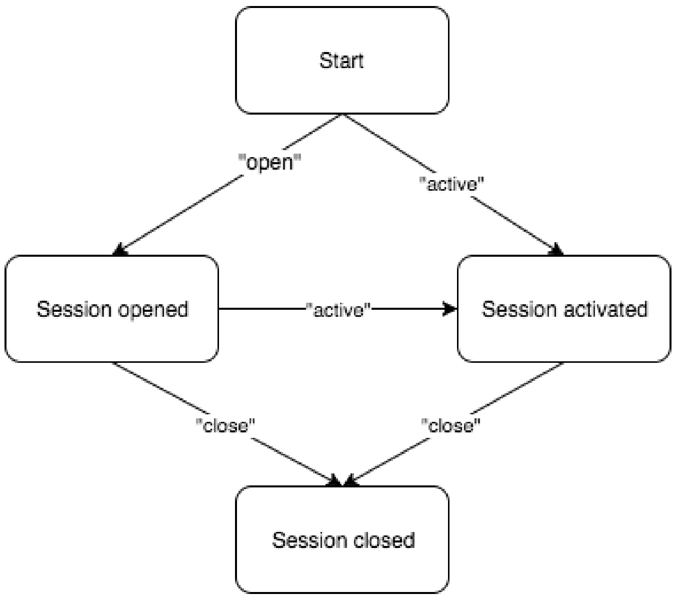


Figure 5.1. CortexAPI - Session workflow diagram.

It has been implemented a new workflow like the following: it is invoked the method queryProfile to get all training profiles of a specific user and check if the profile inserted already exists. If it is not found, create the profile and load an empty training data using the parameter "status" of the method setupProfile; create a session with status "active" and attach the profile to it. Otherwise, if the profile already exists on the server, it is enough to call the setupProfile to load training data from profile. It is very useful when the application crashes unexpectedly.

To be more consistent also the training and the live methods have been improved. In the TrainMode window, after every training and its rating, the application saves the profile updating all training data. In the LiveMode window, instead, if the session is closed, a popup window shows to wait in order to reopen the session or create a new one; then, it loads the associated profile.

Other troubles with the Profile management methods, solved in this new version, are: queryProfile: it returns an empty array while creating several profiles using EmotivBCI, setupProfile: trying to create a profile, it returns an error ”code”:-32031, ”message”: ”Invalid profile name.”.

The empty array is given because of the authorization with Cortex as an anonymous user. So, queryProfile returns empty. The error ”Invalid profile name.” usually means that the application tries to create a profile ”test” when another profile with the same name already exists. The solution to these issues is to add working client_id and client_secret to params when calling the method ”authorize” even without adding the license.

### 5.1.2 Assisted Drive

After the previous testing, some difficulties to turn the car smoothly in the curve has been noticed and the new idea of a different driving mode called Assisted Drive starts to be implemented. In the previous chapter, it has been explained that a swift application is implemented to test this the new concept. In this version, Xavier is upgraded adding this new type of driving.

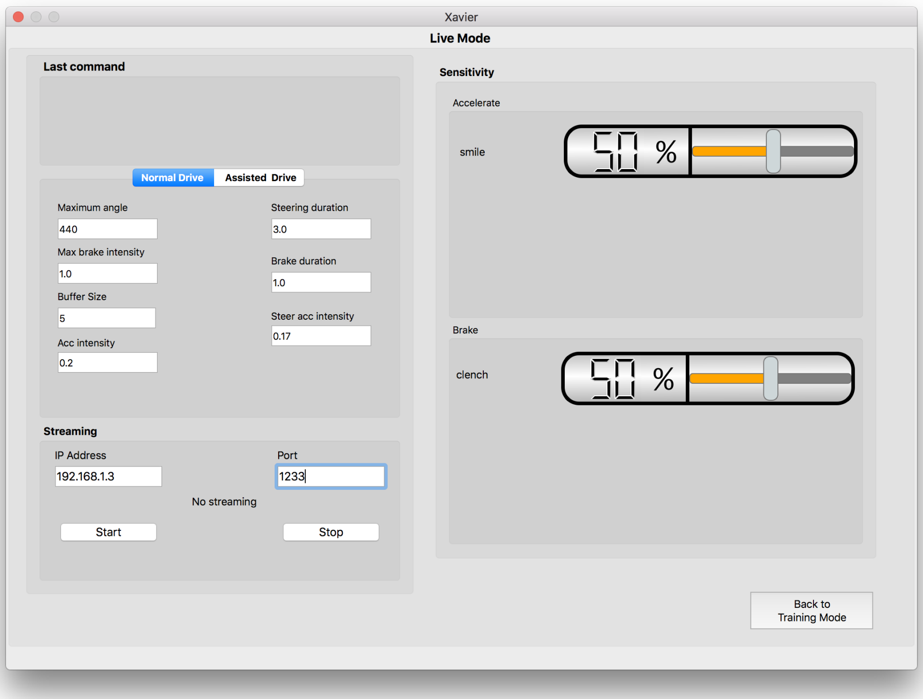
The Assisted Drive mode borns based on a circular buffer to collect a certain amount of signals and select the one according to the maximum number of occurrences. It has been improved setting dynamically the fixed size of the buffer and selecting the maximum with the minimum value got from the field “Min number elements buffer”. The buffer has an upper bound to not create too much delay during the analysis of the signals and computation of the algorithm. If some commands are even, the first frequent one is chosen and not any more the brake command by default. Again, this upper bound value gives better performances between five and ten elements. This mode introduces, also, a constant value expressed in seconds to keep constant the angle of the steering. It is designed, above all, for curves of 90 degrees. Changing this value together with the other parameters like the brake duration or the steering acceleration intensity is possible to get better results and to shape and to adapt for every context.

