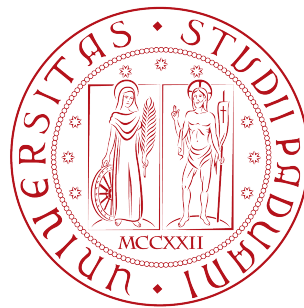


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

TESI DI LAUREA MAGISTRALE IN BIOINGEGNERIA



FROM BRAIN QUICK TO MATLAB AND EEGLAB:
DEVELOPMENT OF AN EEG DATA CONVERSION PLUGIN FOR
MICROMED DEVICES



Relatore
Dott. Ing. Marco Castellaro
Correlatore
Ing. Giulia Pellegrino

Candidata
Mariagrazia Ambrosino
Matricola 1217571

A.A. 2021/2022

*Ai miei genitori per l'amore incondizionato
con cui hanno sostenuto le mie scelte*

Contents

Abstrac	1
1 Electroencephalography (EEG)	2
1.1 A theoretical overview	2
1.2 Detection system and Electrode placement	3
1.3 Derivations	7
1.4 Electrode Mountings	10
1.5 Main waves	12
1.6 Clinical Applications	15
2 Case Study: Brain Quick Software	16
2.1 Main Software Features	16
2.2 Software User Interface	18
2.3 Events	20
3 Product Development Protocol (PDP)	21
3.1 General Informations	21
3.2 Stage 1: New Idea and Feasibility	22
3.3 Stage 2: Design	22
3.4 Stage 3: Development, verification and validation	25
3.5 Stage 4: Market Release and Post-marketing surveillance	25
4 Matlab Plugin Development	26
4.1 C++ Coding Environment	26
4.2 Matlab code and use of EEGlab tool	42
5 Testing Stage	53
5.1 DLL and User Guide	53
5.2 Test report with definition of acquisition conditions	57
5.3 Micromed equipment for the acquisition of EEG traces	60
5.4 Test Performance and Results	65
6 Validation and Future Developments	72
6.1 Conclusion	73

Abstract

This thesis work has been developed during an internship experience at Micromed S.p.a, an Italian company producing electromedical devices for neurophysiological diagnosis. To date, the company is a leader in the field of software and hardware development of intuitive and powerful platforms capable of adapting to any workflow. Special emphasis is placed on electroencephalogram (EEG) review and analysis software that supports easy navigation of EEG traces from routine, long-term monitoring (LTM), intensive care unit (ICU) monitoring, ambulatory EEG, and research studies. The main features of this software include the ability to customize user roles to fit the workflow, the presence of data analysis and management tools and sophisticated archiving capabilities.

The main objective of this thesis project is to develop a plugin that can allow users to access data files with Matlab, through the use of Micromed EEG acquisition and processing software. The main need was to create a plugin to export data recorded with Micromed devices in Matlab format.

Further goal achieved during the development of the thesis work was to make EEG data, from Micromed files, visible and editable by using the EEGLAB interface, a Matlab toolbox for processing electrophysiological signals. Moreover, thanks to the experience in Micromed I was able to follow the project throughout its life, investigating the operating procedure that applies to all activities related to the analysis, design, development and qualification of new products or to the revision/modification of existing projects/products. The goal, therefore, was to interface with different departments with the aim of receiving the business needs, assess the feasibility of changes, implement the plugin, validate and test the work and, finally, release it on the market.

Chapter 1

Electroencephalography

1.1 A theoretical overview

In the field of diagnostics, among the existing biological signals are very important the bioelectric signals that manifest as a change in the electrical potential between specialized tissues or organs in a living organism; they are an indicator of the physiological state of the subject.

One particular type of bioelectrical signal is the electroencephalogram (EEG) whose clinical uses are:

- Detecting and quantifying deficits in brain activity;
- Diagnosis of epilepsy;
- Monitoring during anesthesia;
- Study of sleep phases;
- Effect of medications or drugs or meditation;

The electroencephalographic technique was invented in 1929 by Hans Berger, who discovered that there was a difference in electrical potential between needles inserted into the scalp or between two small metal discs (electrodes) placed in contact with the scalp. The EEG was first used to study brain function in animals and then on humans.

The EEG recording is obtained by appropriately applying electrodes to the scalp. The electrodes record electrical events occurring in the underlying cortex, around the microVolt [μV] range. The traces of potential are recorded with a temporal resolution in milliseconds [ms] that allows, therefore, to detect events in real time.

Electroencephalography is examined for asymmetries between the 2 hemispheres (suggestive of possible structural pathology), excessive slowing (presence of slow delta waves as occurs in depression of consciousness, encephalopathy, and dementia), and abnormal wave patterns.

1.2 Detection system and Electrode placement

The EEG signal recording system, called electroencephalograph, is composed by an acquisition unit for signals measured on the scalp, a signal processing unit and a data display/storage unit. The acquisition unit adopts measurement electrodes typically housed on a costum headset that is placed on the patient's head.

The surface electrodes for EEG can be attached to the skin with the help of adhesive collars, patches or, as mentioned, a costumized headset. In order to obtain a good recording of EEG signals, it is essential to correctly position the electrodes on the scalp and perform a correct derivation. For this reason, a special committee of the International Federation of Societies of Electroencephalography and Clinical Neurophysiology (IFSECN) led by H. Jasper, studied a specific system of electrode placement, to be used in all laboratories. This system, presented at the Second International Congress in Paris in 1949 and published by Jasper in 1958, is still universally used and is known as the International System 10-20 (*SI 10-20*). This type of system establishes the position of the electrodes on the scalp with a measurement system that takes into account well-defined anatomical repère points, so that the measurements are proportional to the size and shape of the skull; in addition, it ensures adequate coverage of the whole head with electrodes placed in standard positions identified by a letter and a number. The letters used refer to positions on the scalp [1]:

- Occipital (O);
- Parietal (P);
- Central (C);
- Frontal (F);
- Temporal (T)

Numeric indexes identify the side of the brain:

- Even numbers for right side;
- Odd numbers for left side;
- Z letter for the middle line.

In practice, to correctly place electrodes on the scalp according to *SI 10-20*, ideal lines must be drawn from particular anatomical repère points. These fundamental lines are perpendicular to each other and are represented by:

- As shown in fig. 1, anteroAntero-posterior middle line, joining the *nasion* (upper hairline of the nose) to the *inion* (prominence at the base of the occipital bone), passing through the vertex; along this line we identify the 5

standard positions. Considering the total distance in centimeters between *nasion* and *inion*, the fronto-polar point (Fp_z) and the occipital point (O_z) are identified at 10% of the total distance from *nasion* and *inion*, respectively. All other points are calculated at 20% of the interposed distance between Fp_z and O_z (the designation 10-20 derives precisely from this percentage calculation of the distance between the electrodes). According to the ideal arrangement the middle electrode should be placed exactly in the middle of the line between *nasion* and *inion*;

- As shown in fig. 2, latero-lateral coronal line, connecting the right and left preauricular points via the central vertex point. On this line the temporal electrodes should be placed at 10% of the total distance, starting from the preauricular point, while the lateral central electrodes should be placed at 20% from the temporal points and the median central point [1];

Starting from these two fundamental lines, perpendicular to each other, it is possible to identify the positions of the electrodes placed longitudinally to the sides of those on the middle line and those placed on the two coronal frontal and parietal lines, respectively to the front and back of the coronal line crossing the vertex. The frontopolar electrodes (Fp_2 and Fp_1) are placed along the longitudinal line, at 10% of the distance lateral to Fp_z , whereas for the occipital electrodes (O_1 and O_2) 10% is calculated relative to O_z . The positions of the inferior frontal (F_8 and F_7) and posterior temporal (T_6 and T_5) electrodes are instead determined at 20% of this line starting, respectively, from Fp/Fp_1 to O_1/O_2 . The remaining frontal (F_4 and F_3) and parietal (P_4 and P_3) electrodes are placed along the coronal frontal and parietal lines, equidistant between the medial and temporal lines on each side.

In a short time, the *SI 10-20* has allowed to obtain a standard positioning of the electrodes on the scalp, allowing a reliable comparison of the results described in the various laboratories around the world; however, the above method of application of the electrodes not being without criticism has given way to application methods with higher resolution such as the systems 10-10 and 10-5.

With 10-10 system it is possible to identify many more localizations on the scalp thanks to 81 electrodes placed on it; instead, the 10-5 system is represented in the fig. 3.