### 5.1.3 GUI

The graphic interface doesn't receive big improvements but it includes the new features implemented. The changes refer, above all, to the LiveMode window. The LiveModeForm includes a tab widget to select the driving mode between "Normal Drive" or "Assisted Drive" and to set up the parameters useful for the configuration.

Normal Drive (Figure 5.2) introduces the following fields:

- Max brake intensity to set the maximum value of the pressure on the brake pedal.
- Brake duration to configure the time to reach the maximum brake intensity.
- Steering acceleration intensity to diversify the speed on the straight away and the speed performing a curve. It is considered as a smaller value than the straight one.

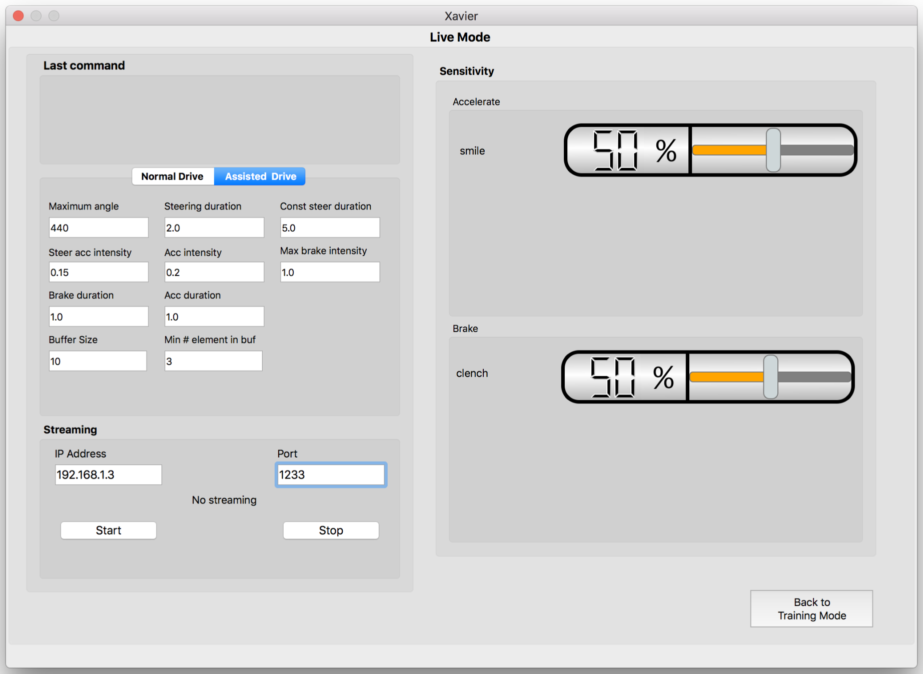


Maximum angle: 440
Max brake intensity: 1.0
Buffer Size: 5
Acc intensity: 0.2
Steering duration: 3.0
Brake duration: 1.0
Steer acc intensity: 0.17
IP Address: 192.168.1.3
Port: 1233

Figure 5.2. Xavier v1.1 - Normal Drive in LiveMode window.

Assisted Drive (Figure 5.3) adds the following fields:

- Constant steering duration, it's the time in seconds to keep constant the angle of steering.
- Acceleration duration, it's the time in seconds to keep constant the value of the speed.
- Min number elements buffer, it's the minimum number of occurrences wanted in the buffer to select the desired command.



Last command

| | | |
|---|---|---|
| Maximum angle | Steering duration | Const steer duration |
| 440 | 2.0 | 5.0 |
| Steer acc intensity | Acc intensity | Max brake intensity |
| 0.15 | 0.2 | 1.0 |
| Brake duration | Acc duration | |
| 1.0 | 1.0 | |
| Buffer Size | Min # element in buf | |
| 10 | 3 | |

Streaming

IP Address: 192.168.1.3
Port: 1233
No streaming

Sensitivity

Accelerate
smile 50 %

Brake
clench 50 %

Figure 5.3. Xavier v1.1 - Assisted Drive in LiveMode window.

### 5.1.4 Improvements

The main problems about internet connectivity have been solved introducing the management of the session in order to reload all current profile in case of bluetooth disconnections of the headset or internet disconnection.

Furthermore, several exceptions errors have been handled case by case to allow correct flow of the application and to improve the stability of the software reacting in the right way.

### 5.1.5 Test on track

The full test took place in the track described in the previous section. After the setup of the car, the headset is connected and the application is launched. The BCI driver sits on the passenger side and a “safety” driver stays on the driver side. The setup configuration includes also cameras to record the driving from inside and from an external point of view. The speed of the car remains limited to 10-15 Km/h.