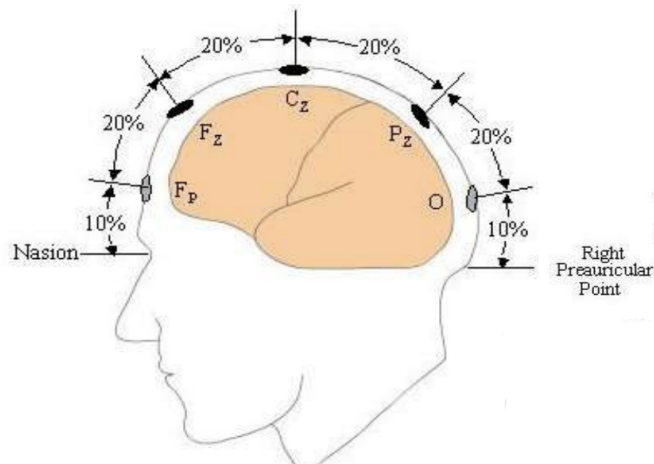


Figure 1: Representation of the antero-posterior midline connecting nasion and inion. Distance between electrodes are reported in percentage. Following the 10-20 system, Fp is the frontopolar electrode, Fz represents the frontal, Cz the central, Pz the posterior and O the occipital [1]

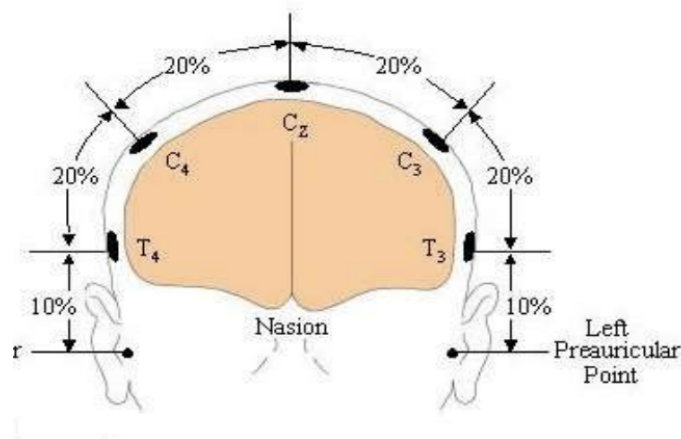
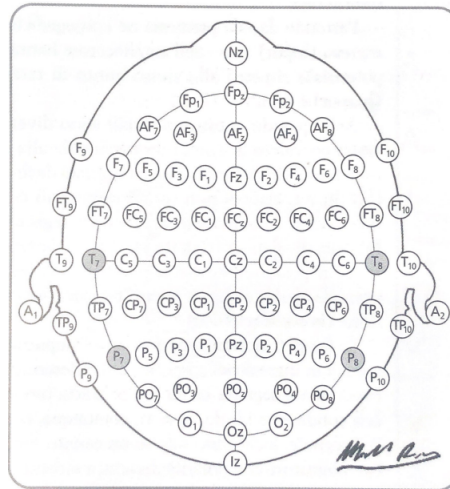
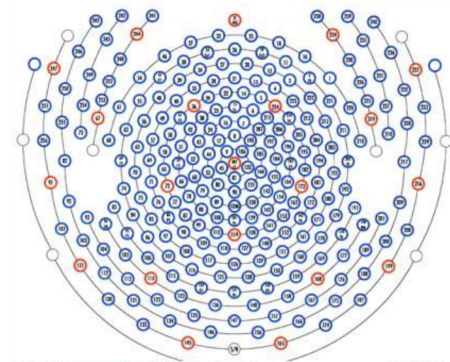


Figure 2: Representation of the latero-lateral coronal line connecting the two pre-auricular points passed through the vertex. Distance between electrodes are reported in percentage. Following the 10-20 system, T4 represents left temporal lobe, C4 the central left hemisphere, Cz the central, C3 the central right hemisphere, T3 the right temporal lobe [1]



(a) Placement of electrodes according to SI 10-10. This system involves placing 75 electrodes on the scalp, arranged along 11 sagittal and 9 coronal chains. In this system, electrodes defined in SI 10-20 as T3/T4 (right and left temporal lobe) are replaced by T7/T8 and electrodes T5/T6 by P7/P8 (right and left parietal).



(b) High Density EEG: Placement of electrodes according to SI 10-5.

Figure 3: Two different system: (a) 10-10 and (b) 10-5 [1]

1.3 Derivations

Among the many factors that influence how the brain bioelectric signal is displayed on the EEG trace, in addition to the placement of the electrodes on the scalp, the way the electrodes are connected to the amplifiers plays a key role.

Specifically, in electroencephalography, the combination of electrodes and how they are connected to the amplifier (mounts and leads) are essential. For historical and practical reasons, the EEG is usually visualized as a set of traces indicating how potential differences change in time. In a traditional trace (analog EEG), each trace is the result of connecting two electrodes to amplifiers and filters, with subsequent passage of the captured signal to the galvanometer (an instrument used to measure small intensities of electrical currents) and the writing pen.

With digital EEG, everything has been replaced by computer hardware and software, but every trace continues to be marked as a channel, whether it is produced by an analog device or processed by a computer.

The main types of derivation include:

- Referential derivations (common reference, average reference):

in *common-reference* recording mode, each electrode placed on the scalp is referenced to a common electrode placed at a point x , on the scalp or elsewhere. The common reference electrode should be as neutral as possible from the electrical point of view (not contaminated by other electrical potentials and biological body potentials), this is rare in reality. The major drawback of common referencing is referential contamination; when the referential electrode is located near a potential peak there is a change in the voltage of all the electrodes referenced to them. Equipotential electrodes with the reference go to zero while those less involved with the reference show pseudo positivity. In theory, then, given a known electric field, it would not be difficult to determine which waveform would appear on the EEG channel depending on the reference. In practice, however, the reasoning must be reversed, we must understand the distribution of the potential without knowing a priori neither the site of localization of the EEG phenomenon nor its polarity (positive or negative). Many problems encountered in the use of the common reference can be overcome with the *average reference*, also known as the mathematical reference, introduced into electroencephalography in 1950 by Goldman and Offner [2]. In this case the average potential of all the electrodes is considered as a reference: the potentials of the single electrodes are therefore measured with respect to an instantaneous average value obtained by summing up all the potentials of the applied electrodes. The average reference potential will therefore be closer to zero the greater the number of electrodes. One of the properties of the mathematical mean of a series of numerical values is

that the sum of the differences of the mean is zero. This means that on the EEG trace we will always have positive or negative deflections, compared to the zero value of the reference.

- Bipolar derivations :

in this case the potential difference is calculated between pairs of electrodes, placed along chains (longitudinal or transversal) in which an electrode is in common between two successive channels. In this way an event located under a certain electrode will produce an equal but opposite deflection in the two adjacent electrodes which precede and follow it in the electrode chain.

Because EEG patterns are highly variable (focal or diffuse, transient or persistent), there is no single ideal derivation to highlight all types of activity. A first important factor to consider when choosing leads is inter-electrode distance, and this is especially true for common active and bipolar leads.

In the case of bipolar leads the distances between the paired electrodes are small and equal and this highlights the fastest EEG activities; in the case of the common active reference, the distances are instead larger and unequal and this allows an amplification of the signal, highlighting better the slow activity.

The figure (fig. 4) shows how the visualization of a real epileptic focus changes, recording with digital equipment, depending on the lead visualized in bipolar lead, medium referential (AVG), active common referential (G2).

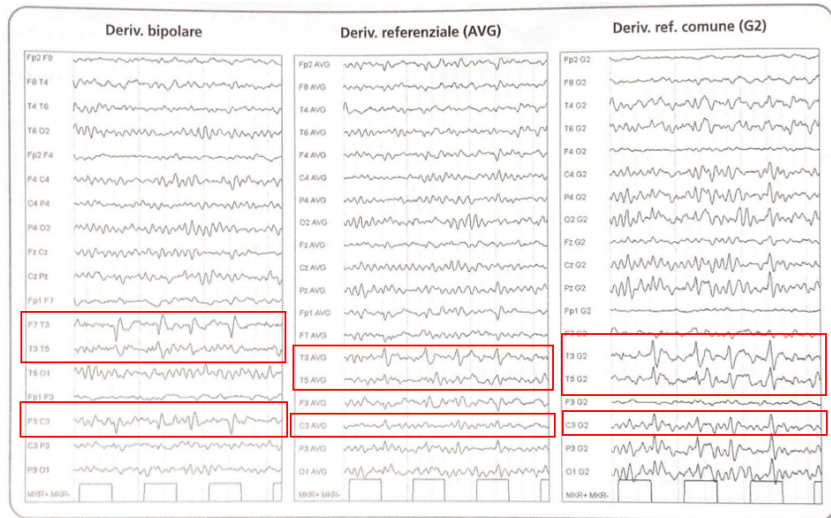


Figure 4: Imaging of a real epileptic focus (spikes in the left middle temporal site), recorded with digital equipment, depending on the derivation visualized in bipolar and referential derivation. Spikes corresponding to seizures in the respective derivation are highlighted in red [1]

1.4 Electrode Mountings

Mounting refers to the specific way the electrodes are connected to the EEG recording channel. In EEG laboratories, there is a wide variety of montages used for routine recordings; this multitude prevents the proper exchange of information between specialists in the field and can be a disadvantage to the patient. To resolve this issue, both the International Federation of Clinical Neurophysiology and the American Clinical Neurophysiology Societies have published guidelines for fitting. The different assemblies are referred to as longitudinal bipolar (LB), transverse bipolar (TB), or referential (R).

Each montage is defined for 16, 18, and 20 channels.

In brief, the main recommendations describing the guidelines are:

- Record while simultaneously viewing at least 16 EEG channels;
- Place at least 21 electrodes according to SI 10-20;
- Use bipolar and referential assemblies;
- Indicate at the beginning of each assembly the electrode connections for each channel with a simple and easily understood connection mode;
- In bipolar leads, electrode connections should follow continuous lines with equal interelectrode distances;
- Electrode progression should be antero-posterior;

The figure (fig. 5) shows the bipolar mounts used routinely, depending on the number of electrodes applied to the scalp, which is also dependent on the size of the patient's head.

Regarding the common reference mounting, the international guidelines recommend as reference the right auricle (A2) for right electrodes and the left auricle (A1) for left electrodes. Currently, in digital electroencephalography, G2 can also be used as active reference, placed on the midline anterior to Fz [1].

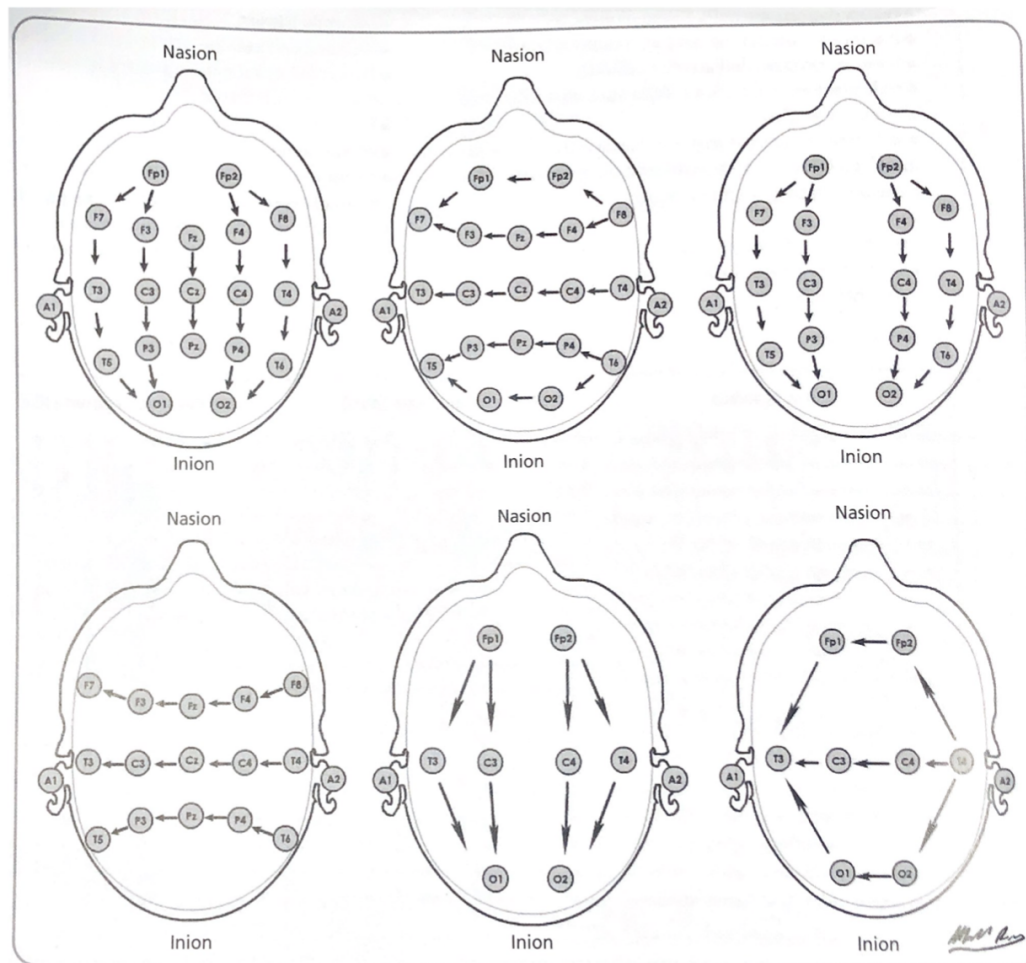


Figure 5: Bipolar, longitudinal, and transverse mounting groups routinely used, depending on the number of electrodes applied to the scalp, also dependent on its size. The right auricular (A2) for right electrodes and the left auricular (A1) for left electrodes are recommended as reference [1]

1.5 Main waves

During the analysis of EEG signals it is possible to identify several areas, with distinct amplitudes and characteristics; each of these areas is characterized by a wave signal:

- Alpha (α) rhythm, is the basic rhythm present in EEG, also called "Berger rhythm"; it is possible to distinguish the slow α (8-9 Hz), the intermediate α (9-11.5 Hz) and rapid α (11.5-13 Hz), with an average amplitude of 30 μV .

This rhythm is recorded with closed eyes in an awake subject, especially between the occipital and parietal electrodes, compared with the posterior central and temporal electrodes. Alpha waves (8-13 Hz) are characterized by waking conditions but at mental rest but are not present in sleep;

- Beta (β) rhythm, is distinguished into slow β (13.5-18 Hz) and fast β (18.5-30 Hz) and has an average electrical voltage of 19 μV (8-30 μV).

Beta waves are dominant in a subject with eyes open and engaged in any brain activity, almost continuous in alert states but also during dream sleep (during dreaming);

- Theta (θ) rhythm is dominant in the newborn but also present in many adult brain disorders, in states of emotional tension and in hypnosis. It is distinguished in slow θ (4-6 Hz) and rapid θ (6-7.5 Hz), with an average voltage of 75 μV .

In normal conditions the theta phase occurs in the first minutes of falling asleep, when a subject is still in a state of drowsiness, where it is then alternated by graphemes called sleep spindles.

- Sigma (σ) rhythm, which appears during sleep. It appears at a frequency of 12-14 Hz and electrical voltage of 5-50 μV .

The sigma rhythm shows up as trains of waves together with other graphical elements called K complexes.

- Delta (δ) rhythm, which appears about 20 minutes after sleep start (estimated time to reach deep sleep stage). It is characterized by a frequency between 0.5 and 4 Hz and an average electrical voltage of 150 μV .

Delta waves are not present under physiological conditions in the waking state in adulthood; they are predominant in childhood, occur during general anesthesia of a subject, in some brain diseases or in general dysmetabolic diseases, such as hyperazotemia.

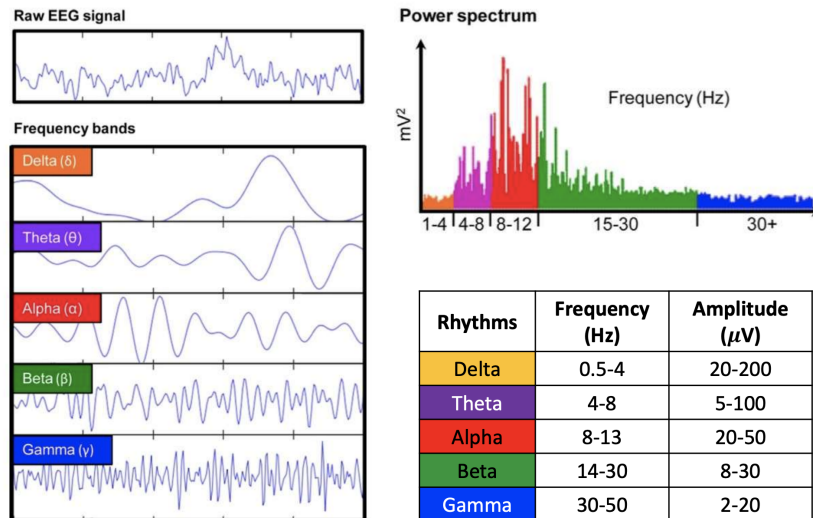


Figure 6: Characteristic EEG waves. In red, Alpha (α) rhythm, is the basic rhythm present in EEG; in green, Beta (β) rhythm, is dominant in a subject with eyes open and engaged in any brain activity, almost continuous in alert states but also during dream sleep (during dreaming); in orange, Delta (δ)rhythm, is predominant in childhood; in violet, Theta (θ) rhythm is occurs in the first minutes of falling sleep; in blue, gamma (γ) rhythm is hardly visible in an EEG recording given the limited amplitude [1]

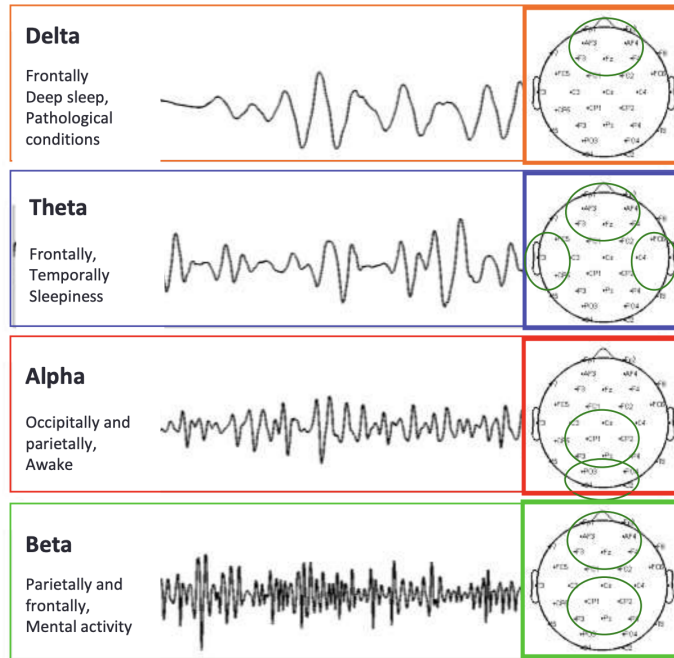


Figure 7: Spatial localization of EEG rhythms: the parts of the scalp from which the main characteristic EEG waves originate are highlighted [1]

1.6 Clinical Applications

The EEG is a non-invasive procedure, and for this reason it is suitable for subjects of any age and allows the doctor to observe brain activities: it is used for research purposes in neuroscience (for example for the study of disorders of the sleep or memory) and for medical evaluation and diagnosis of certain diseases such as epilepsy or primary brain tumors and any metastases that lead to epileptic episodes.

The EEG allows not only the recording of anomalous activities, but also their location. The EEG allows to study the brain condition in different states of the patient:

- During wakefulness and during sleep;
- During a particular activity or during the inactivity of the subject.

The EEG then makes it possible to identify the activations caused by abnormal changes in electrical activity, such as epilepsies.

It has diagnostic validity as it allows the diagnosis of numerous diseases, but it can also be used for the diagnosis of exclusion (i.e. to exclude a pathology or cerebral anomalies, rather than to confirm their presence).