The first step is to test fully the facial expressions. Even if the Assisted Drive, originally, has been thought to correct and to help the intensity given by the mental commands above all during the curves, it was useful to test also for the facial expressions to understand fully the accurate parameters to assign. Then, three laps are done in Normal Drive and one with the Assisted Drive. Above all the last two laps allow driving correctly the car only with small defections. So these tests give satisfactory outcomes and they are completely passed.

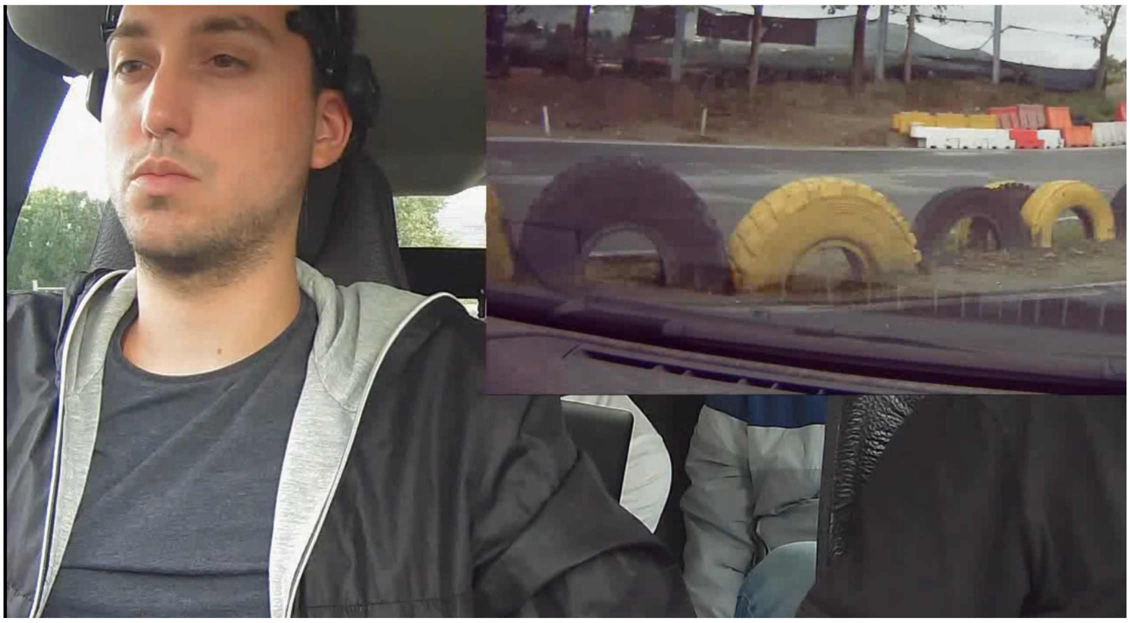


Figure 5.4. Test - Internal view of the car during driving with Mental Commands.

More complex is the second step about testing the mental commands. Initially, only push and brake command are trained and tested with good results. Step by step, before the left and later the right commands, are added to the test. In this case, due to strong difficulties in the commands recognition, only two laps are executed.

The results, unluckily, are not satisfying like for the facial expressions.

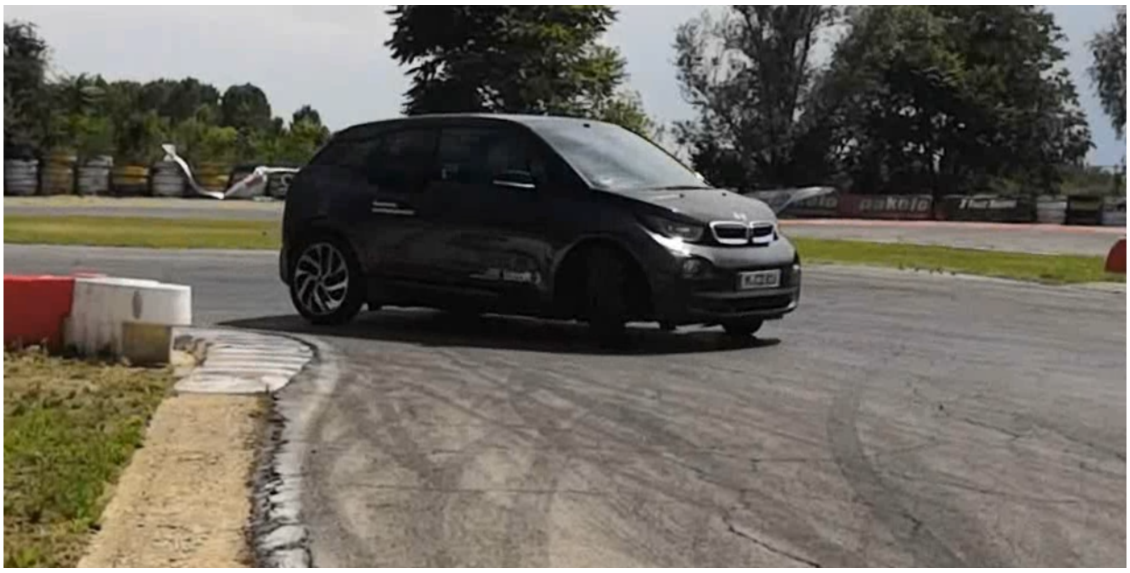


Figure 5.5. Test - The car correctly curves with Facial Expressions.

Here, clearly, the training plays a more relevant role and it shows big differences during the tests in live mode. It is not possible to complete autonomously a full lap but some manual corrections are introduced by the safety driver.