The conditions that benefit most from the exam include:

- Tumor or even circulatory brain lesions;
- Degenerative diseases (Alzheimer);
- States of reduced consciousness (such as, vegetative states, brain death);
- Metabolic alterations;
- Response to certain medications or drug use;
- Head trauma;
- Psychiatric diseases;
- Encephalopathies;
- Stroke;
- Cerebrovascular disorders.

Depending on the type of examination performed, the duration of the EEG can vary from minutes (basic EEG) to hours (dynamic EEG). In any case, it is a non-invasive examination and on average well tolerated by patients.

Chapter 2

Case Study: Brain Quick Software[®]

This chapter includes an overview of Brain Quick (BQ) Software which is the software that manages all Micromed acquisition and review systems for EEG, Video EEG, Long Term Epilepsy Monitoring (LTM), Stereo EEG, Ambulatory EEG/PSG, EEG recording during functional MRI, evoked potentials.

2.1 Main Software Features

BQ Software is designed to help physicians in recording, reviewing and analyze data coming from Micromed digital acquisition systems and it can be used in EEG, LTM, PSG and EP exams for neurophysiologic studies. The software could also be used for cortical stimulation during electroencephalography examinations (i.e. stereo EEG) in combination with specific Micromed stimulators. In addition, BQ Software can monitor physiological measurements such as Intracranial Pressure (ICP), the Oxygen of the Brain Tissue ($PbtO_2$), Cerebral Perfusion, Heart Rate (HR) and the Oxygen Saturation in the Blood (SpO_2) coming from third- party interfaced medical devices.

Some functional analysis tools are provided as predefined options or software, but results from these tools should never replace critical interpretation and clinical conclusions that pertain only to the physician. With reference to this, the use of BQ software is reserved for physicians, technicians, or other health care professionals who are educated in the recording of biopotentials. This is because the software is not designed to continuously monitor the functionality of the Central Nervous System autonomously, as it is not equipped with adequate alarms that replace medical surveillance.

Therefore, the use of BQ Software should always be done under the supervision of a physician or a qualified technician.

BQ software is available in a variety of languages so that it can be used internationally. Thus, the software is able to display information in all languages using a multitude of characters. The language can be set by the archive data manager. It is also able to handle non-character languages such as Chinese, Russian, Hebrew and so on.

The BQ software is designed to be multifunctional giving the possibility to manage archived and non-archived historical studies, make remote reviews of ongoing studies performed by SystemPlus Evolution (Acquisition system that generates Micromed EEG files - TRC files), create reports and manage templates, make multiview examinations, start a new recording of the EEG examination according to predefined or selected protocols or start a new recording with the last protocol used, create a new history, a new report and manage report templates.

BQ manages the configuration of the various settings on three levels:

- User : the setting is unique for a specific user on a specific machine; at this level the defined settings are preferences such as branch labels, cursor time, background etc... or screen capture preferences.
- Unit : the setting is unique for all users on a specific machine; at this level the defined settings are electrode position configuration, event definition, average (AVG) reference configuration, calibration monitoring.
- Central : the settings are applied to all machines and for all users of the system; at this level the defined settings are Internationalization, Language, Notch.

2.2 Software User Interface

The interface of BQ Software presents, first of all, the title bar from which it is possible to view the type of open window (EEG trace or Report) and the information regarding the patient. A ribbon bar (fig. 8) contains the toolbar which allows access to all the functions and their visualization in different modes. In this way, the user can customize all the tabs by defining the functions that must be present.

This software, moreover, makes it possible during acquisition and review, the tagging of general events such as notes, digital triggers, annotations, flags and selections thus allowing a customization of the path. In the same way it is possible to tag events such as High Frequency Oscillations (HFO).

Through the interface it is possible to view the properties of the EEG trace such as start/end and duration of the recording, number of raw channels recorded, sampling rate, resolution and acquisition device. To complete the software, the "video" option allows the user to define the characteristics of the recorded video; furthermore, the user can also access the acquisition protocol.

Among the many features listed so far, BQ has in correspondence of the main button, the "Export" function which has been updated during my internship experience at Micromed, as we will in chapter 4. BQ Software has been designed with an option that allows the user to perform different types of exports (fig. 9) such as :

- Export in Ascii (American Standard Code for Information Interchange)
- Export in edf (European Data Format - simple and flexible format for exchange and storage of multichannel biological and physical signals)
- Export Event
- Export Analyzer Data

The export function is available from the menu that opens by clicking on the BQ Application button. In chapter 4, we will see how this menu has been updated with an export function in .mat files, that is the proprietary file format that can be managed in the Matlab programming environment.

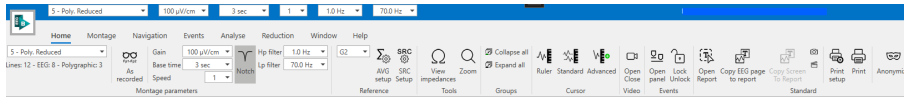


Figure 8: Example of BQ Software Ribbon Bar; main features are shown such as montage parameters, events, filter parameters, analyzers

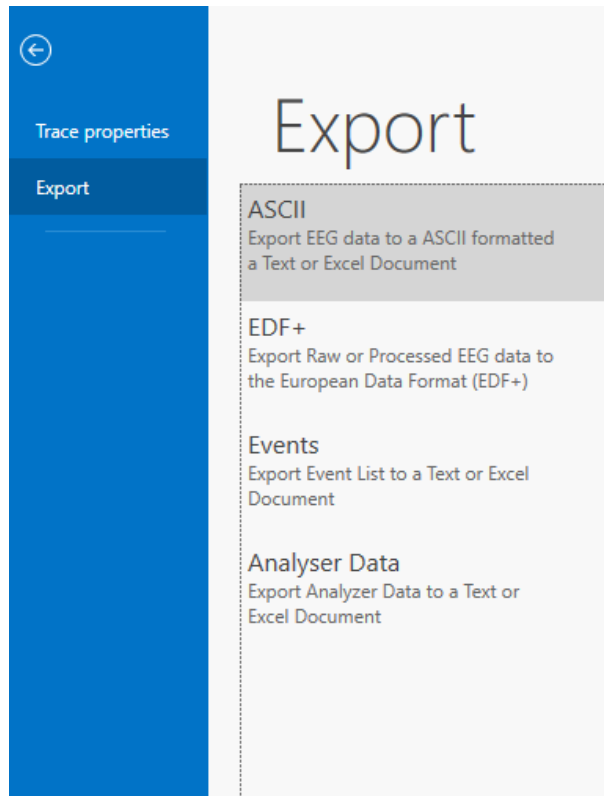


Figure 9: BQ Software Export Features: existing export features in the EEG acquisition software are shown

2.3 Events

As anticipated, BQ Software allows to customize the EEG trace. In this paragraph the different types of customization are discussed in detail (fig. 10).

Regarding the Notes, the software allows the user to insert notes to distinguish particular events during the review and acquisition. Notes are displayed as vertical blue lines and direct insertion of comments can take place.

As for Digital Triggers, the software allows the user to mark the track using events of this type during acquisition and review. Digital Triggers are composed of a numerical value stored inside the track and an analog trigger; the numerical value informs about the type of event that generated the trigger. This type of event is displayed with green vertical lines in the EEG page and in the time bar.

In addition, BQ Software allows the user to mark a piece of track with HFO events during the review and acquisition phase. BQ enables automatic HFO detection by automatically marking the parts of the trace where they are present. The various waves in the EEG tracing corresponding to particular groups of applied electrodes can be seen in different colors from each other.

Moreover, BQ allows the user to define custom events in order to create a new type of event by choosing its graphical and input properties. The following information can be displayed for each event: type (each event corresponds to a label), time position in the trace (time the event was added to the trace), duration (the event has a start date and an end date).

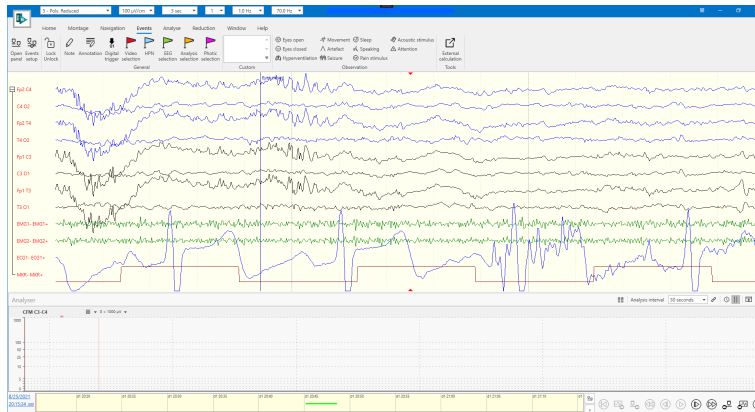


Figure 10: Example of customization of an EEG trace: during review the user can add different types of events (colorful flags on the ribbon bar) with vertical lines. It is shown how the user can select parts of the trace to indicate particular events such as hyperventilation, seizures, sleep, artifacts but also to select parts of the trace that are clinically interesting

Chapter 3

Product Development Protocol (PDP)

In order to implement a change to the main software and thus add the new export functionality, a new software version needs to be released. To do this, documentation must be produced that is useful for the company to be able to supervise the activities carried out; with this purpose, company protocol was followed to correctly perform the thesis work.

3.1 General Informations

Product Development Protocol (PDP) refers to an operating procedure that applies to all company activities related to the analysis, design, development and qualification of new products or the revision of existing designs/products.