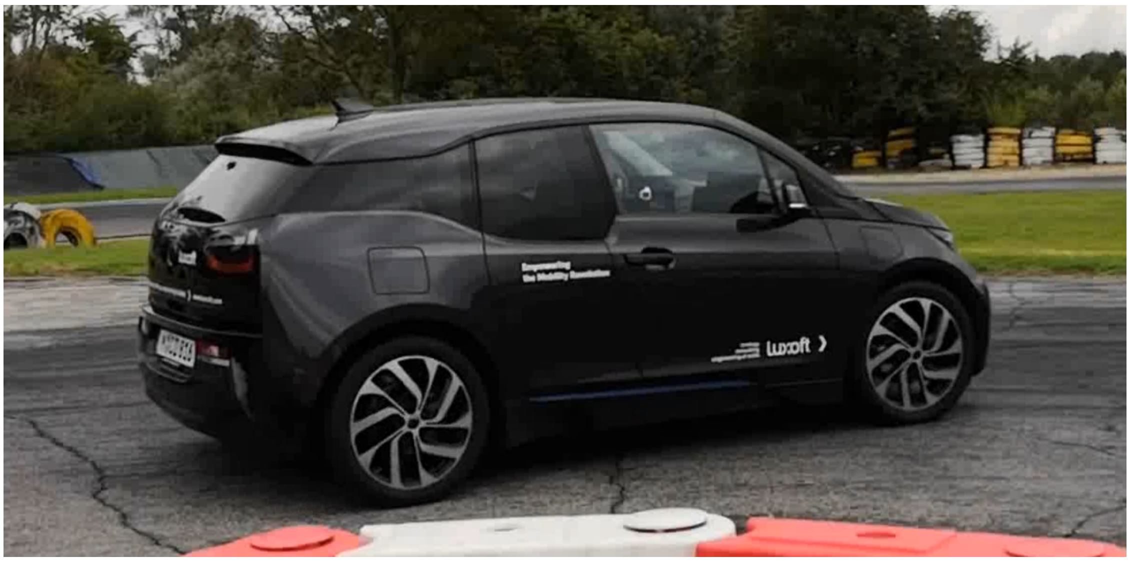


Figure 5.6. Test - The car shows difficulties to curve with Mental Commands.

The optimal parameters in Normal Drive mode are the max angle of 440°, the pressure of the 22% on the gas pedal on the straightaway and of the 17% on the curves, the pressure of 100% on the brake pedal with intensity computed by the mathematical cubic model. To reach the maximum angle, the period chosen is of 3 seconds with a cubic function.

Besides, the optimal parameters in Assisted Drive mode are: 5 seconds for the constant steering duration, a buffer size of 10 elements and the minimum number of the most occurrences is set to 3. The thresholds of sensitivity are set to 70% for the acceleration command, 50% for the brake, 60% for the turning commands. Theoretically, this mode should improve the smoothness of the curve adapting to any type of it. Unluckily, it didn't bring the expected results because, in order to to have a real improvements, the parameters should be changed and adjusted according to every kind of curve. Thought mainly for 90 degrees curve, it has good performances but when the curve is smaller, the steering wheels too fast.

The headset works all day (at least 7-8 hours) and it shows good features about battery performances. The stability still is not fully reliable but the changes, introduced in the implementation of the code, helped to get better results.

Driving with four mental commands and have full control of the car remains still very hard and for sure not possible in normal driving situations.

### 5.1.6 Statistical results

The bar of Skill Rating (SR), present in the SDK, is used as indication to evaluate all the stimuli applied when trying to improve the correspondence between the intention of movement produced by the brain and the action executed by the device. The SR bar provides a measure of how consistently the user can mentally perform the intended action [25]. The most relevant aspect is the percentage of errors done during the drive. An error is defined as every time the safety driver had to press the brake to avoid leaving the track. To generate a point of comparison between the different stimuli, relevant information about the results of the user training is presented next.

#### Facial Expressions

Tables 5.1-5.4 show some statistics about the different indicators in each trained action during the training phase. In the Table 5.5, are shown the mistakes made during the live mode while driving the car on the track using the trained facial expressions and the time, in minutes, to complete a full lap. The results, with facial expressions, are very performant. It's possible to notice small difficulties in the training of the fourth action and in the recognition of it, sometimes misunderstood with push or turn left command. The attempt to test the Assisted Drive, already for the facial expression, don't give the expected results because this mode, thought for 90 degree curves, is too much dependent on parameters set.

Table 5.1. Training of first action: Accelerate

| **Action1 - Accelerate** | |
|---|---|
| Min. Trials to reach 75% SR | 1 |
| Max. Trials to reach 75% SR | 4 |
| Avg. Trials to reach 75% SR | 2 |
| Max. SR reached | 100 |
| Trials to reach Max. SR | 2 |
| Avg. Max. SR | 96 |
| Avg. Trials to reach Max. SR | 3 |
| Max. Trials to reach Max. SR | 6 |

Table 5.2. Training of second action: Brake

| **Action2 - Brake** | |
|---|---|
| Min. Trials to reach 75% SR | 1 |
| Max. Trials to reach 75% SR | 6 |
| Avg. Trials to reach 75% SR | 3 |
| Max. SR reached | 100 |
| Trials to reach Max. SR | 3 |
| Avg. Max. SR | 94 |
| Avg. Trials to reach Max. SR | 3 |
| Max. Trials to reach Max. SR | 6 |

Table 5.3. Training of third action: Turn left

| **Action3 - Turn left** | |
|---|---|
| Min. Trials to reach 75% SR | 1 |
| Max. Trials to reach 75% SR | 6 |
| Avg. Trials to reach 75% SR | 3 |
| Max. SR reached | 100 |
| Trials to reach Max. SR | 3 |
| Avg. Max. SR | 94 |
| Avg. Trials to reach Max. SR | 3 |
| Max. Trials to reach Max. SR | 8 |

Table 5.4. Training of fourth action: Turn right

| **Action4 - Turn right** | |
|---|---|
| Min. Trials to reach 75% SR | 2 |
| Max. Trials to reach 75% SR | 8 |
| Avg. Trials to reach 75% SR | 4 |
| Max. SR reached | 100 |
| Trials to reach Max. SR | 5 |
| Avg. Max. SR | 90 |
| Avg. Trials to reach Max. SR | 5 |
| Max. Trials to reach Max. SR | 10 |

Table 5.5. Facial Expression: tests executed on track.

| | **Time** | **#errors** |
|---|---|---|
| Lap 1 - N.D. | 33 min | 16 |
| Lap 2 - N.D. | 25 min | 9 |
| Lap 3 - N.D. | 20 min | 3 |
| Lap 4 - A.D. | 27 min | 12 |
| N.D.: Normal Drive, A.D.:Assisted Drive | | |

## Mental Commands

Tables 5.6-5.9 show some statistics about the different indicators in each trained action during the training phase. In the Table 5.10, are shown the mistakes made during the live mode while driving the car on the track using the trained mental commands and the time, in minutes, to complete a full lap. The results with mental commands, despite the facial expressions, are not performing so well. It's possible to notice difficulties already during the training of the third and fourth action and in the recognition of them, too often not recognised or not reaching the satisfactory threshold. The Assisted Drive is not tested due to all the complications had in Normal Drive mode.

Table 5.6. Training of first action: Brake

| **Action1 - Brake** | |
|---|---|
| Min. Trials to reach 75% SR | 1 |
| Max. Trials to reach 75% SR | 8 |
| Avg. Trials to reach 75% SR | 3 |
| Max. SR reached | 100 |
| Trials to reach Max. SR | 3 |
| Avg. Max. SR | 94 |
| Avg. Trials to reach Max. SR | 4 |
| Max. Trials to reach Max. SR | 10 |

Table 5.7. Training of second action: Accelerate

| **Action2 - Accelerate** | |
|---|---|
| Min. Trials to reach 75% SR | 3 |
| Max. Trials to reach 75% SR | 10 |
| Avg. Trials to reach 75% SR | 5 |
| Max. SR reached | 100 |
| Trials to reach Max. SR | 10 |
| Avg. Max. SR | 90 |
| Avg. Trials to reach Max. SR | 8 |
| Max. Trials to reach Max. SR | 15 |

Table 5.8. Training of third action: Turn left

| **Action3 - Turn left** | |
|---|---|
| Min. Trials to reach 75% SR | 8 |
| Max. Trials to reach 75% SR | 25 |
| Avg. Trials to reach 75% SR | 12 |
| Max. SR reached | 80 |
| Trials to reach Max. SR | 10 |
| Avg. Max. SR | 77 |
| Avg. Trials to reach Max. SR | 14 |
| Max. Trials to reach Max. SR | 35 |

Table 5.9. Training of fourth action: Turn right

| **Action4 - Turn right** | |
|---|---|
| Min. Trials to reach 75% SR | N.R. |
| Max. Trials to reach 75% SR | N.R. |
| Avg. Trials to reach 75% SR | N.R. |
| Max. SR reached | 64 |
| Trials to reach Max. SR | 14 |
| Avg. Max. SR | 60 |
| Avg. Trials to reach Max. SR | 18 |
| Max. Trials to reach Max. SR | 50 |
| N.R.: not reached | |

Table 5.10. Mental Commands: tests executed on track.

| | **Time** | **#errors** |
|---|---|---|
| Lap 1 - N.D. | 58 min | 37 |
| Lap 2 - N.D. | incomplete | - |
| N.D.: Normal Drive | | |

# Chapter 6

# Conclusions

From the beginning of this project, it was clear that the main result to achieve would have been very complex and it could have required big effort. The results indicate that the EEG data obtained with the Emotiv EPOC device contain sufficient information to distinguish different states and that machine learning techniques are able to learn the patterns that distinguish these states.

Some of the results achieved in this thesis project:

- It is clear how a simulation software is essential in the development of a remote/brain-controlled driving process, thanks to the possibility to perform and validate the softwares, the algorithms and the new logics under different conditions and scenarios without wasting of time and additional resources.
- The development of the application able to connect remotely to the headset and to send signals, converted to driving commands, to the car.
- The driving tests with facial expressions brought to very good results and on track several laps were fully completed. Instead, with mental command, tests result still very hard and it is not possible to complete autonomously a good real drive.
- The training phase for the mental commands remains very important and an essential step to have clean data to be recognised during driving.