The purpose is to provide guidance and determine responsibilities for the design and implementation of all new, custom and experimental devices, including prototypes and product variants. In the PDP, it must be considered that there are different types of projects; in the specific case of this thesis work, a so-called minor project (MPR) was developed that requires the existence of a device that meets the regulatory requirements applicable to the intended markets. Specifically, the minor modification made to the existing EEG acquisition software consists of the addition of a new feature that does not require a new regulatory submission or substantial verification and validation.

Regarding the design documentation for software projects, we refer to the Software Requirement Specification (SRS) which must include the implementation of a specific Functional Requirement Document (FRD), an overview of the hardware device features that are controlled by the software and a description of the expected operating environment. The SRS describes the implementation of the requirements for the software device.

In terms of the relationship between the FDR and SRS, the FDR describes what the product must do (user requirements) and the SRS describes how the requirements in the FRD will be translated into software constraint capabilities (design inputs) [3]. Below we will detail the steps approached to design the plugin that is the subject of this thesis.

3.2 Stage 1: New Idea and Feasibility

The purpose of this phase is to identify potential ideas for new products or reasons for redesigns, determine the scope or variant of the project, present them to the project management team, and evaluate the costs and resources involved.

In practice, following meetings with the R&D department team, it was decided to implement a program that would allow the export of data corresponding to EEG tracings acquired with Micromed devices.

The project should have adapted to the needs of the end user and therefore aimed to export key values such as sampling rate, various acquisition conditions, number and type of events placed on the trace during and after acquisition (in review). Next, we also wanted to expand the visibility of the exported values through the EEGlab tool (please refer to chapter 4 section 4.2); this required further planning of both the work and the resources to be employed.

3.3 Stage 2: Design

In this stage it is expected to complete the majority of the design work, define user requirements, and gather associated information to identify and refine applicable design parameters and create design specifications. This phase basically involves answering the questions:

- What should the device do?
- Who is it intended for?
- What are the known constraints for development?
- What does it look like?
- How is it used?

In order to answer most of these questions formally, I needed to draft the FRD document; as introduced in the previous paragraphs, this document describes the software requirements for the new export plugin in Matlab to be added to Brain Quick. It is organized into two sections: the first contains the user requirements and the second contains the functional requirements that satisfy those requirements. The following is a schematic outline of the contents of that document(fig. 11)(fig. 12).

User Needs

User Need number	Description
MP-UN1	Export of EEG traces in Matlab User needs to open EEG files recorded by Micromed systems in Matlab.
MP-UN2	Electrode The user needs to be able to see which electrodes are recorded, the order and number of electrodes used.
MP-UN3	Events The user needs to display in Matlab different types of events marked in the EEG trace during recording or in review with Brain Quick and saved in EVT file
MP-UN4	Notes The user needs to see in Matlab the notes added in the EEG trace during recording or in review and saved in the TRC file.
MP-UN5	Flags The user needs to see in Matlab the portions of data highlighted with red, green and blue FLAGS in the EEG trace during recording or in review and saved in the TRC file because they are areas of interest or for further processing.
MP-UN6	Trigger The user must see in Matlab the digital triggers added in the EEG trace during recording or in review and saved in the TRC file.

Figure 11: Functional Requirements Document: User Needs

Functional Requirements

Requirement number	Description
MP-FR1	<p>The software shall convert TRC files to .mat files (MATLAB) containing the corresponding information. The software shall also manage EVT files.</p> <p>The operation can be done directly through Brain Quick with the addition of a new feature: "Export .mat".</p>
MP-FR2	<p>The software shall identify the stored electrodes, their order and number. It shall be possible to view the electrodes registered and associated with the correct channel.</p>
MP-FR3	<p>The software shall allow to visualize in the correct position events entered in recording and review with Brain Quick.</p> <p>The software shall recognize and plot on the trace all events contained in the EVT files.</p>
MP-FR4	<p>The software shall allow to see in the correct position the notes included in the recording and review steps.</p>
MP-FR5	<p>The software shall show in the correct position the red, blue and green flags placed during recording and review.</p>
MP-FR6	<p>The software shall show in the correct position the digital triggers placed during recording and review.</p>
MP-FR7	<p>The software shall be able to convert recorded EEG files up to 256 channels.</p>
MP-FR8	<p>The software shall be able to convert EEG files recorded up to 16KHz.</p>

Figure 12: Functional Requirements Document: Functional Requirements

3.4 Stage 3: Development, verification and validation

This is the operational phase in which all the information extracted is considered design output.

In agreement with the R&D department managers, it was decided to develop the plugin in the C++ programming environment. Subsequently, an entire chapter will be spent on precisely describing this crucial stage for the finalization of the project.

As in all new product development processes, there was a testing phase conducted in Micromed laboratories. How this phase was conducted and its results will be discussed in chapter 5.

3.5 Stage 4: Market Release and Post-marketing surveillance

Market Release refers to the stage at which the product is formally released for commercial distribution. Following this is the post-marketing surveillance phase, which begins as soon as the product is released for market and continues throughout the life of the product. Inputs for post-marketing surveillance include information and data from physicians, clinics, and hospitals.

During this thesis project this stage was not reached because the export functionality will be formally added in a later version of the BrainQuick software to be released in the coming months.

Chapter 4

Matlab Plugin Development

In this chapter I will describe in detail the predominant part of the entire project, the development phase in two different programming environments: C++ and Matlab.

4.1 C++ Coding Environment

During the development phase, I used Visual Studio's Integrated Development Environment (IDE), which offers a feature set that allows you to manage large and small code projects, write and refactor code, detect and correct errors through static analysis, and powerful debugging tools. This set of items is designed to guide every step needed to manage projects, write, test and debug code, and then deploy it to another computer [4].

C++ quickly showed to be the appropriate environment for the development of the plugin first because the company's software infrastructure was primarily created in this programming environment and then because, through it, I could best manage the so-called "header" of the proprietary Micromed TRC file.

In fact, any Micromed EEG file contains a data file structure that can be easily interpreted. Data file is broken down into two main sections: the header, which contains the patient and setup data and a variable length trailer that contains the digitised trace data. The sequence of this data can be interpreted with the header block. Files have the capability to store up to 256 individual channels of signal data, the number stored in each file may vary and this information may be found in the header [5].

In addition to this, in the new Micromed Brain Quick EEG Application platform there is the possibility of importing events generated by external plugin or programs. These are additional event types that are grouped in files other than the TRC. Again, a precise data structure [6] must be followed in order to obtain a correct definition of the event. During development, it was decided that data export would also concern this type of event.

So, after defining the main libraries that are useful in the C++ program, I initialized the strings that automatically call, via path, the TRC and EVT file whose data we want to export:

```
1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <iomanip>
5 #include <string>
6 #include <tinymatwriter.h>
7 #include <cmath>
8 #include <stdexcept>
9 #include <rapidxml/rapidxml.hpp>
10 #include <sstream>
11 #include <chrono>
12 #include <map>
13 #include <algorithm>
14 #include <settings.h>
15
16 using namespace rapidxml;
17 using namespace std;
18
19 string TRCPATH = "";
20 string EVT_PATH = "";
21 string OUTPUT_PATH = "";
```

Subsequently, I defined useful structures for export purposes; below are the most relevant ones:

```
1 struct MarkerPair {
2     int begin;
3     int end;
4 };
5
6 struct Event {
7     char description[64];
8     MarkerPair selection[MAXEVENT];
9 };
10
11 struct Annotation {
12     unsigned int Sample;
13     char Comment[40];
14 };
```

```

1 struct Trigger {
2     unsigned int Trig_Sample;
3     unsigned short int Trig_Value; };
4
5 struct Electrode
6 {
7     unsigned char Status;
8     unsigned char Type;
9     char Positive_input_label [6];
10    char Negative_input_label [6];
11    int Logic_Minimum;
12    int Logic_Maximum;
13    int Logic_Ground;
14    int Physic_Minimum;
15    int Physic_Maximum;
16    unsigned short Measurement_Unit;
17    unsigned short Prefiltering_HiPass_Limit;
18    unsigned short Prefiltering_HiPass_Type;
19    unsigned short Prefiltering_LowPass_Limit;
20    unsigned short Prefiltering_LowPass_Type;
21    unsigned short Rate_Coefficient;
22    unsigned short Position;
23    float Latitudine;
24    float Longitudine;
25    unsigned char PresetInMap;
26    unsigned char IsInAvg;
27    char Description [32];
28    float X;
29    float Y;
30    float Z;
31    unsigned short Cordinate_Type;
32    unsigned char Free [24];
33 };
34
35 struct Evt {
36     string Name;
37     string BeginDateTime;
38     string EndDateTime;
39     string Text;
40     string Guid; };

```



```

1 struct EvtType {
2     string Name;
3     string Guid;
4     string Description;
5 };
6
7 struct DateTime {
8     unsigned char Day;
9     unsigned char Month;
10    unsigned char Year;
11    unsigned char Hour;
12    unsigned char Minute;
13    unsigned char Second;
14 };

```

I defined a particular class to visualize the data using the EEGLab tool, which we will see in section 4.2:

```

1 class EegLabEvent {
2 public:
3     string type;
4     double latency;
5     int urevent;
6 };

```

Then I created a structure for the “DateTime” in order to export the time when the trace is created and events are affixed. This information will be used in testing, as we will see in chapter 5.

```

1 struct DateTime {
2     unsigned char Day;
3     unsigned char Month;
4     unsigned char Year;
5     unsigned char Hour;
6     unsigned char Minute;
7     unsigned char Second;
8 };