In conclusion, the main advantages of Emotiv Epoc+ are the low cost, compact, convenient and suitable for studies that do not require high sampling frequency and the performance is above random and not due to muscular or ocular artefacts. Robustness is a major concern about the Emotiv headset. The hardware is basically made of plastics and low-cost components, which results in a fragile headset. The plastic-based screw thread can easily break up if a particular attention is not brought during each experiment. In case makeshift repairs are not possible, a new headset has to be bought. Moreover, the electrode metallic parts are quickly oxidized even if cleaned at the end of each experiment as shown in Figure 6.1. After a while, they appear to produce less good signals (during this experiment, all the Emotiv headset electrodes were non oxidized). Moreover, the moss part of all the electrodes is degrading with time and has to be considered as a consumable.

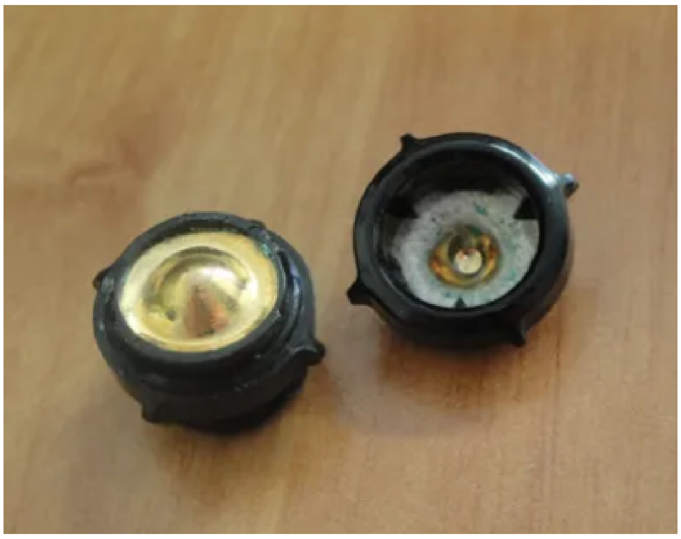


Figure 6.1. Oxidation of the electrodes. The oxidation of the electrode is clearly visible in green. On the back, the non-oxidized electrode has a gold color.

All of that concludes only the first step of this ambitious and challeging project started few months ago. Several aspects has to be taken into consideration.

While a delayed response rate between the thinking and command action currently seems to be the biggest flaw, there are other major issues of concern such as: what about the drivers? Every group of people can be factored into the system? What are they suppose to say if an accident were to occur due to road rage for example? “I’m sorry officer; I didn’t really mean to ram my front end of the car into the back of that vehicle. I only ‘thought’ about accelerating.” On the other hand, while surely there are still flaws that need to work on, this brain driven car can do wonders for others—especially disabled people who many no longer have the use of limbs and legs.

The control of real world apparatus with your thoughts is a talk of fantasy and has been a science-fiction author's favourite theme for decades. But with the recent advancements in Brain–Computer Interfaces, such technology is no longer absurd and their development is coming to a prime time for applications. While, maybe, the concept of brain-driven cars is currently under development, this work can be considered just proof of a concept. The task here was to show free driving by detecting brain patterns. There is still a long way to go until it could be possible to take full control of the machines with human brains.

## 6.1 Future steps

Hopefully, the results obtained can be a starting point for the next developers. About the software application developed, it can be improved the stability, beautified the user interface making lighter and aligned to new features. It could be laso convenient to integrate the ROS simulation as part of the application.

As future projects in emotion research, it is interesting to systematically consider the different feature extraction methods and learning methods to improve the accuracy of classifiers. It could be worth to investigate over the performance metrics (interest / affinity, excitement, frustration, engagement, relaxation, focus, stress, long term excitement) and capturing these metrics in real life situations and other professional fields to control or directing some actions (Figure 6.2).