```

```

1 time_t timeParse(const string& str, struct tm* datetime)
2 {
3
4     int D, h, m, s, Y, M, cns;
5     sscanf_s(str.c_str(), "%d-%d-%dT%d:%d:%d.%dZ",
6     &Y, &M, &D, &h, &m, &s, &cns, str.size() + 1);
7     datetime->tm_sec = s;
8     datetime->tm_min = m;
9     datetime->tm_hour = h;
10    datetime->tm_mday = D;
11    datetime->tm_mon = M - 1;
12    datetime->tm_year = Y - 1900;
13    return mktime(datetime);
14 }

```

As already anticipated, additional events are contained in an EVT file that is an XML (Extensible Markup Language) format file; XML documents contain data enclosed within tags that define the structure and meaning of the data [7]. Therefore, in the C++ code it was necessary to read this file to extract additional types of events:

```

1 void ParseEvtFile()
2 {
3     xml_document<> doc;
4     xml_node<>* root_node;
5
6     // Read the xml file into a vector
7     ifstream evtFile(EVTPATH, std::ios::out | std::ios::binary)
8     ;
9     if (evtFile) {
10        vector<char> buffer((istreambuf_iterator<char>(evtFile)),
11        istreambuf_iterator<char>());
12        buffer.push_back('\0');
13
14        // Parse the buffer using the xml file parsing library
15        into doc
16        doc.parse<0>(&buffer[0]);
17
18        // Find our root node
19        root_node = doc.first_node();

```

```

1 //std::cout << Name of my first node is:
2   << root_node->name() << \ n ;
3   navigateNode(root_node);
4
5   for (list<Evt>::iterator itr = evtList.begin();
6   itr != evtList.end(); ++itr)
7   {
8       for (list<EvtType>::iterator itr2 =
9   evtTypeList.begin(); itr2 != evtTypeList.end(); ++itr2
10          )
11          {
12              if ((*itr).Guid == (*itr2).Guid)
13              {
14                  (*itr).Name = (*itr2).Name;
15              }
16          }
17      }
18      else {
19          throw std::invalid_argument(string
20      ("Cannot open input EVT file ") + EVT.PATH + " ");
21      }
22  };

```

The main has a referenced to a JSON (JavaScript Object Notation) file that will be used in the testing phase to define the file paths to be exported. Also, after defining several quantities, the TRC file to be exported is opened and read.

```
1 int main(int argc, char* argv) {
2     try {
3         // read a JSON file
4         std::ifstream i("settings.json");
5         json j;
6         i >> j;
7         auto settings = j.get<ExportSettings>();
8         TRCPATH = settings.trcPath;
9         EVT_PATH = settings.evtPath;
10        OUTPUT_PATH = settings.outputPath;
11
12        unsigned int data_start_offset;
13        unsigned int data_off_offset;
14        unsigned short int multiplexer;
15        unsigned short int numch;
16        unsigned short int Fc;
17        unsigned int campioni;
18        unsigned int length;
19        unsigned short int n;
20        const int MAXCAN = 256;
21        const int MAXLAB = 640;
22        uint32_t eventA_pos;
23        uint32_t eventB_pos;
24        unsigned int Electrode_pos;
25        unsigned int Code_pos;
26        const int MAXFLAG = 100;
27        unsigned int Flags_pos;
28        const int MAXNOTE = 1000;
29        unsigned int Note_pos;
30        const int MAX_TRIGGER = 8192;
31        unsigned int Trigger_pos;
32        unsigned char Date_pos;
33
34        // opening and reading the file
35        std::ifstream is(TRCPATH, std::ios::out | std::ios::
        binary);
```

The crucial action for the export to be accurate was to use pointers pointing to the different areas of the TRC file header containing useful information, such as:

```

1  if (is)
2      {
3          // Data length
4          is.seekg(0, is.end);
5          int length = is.tellg();
6          is.seekg(pointer_A, is.begin);
7
8          is.read((char*)&data_start_offset, sizeof(
9              data_start_offset));
10         std::cout << data_start_offset << endl;
11
12         // pointer to multiplexer
13         is.seekg(pointer to multiplexer, std::ios::begin);
14         is.read((char*)&multiplexer, sizeof(multiplexer));
15         std::cout << data_start_offset mp: << multiplexer <<
16             endl;
17
18         // number of channels
19         is.seekg(pointer to number of channels, std::ios::begin);
20         ;
21         is.read((char*)&numch, sizeof(numch));
22         std::cout << numero canali: << numch << std::
23             endl;
24
25         // sampling rate
26         is.seekg(pointer to sampling rate, std::ios::begin);
27         is.read((char*)&Fc, sizeof(Fc));
28         std::cout << Frequenza di campionamento: << Fc
29             << endl;
30
31         // pointer on data_start
32         is.seekg(data_start_offset, std::ios::begin);
33
34         // calculate the number of samples
35         /*std::cout << campioni : << (length -
36             data_start_offset) / multiplexer << std::endl;*/
37         campioni = ((length - data_start_offset) / multiplexer
38             );
39         std::cout << campioni : << campioni << std::
40             endl;
41
42         //I identify Event A (hyperventilation )
43         is.seekg(pointer to event A, std::ios::begin);
44         is.read((char*)&eventA_pos, sizeof(eventA_pos));
45         std::cout << start_offset_EventoA : <<
46             eventA_pos << std::endl;
47
48         // I identify EventB (selection of a part of the EEG
49             trace made by the doctor)
50         is.seekg(pointer to event B, std::ios::begin);
51         is.read((char*)&eventB_pos, sizeof(eventB_pos));
52         std::cout << start_offset_EventoB : <<

```

```

43         eventB_pos << std::endl;
44     // Code area
45     is.seekg(pointer to Code area std::ios::beg);
46     is.read((char*)&Code_pos, sizeof(Code_pos));
47     std::cout << Start_offset_Code : << Code_pos <<
        std::endl;
48
49     // Electrode
50     is.seekg(pointer to Electrode, std::ios::beg);
51     is.read((char*)&Electrode_pos, sizeof(Electrode_pos));
52     std::cout << Start_offset_electrode : <<
        Electrode_pos << std::endl;
53
54     // Order
55     short int* Order = new short int [MAX_CAN];
56     is.seekg(Code_pos, std::ios::beg);
57     is.read((char*)&Order, sizeof(MAX_CAN * sizeof(short
        int)));
58
59     Electrode* electrodes = new Electrode [MAX_LAB];
60     for (int e = 0; e < MAX_LAB; e++)
61     {
62         is.seekg(Electrode_pos + e * sizeof(Electrode), std
            ::ios::beg);
63         is.read((char*)&(electrodes[e]), sizeof(Electrode))
            ;
64     }
65
66     Event eventA;
67     is.seekg(eventA_pos, std::ios::beg);
68     is.read((char*)&eventA, sizeof(Event));
69
70     Event eventB;
71     is.seekg(eventB_pos, std::ios::beg);
72     is.read((char*)&eventB, sizeof(Event));
73
74
75     // Flags
76     is.seekg(pointer to Flags, std::ios::beg);
77     is.read((char*)&Flags_pos, sizeof(Flags_pos));
78     std::cout << Start_offset_Flags : << Flags_pos
        << std::endl;
79
80     MarkerPair flags [MAX_FLAG];
81     is.seekg(Flags_pos, std::ios::beg);
82     is.read((char*)&flags, sizeof(flags));
83
84     // Note
85     is.seekg(pointer to notes, std::ios::beg);
86     is.read((char*)&Note_pos, sizeof(Note_pos));
87     std::cout << Start_offset_Note : << Note_pos <<
        std::endl;
88
89     Annotation notes [MAX_NOTE];
90     /*Annotation* Note = new Annotation [MAX_NOTE];*/
91     is.seekg(Note_pos, std::ios::beg);

```

```

92     is.read((char*)&notes, sizeof(Annotation) * MAXNOTE);
93
94     // Trigger
95     is.seekg(pointer to trigger, std::ios::beg);
96     is.read((char*)&Trigger_pos, sizeof(Trigger_pos));
97     std::cout << Start_offset_Trigger : <<
          Trigger_pos << std::endl;
98
99     Trigger triggers[MAX_TRIGGER];
100    is.seekg(Trigger_pos, std::ios::beg);
101    is.read((char*)&triggers, sizeof(Trigger) *
          MAX_TRIGGER);
102
103    // Date
104    DateTime date;
105    is.seekg(pointer to DateTime date, std::ios::beg);
106    is.read((char*)&date, sizeof(DateTime));
107
108    // descriptor of code
109    is.seekg(pointer to descriptor of code, std::ios::beg)
        ;
110    is.read((char*)&Code_pos, sizeof(Code_pos));
111    std::cout << Start_offset_CodePos : << Code_pos
        << std::endl;
112
113    unsigned short Codes[MAX_CAN];
114    is.seekg(Code_pos, std::ios::beg);
115    is.read((char*)&Codes, sizeof(short) * MAX_CAN);
116
117    std::vector<std::string> electrodes_channels;
118    for (int i = 0; i < numch; i++)
119    {
120        int ch1 = (Codes[i] & 0xff00) >> 2;
121        int ch2 = Codes[i] & 0x00ff;
122        std::string s;
123        if (ch2 > 0) {
124            s += electrodes[ch2].Positive_input_label;
125            s += " - ";
126            s += electrodes[ch2].Negative_input_label;
127        }
128        if (ch1 > 0 && ch2 > 0)
129            s += " / ";
130        if (ch1 > 0) {
131            s += electrodes[ch1].Positive_input_label;
132            s += " - ";
133            s += electrodes[ch1].Negative_input_label;}
134        electrodes_channels.push_back(s);}