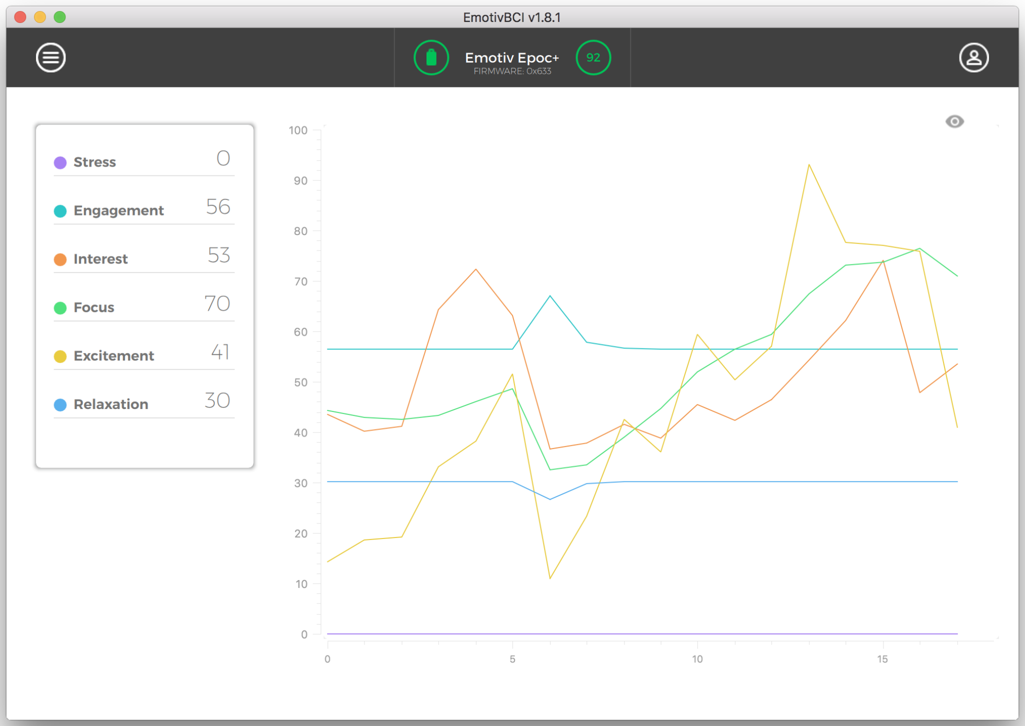


Figure 6.2. Performance metrics graphs includes 6 metrics for Insight and EPOC+ headsets.

Regarding the evaluation of the general quality of the Emotiv EPOC device, three main axes can be explored: a larger number of subjects, other BCI paradigms and the design of a new low-cost EEG headset. First, some results are not strong enough to resolve the issue related to Emotiv performance. Therefore, a much larger number of subjects could be used to obtain even clearer results.

Secondly, given the need for greater reliability in critical applications such as rehabilitation, orthosis / prosthesis control or driving a motor vehicle, the design of a new low-cost EEG headset is required. Ideally, this headset should be light, have great battery runtimes, and have performance that is as close as possible to a medical system and should be relatively inexpensive.

EEG is a complex signal and can require several years of training, as well as advanced signal processing and feature extraction methodologies to be correctly interpreted. Deep learning has shown great promise in helping make sense of EEG signals due to its capacity to learn good feature representations from raw data but this concept is still open.

# Bibliography

[1] Picard,R.W.,Klein,J., “Toward computers that recognize and respond to user emotion: Theoretical and practical implications”, MIT Media Lab Tech Report 538, to appear in Interacting with Computers 14(2),141–169 (2002),

[2] J. del R. Mill´an, R. Rupp, G. Mueller-Putz, R. Murray-Smith, C. Giugliemma, M. Tangermann, C. Vidaurre, F. Cincotti, A. Kubler, R. Leeb, C. Neuper, K. R. Mueller, and D. Mattia, “Combining braincomputer interfaces and assistive technologies: Stateof-the-art and challenges”, Frontiers in Neuroscience, vol. 4, no. 0, p. 12, 2010.

[3] G. Pfurtscheller, G. R. Muller, J. Pfurtscheller, H. J. Gerner, and R. Rupp, “Thought’ - control of functional electrical stimulation to restore hand grasp in a patient with tetraplegia”, Neuroscience Letters, vol. 351, no. 1, pp. 33 – 36, 2003.

[4] B. Rebsamen, C. Guan, H. Zhang, C. Wang, C. Teo, M. Ang, and E. Burdet, “A brain controlled wheelchair to navigate in familiar environments”, Neural Systems and Rehabilitation Engineering, IEEE Transactions on, vol. 18, no. 6, pp. 590–598, 2010.

[5] A. Nijholt, D. P.-O. Bos, and B. Reuderink, “Turning shortcomings into challenges: Brain-computer interfaces for games” Entertainment Computing, vol. 1, no. 2, pp. 85 – 94, 2009.

[6] Partala, T., Jokiniemi, M., Surakka, V., “Pupillary responses to emotionally provocative stimuli”, In: ETRA 2000: Proceedings of the 2000 Symposium on Eye Tracking Research and Applications, pp. 123–129. ACM Press, New York (2000).

[7] Picard,R.W.,Klein,J., “Toward computers that recognize and respond to user emotion: Theoretical and practical implications”, Interacting with Computers 14(2), 141–169 (2002).

[8] Takahashi, K., “Remarks on emotion recognition from biopotential signals.”, In: 2nd International Conference on Autonomous Robots and Agents, pp. 186–191 (2004).

[9] Chanel, G., Kronegg, J., Grandjean, D., Pun, T., “Emotion Assessment: Arousal Evaluation Using EEG’s and Peripheral Physiological Signals”. In: Gunsel, B., Jain,A.K., Tekalp, A.M., Sankur, B. (eds.) MRCS 2006. LNCS, vol. 4105, pp. 530–537. Springer, Heidelberg (2006).

[10] Choppin, A., “Eeg-based human interface for disabled individuals: Emotion expression with neural networks”. Masters thesis, Tokyo Institute of Technology, Yokohama, Japan (2000).

[11] J. del R. Mill´an, R. Rupp, G. Mueller-Putz, R. Murray-Smith, C. Giugliemma, M. Tangermann, C. Vidaurre, F. Cincotti, A. Kubler, R. Leeb, C. Neuper, K. R. Mueller, and D. Mattia, “Combining braincomputer interfaces and assistive technologies: Stateof-the-art and challenges” Frontiers in Neuroscience, vol. 4, no. 0, p. 12, 2010.

[12] R. Ortner, B. Z. Allison, G. Korisek, H. Gaggl, and G. Pfurtscheller, “An SSVEP BCI to control a hand orthosis for persons with tetraplegia” IEEE Trans Neural Syst Rehabil Eng, vol. 19, no. 1, pp. 1–5, 2011.

[13] M. Duvinage, T. Castermans, R. Jimnez-Fabian, T. Hoellinger, C. De Saedeleer, M. Petieau, K. Seetharaman, T. Dutoit, and G. Cheron, “A fivestate P300-based foot lifter orthosis: Proof of concept” in 5rd ISSNIP Biosignals and Biorobotics IEEE Conference, 2012.

[14] K. Stamps and Y. Hamam, “Towards inexpensive BCI control for wheelchair navigation in the enabled environment - a hardware survey” in Proceedings of the 2010 international conference on Brain informatics, ser. BI’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 336–345.

[15] A. Campbell, T. Choudhury, S. Hu, H. Lu, M. K. ukerjee, M. Rabbi, and R. D. Raizada, “Neurophone: brain-mobile phone interface using a wireless EEG headset” in Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds, MobiHeld ’10. New York, NY, USA: ACM, 2010, pp. 3–8.

[16] P. Bobrov, A. Frolov, C. Cantor, I. Fedulova, M. Bakhnyan, and A. Zhavoronkov, “Brain-computer interface based on generation of visual images” PLoS ONE, vol. 6, no. 6, p. e20674, 06 2011.

[17] K. Stytsenko, E. Jablonskis, and C. Prahm, “Evaluation of consumer EEG device emotiv epoc”, in MEi:CogSci Conference, 2011, IEEE review, “IEEE Spectrum.” IEEE Press Piscataway, NJ, USA, January 2009.

[18] 26. M. Duvinage, T. Castermans, R. Jimnez-Fabian, T. Hoellinger, C. De Saedeleer, M. Petieau, K. Seetharaman, T. Dutoit, and G. Cheron, “A fivestate P300-based foot lifter orthosis: Proof of concept” in 5rd ISSNIP Biosignals and Biorobotics IEEE Conference, 2012.

[19] Lin, Y.-P., Wang, C.-H., Jung, T.-P., Wu, T.-L., Jeng, S.-K., Duann, J.-R., Chen, J.-H., “EEG-Based Emotion Recognition in Music Listening”. IEEE Transactions on Biomedical Engineering 57(7) (2010).

[20] Kandel, E.R., Schwartz, J.H., Jessell, T.M., “Principles of Neural Science”, Mc Graw Hill (2000).

[21] Ramirez, Rafael and Vamvakousis, Zacharias. (2012), “Detecting Emotion from EEG Signals Using the Emotive Epoc Device”. 7670. 175-184. 10.1007/978-3-642-35139-6_17.

[22] https://www.eteknix.com/man-drives-f1-car-with-the-power-of-his-mind/

[23] Zhang, S., Yuan, S., Huang, L. et al. Human Mind Control of Rat Cyborg’s Continuous Locomotion with Wireless Brain-to-Brain Interface. Sci Rep 9, 1321 (2019). https://doi.org/10.1038/s41598-018-36885-0

[24] https://www.emotiv.com/news/wired-emotivs-mind-powered-road-safety-system-slows-the-car-when-a-drivers-distracted/

[25] Source: Emotiv. Emotiv Software Development Kit User Manual for Release 1.0.0.5.

[26] http://www.eecs.ucf.edu/seniordesign/fa2012sp2013/g21/FinalPaperII.pdf

[27] H.S. Anupama, N.K. Cauvery, G.M. Lingaraju, International Journal of Advances in Engineering & Technology, 3, 739-745 (2012)

[28] B. He, Neural Engineering Second Edition, (New York, NY: Springer Berlin Heidelberg, 2013), ISBN 9781489978875

[29] https://emotiv.github.io/cortex-docs

[30] https://www.emotiv.com/epoc/

[31] https://emotiv.gitbook.io/emotivbci/mental-commands/tips-and-tricks

[32] https://www.upwork.com/hiring/for-clients/qt-cross-platform-app-development/

[33] https://www.qt.io/built-with-qt/

[34] https://doc.qt.io/qt-5/signalsandslots.html

[35] https://doc-snapshots.qt.io/qt5-5.12/why-moc.html

[36] http://wiki.ros.org/ROS/Concepts

[37] http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1

[38] https://github.com/osrf/car_demo

[39] Esther F. Kutter, Jan Bostroem, Christian E. Elger, Florian Mormann, Andreas Nieder. Single Neurons in the Human Brain Encode Numbers. Neuron, 2018; DOI: 10.1016/j.neuron.2018.08.036

# Acknowledgements

We would like to thank firstly Marco Bottero, who, as representative of Luxoft, gave us the possibility and the necessary financial support to carry out this project.

We wish to express our sincere gratitude to our internship tutor Vincenzo Campanale, who shared with us his immense knowledge and passion about brain computer interfaces and guided us with patience in the development of our thesis. Without his help and motivation this project would not have been possible.

We would like to thank also our thesis advisor Professor Massimo Violante of the Polytechnic of Turin for this great opportunity, his advices and useful information.

Beside our advisors, our sincere thanks go to our colleagues Graham Hili and Sebastiano Barrera for the continuous support with the teleoperated car. Only with their help and knowledge the tests with the real car were possible.