```

With the aim of extrapolating the number of events contained in the TRC and EVT files, I implemented a series of for cycles useful for the purpose:

```

1  int nEvents = 0;
2      /* for loop in eventA and eventB to understand the
3      effective number of events*/
4      for (int i = 0; i < MAXEVENT; i++) {
5          if (eventA.selection[i].begin > 0
6              && eventA.selection[i].begin < INT32_MAX
7              && eventA.selection[i].end > 0
8              && eventA.selection[i].end < INT32_MAX) {
9              nEvents++;
10             }
11             if (eventB.selection[i].begin > 0
12                 && eventB.selection[i].begin < INT32_MAX
13                 && eventB.selection[i].end > 0
14                 && eventB.selection[i].end < INT32_MAX) {
15                 nEvents++;
16             }
17             }
18             //I add the count of events in the evt file
19             nEvents += evtList.size();
20             // conto anche trigger, flag e note
21             for (int i = 0; i < MAX_TRIGGER; i++) {
22                 if (triggers[i].Trig.Sample > 0 && triggers[i].
23                     Trig.Sample < UINT32_MAX)
24                     nEvents++;
25             }
26             for (int i = 0; i < MAX_FLAG; i++) {
27                 if (flags[i].begin > 0 && flags[i].begin <
28                     UINT32_MAX
29                     && flags[i].end > 0 && flags[i].end < UINT32_MAX
30                     )
31                     nEvents++;
32             }
33             for (int i = 0; i < MAX_NOTE; i++) {
34                 if (notes[i].Sample > 0 && notes[i].Sample <
35                     UINT32_MAX)
36                     nEvents++;
37             }
38             MarkerPair* allEvents = new MarkerPair[nEvents];
39             std::vector<std::string> allEventNames;
40             EegLabEvent tempE;

```



```

1      int c = 0;
2      for (int i = 0; i < MAXEVENT; i++) {
3          if (eventA.selection[i].begin > 0
4              && eventA.selection[i].begin < INT32_MAX
5              && eventA.selection[i].end > 0
6              && eventA.selection[i].end < INT32_MAX) {
7              allEvents[c++] = eventA.selection[i];
8              string eventAName(eventA.description);
9              allEventNames.push_back(eventAName.length() > 0
10                 ? eventAName : EventA );
11             tempE.type = string("EventA") + (i == 0 ? "" :
12                 "(" + to_string(i) + ")") + "- begin";
13             tempE.latency = 1.0 * eventA.selection[i].begin
14                 /*/ Fc*/;
15             tempE.urevent = 0;
16             eegLabEvents.push_back(tempE);
17             tempE.type = string("EventA") + (i == 0 ? "" :
18                 "(" + to_string(i) + ")") + "- end";
19             tempE.latency = 1.0 * eventA.selection[i].end /*
20                 / Fc */;
21             tempE.urevent = 0;
22             eegLabEvents.push_back(tempE);
23         }
24         if (eventB.selection[i].begin > 0
25             && eventB.selection[i].begin < INT32_MAX
26             && eventB.selection[i].end > 0
27             && eventB.selection[i].end < INT32_MAX) {
28             allEvents[c++] = eventB.selection[i];
29             string eventBName(eventB.description);
30             allEventNames.push_back(eventBName.length() > 0
31                 ? eventBName : EventB );
32             tempE.type = string("EventB ") + (i == 0 ? "" :
33                 "(" + to_string(i) + ")") + "- begin";
34             tempE.latency = 1.0 * eventB.selection[i].begin
35                 /*/ Fc*/;
36             tempE.urevent = 0;
37             eegLabEvents.push_back(tempE);
38             tempE.type = string("EventB ") + (i == 0 ? "" :
39                 "(" + to_string(i) + ")") + "- end";
40             tempE.latency = 1.0 * eventB.selection[i].end /*
41                 / Fc*/;
42             tempE.urevent = 0;
43             eegLabEvents.push_back(tempE);
44         }
45     }

```

At this point, I used the C++ TinyMAT library [8] to handle writing a Matlab file in version “MATLAB 5.0” or later. This library implements a very simple interface to write Matlab files following the structure described in the official MathWorks documentation. [9].

First, I used the `TinyMATWriter_open()` function to create a new MAT file:

```
TinyMATWriterFile* mat = TinyMATWriter_open(OUTPUT_PATH.c_str());
```

To write inside the MAT file thus created, I took advantage of several TinyMAT functions which we see below generally:

- To write a string into the mat file :

```
TinyMATWriter_writeString (TinyMATWriterFile* mat,  
                           const char * name,  
                           const char * data)
```

- To write a single (numeric) value (as 1x1 matrix) into a MAT-file:

```
TinyMATWriter_writeValue (TinyMATWriterFile* mat,  
                          const char * name,  
                          T      data_real )
```

- To write a 2-dimensional double matrix in column-major order into a MAT file:

```
TTinyMATWriter_writeMatrix2D_colmajor (TinyMATWriterFile *mat,  
                                       const char * name,  
                                       const T * data_real,  
                                       int32_t cols,  
                                       int32_t rows )
```

- To write a 2-dimensional double matrix in row-major order into a MAT-file:

```
TTinyMATWriter_writeMatrix2D_rowmajor (TinyMATWriterFile *mat,  
                                       const char * name,  
                                       const T * data_real,  
                                       int32_t cols,  
                                       int32_t rows )
```

- To write an empty (double) matrix into a MAT-file:

```
TinyMATWriter_writeEmptyMatrix (TinyMATWriterFile * mat,  
                                const char * name  
                                )
```

- Start to write a struct-element:

```
TinyMATWriter_startStruct (TinyMATWriterFile * mat,
                           const char * name
                           )
```

- Low-Level-Interface for writing Cell-Arrays: starts a generic Cell-Array:

```
TinyMATWriter_startCellArray(TinyMATWriterFile * mat,
                              const char * name,
                              const int32_t * sizes,
                              uint32_t ndims
                              )
```

After this function invocation, simply use any TinyMATWriter-function to write the cell-array entires in column-major order with “name” left blank. Finally close the array by invoking TinyMATWriter_endCellArray(). You can nest severall startCellArray/endCellArray-invocations.

In particular, I used these functions to export data with a precise structure that could be replicated both in a normal Matlab environment but also through the EEGLab interface:

```
1 if (mat) {
2
3     TinyMATWriter_startStruct(mat, "EEG");
4
5     std::string mystr;
6     TinyMATWriter_writeString(mat, "setname", mystr);
7     TinyMATWriter_writeString(mat, "filename", mystr);
8     TinyMATWriter_writeString(mat, "filepath", mystr);
9     TinyMATWriter_writeString(mat, "subject", mystr);
10    TinyMATWriter_writeString(mat, "group", mystr);
11    TinyMATWriter_writeString(mat, "condition", mystr);
12    TinyMATWriter_writeEmptyMatrix(mat, "session");
13    TinyMATWriter_writeString(mat, "comments", mystr);
14    TinyMATWriter_writeValue(mat, "nbchan", numch);
15    TinyMATWriter_writeValue(mat, "trials", 1);
16
17    // time points
18    TinyMATWriter_writeValue<double>(mat, "pnts", 1.0 *
19    *   campioni)
20    TinyMATWriter_writeValue<double>(mat, "srate", 1.0
21    *   * Fc);
```

```

1
2     double* timesMillis = new double[campioni];
3     for (int t = 0; t < campioni; t++) {
4         timesMillis[t] = 1000.0 * t / Fc;
5     }
6     double minSec = timesMillis[0] / 1000;
7     double maxSec = timesMillis[campioni - 1] / 1000;
8
9     TinyMATWriter_writeValue(mat, "xmin", minSec);
10    TinyMATWriter_writeValue(mat, "xmax", maxSec);
11
12    TinyMATWriter_writeMatrix2D_colmajor(mat, "times",
13        timesMillis, campioni, 1);
14    delete[] timesMillis;

```

I still use a TinyMAT function to export a matrix number of channels per number of samples, through which the trace can be represented in EEGLab (we will see in section 4.2) :

```

1 TinyMATWriter_writeMatrix2D_rowmajor(mat, "data", values, (int)
    campioni, (int)numch);

```

Finally, I built a cell array that contained all the extracted events:

```

1 TinyMATWriter_startCellMatrix2D(mat, "Events", 3, eegLabEvents.
    size());
2
3     std::map<std::string, double> m;
4     std::map<std::string, int> kMap;
5     // campo eventi
6     for (int i = 0; i < eegLabEvents.size(); i++) {
7         TinyMATWriter_writeString(mat, "", eegLabEvents[
8             i].type);
9     }
10    for (int i = 0; i < eegLabEvents.size(); i++) {
11        TinyMATWriter_writeValue<double>(mat, "",
12            eegLabEvents[i].latency);
13    }
14    int k = 1;
15    for (int i = 0; i < eegLabEvents.size(); i++) {
16        eegLabEvents[i].urevent = k++;
17        TinyMATWriter_writeValue<int>(mat, "",
18            eegLabEvents[i].urevent);
19    }
20    TinyMATWriter_endCellArray(mat);

```

```

1      TinyMATWriter_startCellMatrix2D(mat, "Electrodes",
2      9, electrodes_channels.size());
3      for (int i = 0; i < electrodes_channels.size(); i
4      ++) {
5          TinyMATWriter_writeValue<double>(mat, "", 0);
6      }
7      for (int i = 0; i < electrodes_channels.size(); i
8      ++) {
9          TinyMATWriter_writeValue<double>(mat, "", 0);
10     }
11     for (int i = 0; i < electrodes_channels.size(); i
12     ++) {
13         TinyMATWriter_writeString(mat, "",
14         electrodes_channels[i]);
15     }
16     for (int i = 0; i < electrodes_channels.size(); i
17     ++) {
18         TinyMATWriter_writeValue<double>(mat, "",
19         electrodes[i].Longitude);
20     }
21     for (int i = 0; i < electrodes_channels.size(); i
22     ++) {
23         TinyMATWriter_writeValue<double>(mat, "",
24         electrodes[i].Latitude);
25     }
26     for (int i = 0; i < electrodes_channels.size(); i
27     ++) {
28         TinyMATWriter_writeValue<double>(mat, "", 1);
29     }
30     for (int i = 0; i < electrodes_channels.size(); i
31     ++) {
32         TinyMATWriter_writeValue<double>(mat, "",
33         electrodes[i].X);
34     }
35     for (int i = 0; i < electrodes_channels.size(); i
36     ++) {
37         TinyMATWriter_writeValue<double>(mat, "",
38         electrodes[i].Y);
39     }
40     for (int i = 0; i < electrodes_channels.size(); i
41     ++) {
42         TinyMATWriter_writeValue<double>(mat, "",
43         electrodes[i].Z);
44     }
45     TinyMATWriter_endCellArray(mat);

```

Running this code, it produces an output file in MAT format contains all the data useful for identifying an EEG tracing acquired with Micromed devices. In the next sections we will see how these data were manipulated in Matlab and I will explain how to view them with EEGlab.

4.2 Matlab code and use of EEGlab tool

In the first stage, where the export functionality has not yet been added directly to the whole software, it is necessary to import the output file obtained from the execution of the C++ program into the current folder of Matlab.

So, I structured two different scripts in Matlab; the first script handles output files with the MAT extension. With a few lines of code, I wanted to schematize the data as much as possible so that it would be readable for the end user. Below is the code:

```
%% Load file
load('output_file.mat')

%% Data preparation
eventi_struct=cell2struct(Events,{'type','latency','urevent'},2);
EEG.event=eventi_struct;

Electrodes_struct=cell2struct(Electrodes,{'theta','radius','labels',
'sph_theta','sph_phi','sph_radius','X','Y','Z'},2);
EEG.chanlocs=Electrodes_struct;

%% Save modified file
save output_file.mat
```

The most important information we can obtain from the running of this script is:

- The structure of the events (fig. 13): type, latency (location), count (urevent)
- The structure of electrodes (fig. 14): labels and different types of coordinates (spherical, Cartesian)

Fields	type	latency	urevent
1	'HfoRipple - begin'	16384	1
2	'EventB - begin'	22331	2
3	'prova'	32489	3
4	'EventB - end'	74015	4
5	'HfoRipple - end'	81920	5
6	'Trigger(0)'	105759	6
7	'PhoticFlag - begin'	450560	7
8	'PhoticFlag - end'	557056	8
9	'EventA - begin'	1021698	9
10	'Movement - begin'	1245184	10
11	'Movement - end'	1245184	11
12	'HfoGamma - begin'	1261568	12
13	'Annotation - begin'	1269760	13
14	'Annotation - end'	1269760	14
15	'EyesClosed - begin'	1294336	15
16	'EyesClosed - end'	1294336	16
17	'HfoGamma - end'	1368064	17
18	'Trigger(1)'	1384595	18
19	'EventA - end'	1399063	19

Figure 13: Example of event structure exported from TRC file to Matlab. Three column subdivision: type (event label), latency (sample on which the event is placed), urevent (index)

...	theta	radius	labels	sph_theta	sph_phi	sph_radius	X	Y	Z
1	0	0	'el001 - G2'	0	0	1	0	0	0
2	0	0	'el002 - G2'	108	90	1	0	0	0
3	0	0	'el003 - G2'	72	90	1	0	0	0
4	0	0	'el004 - G2'	130.7000	61.8000	1	0	0	0
5	0	0	'el005 - G2'	49.3000	61.8000	1	0	0	0
6	0	0	'el006 - G2'	144	90	1	0	0	0
7	0	0	'el007 - G2'	36	90	1	0	0	0
8	0	0	'el008 - G2'	90	45	1	0	0	0
9	0	0	'el009 - G2'	180	45	1	0	0	0
10	0	0	'el010 - G2'	0	45	1	0	0	0
11	0	0	'el011 - G2'	0	0	1	0	0	0
12	0	0	'el012 - G2'	229.3000	61.8000	1	0	0	0
13	0	0	'el013 - G2'	310.7000	61.8000	1	0	0	0
14	0	0	'el014 - G2'	270	45	1	0	0	0
15	0	0	'el015 - G2'	252	90	1	0	0	0
16	0	0	'el016 - G2'	288	90	1	0	0	0
17	0	0	'el017 - G2'	180	90	1	0	0	0
18	0	0	'el018 - G2'	0	90	1	0	0	0
19	0	0	'el019 - G2'	216	90	1	0	0	0
20	0	0	'el020 - G2'	324	90	1	0	0	0
21	0	0	'el021 - G2'	90	90	1	0	0	0
22	0	0	'el022 - G2'	270	90	1	0	0	0
23	0	0	'el023 - G2'	0	0	1	0	0	0
24	0	0	'el024 - G2'	0	0	1	0	0	0

Figure 14: Example of electrodes structure exported from TRC file to Matlab. The label column shows the names of the main electrode and the reference electrode (G2 in this case). In the other columns are the coordinates that allow the electrodes to be mapped.

With the second script, Settings Files (SET) are managed. In fact, the C++ program allows the user, via a JSON file (we will see next chapter how it is structured), to decide whether to save as output a file with the extension MAT or SET. The SET filename extension is primarily associated with the generic SET file type. This extension is usually assigned to files that contain various settings and preferences for operating systems, applications, games, hardware devices, etc. SET files are used by many applications, and can come in a variety of formats, both textual and binary, with the SET extension indicating the function of the file, rather than its format. SET files are loaded at the beginning, and all the data in them provide initialization, or default configuration values [10]. So, we prefer to export a SET file for displaying the TRC data using the Matlab tool EEGLAB.

EEGLAB is an interactive Matlab toolbox for processing continuous and event-related EEG, MEG and other electrophysiological data incorporating independent component analysis (ICA), time/frequency analysis, artifact rejection, event-related statistics, and several useful modes of visualization of the averaged and single-trial data. EEGLAB runs under Linux, Unix, Windows, and Mac OS X. EEGLAB provides an interactive graphic user interface (GUI) allowing users to flexibly and interactively process their high-density EEG and other dynamic brain data using ICA and/or time/frequency analysis (TFA), as well as standard averaging methods. EEGLAB also incorporates extensive tutorial and help windows, plus a command history function that eases users transition from GUI-based data exploration to building and running batch or custom data analysis scripts. EEGLAB offers a wealth of methods for visualizing and modeling event-related brain dynamics, both at the level of individual EEGLAB “datasets” and/or across a collection of datasets brought together in an EEGLAB “studysset”.

For experienced Matlab users, EEGLAB offers a structured programming environment for storing, accessing, measuring, manipulating and visualizing event-related EEG data. For creative research programmers and methods developers, EEGLAB offers an extensible, open-source platform through which they can share new methods with the world research community by publishing EEGLAB ‘plug-in’ functions that appear automatically in the EEGLAB menu of users who download them [11].

The main advantages and features of EEGLab are:

- Academic (free) software
- Running on Matlab or standalone
- Graphic user interface
- Multiformat data importing
- High-density data scrolling
- Interactive plotting functions
- Semi-automated artifact removal
- ICA & time/frequency transforms
- Event & channel location handling
- Forward/inverse head/source modeling
- Defined EEG & STUDY data structures
- Over 120 advanced plug-in/extensions

To adapt the data in Micromed's TRC file for visualization with EEGLab, it was necessary to follow the EEG data structures suggested by the tool's official website.

First, I recreated the EEG structure (fig. 15): EEGLAB variable EEG is a MATLAB structure that contains all the information about the current EEGLAB dataset. As seen above, I recreated this main structure in C++ through the use of the TinyMAT library.

To fill in the different parts of this main structure, I still followed the guideline on EEGLAB's data structure; in particular, I recreated the EEG.chanlocs (fig. 16) structure that stores the information about the EEG channel locations and channel names.

Similarly, I filled the EEG.event (fig. 17) structure that contains records of the experimental events that occurred while the data was being recorded, plus possible additional user-defined events. In general, fields type, latency, and urevent are always present in the event structure:

- type contains the event type
- latency contains the event latency in data sample unit
- urevent contains the index of the event in the original urevent table.

Other fields like position are user defined and are specific to the experiment.

The user may also define a field called duration (recognized by EEGLAB) for defining the duration of the event (if portions of the data have been deleted, the field duration is added automatically to store the duration of the break (i.e. boundary event)).

```

EEG =
  setname: 'EEG Data epochs'
  filename: 'eeglab_data_epochs_ica.set'
  filepath: '/data/matlab/eeglab/sample_data/'
  subject: ''
  group: ''
  condition: ''
  session: []
  comments: [9x769 char]
  nbchan: 32
  trials: 80
  pnts: 384
  srate: 128
  xmin: -1
  xmax: 1.9922
  times: [1x384 double]
  data: [32x384x80 single]
  icaact: [32x384x80 single]
  icawinv: [32x32 double]
  icasphere: [32x32 double]
  icaweights: [32x32 double]
  icachansind: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32]
  chanlocs: [1x32 struct]
  urchanlocs: [1x32 struct]
  chaninfo: [1x1 struct]
    ref: 'common'
  event: [1x157 struct]
  urevent: [1x154 struct]
  eventdescription: {[2x29 char] [2x63 char] [2x36 char] '' ''}
  epoch: [1x80 struct]
  epochdescription: {}
  reject: [1x1 struct]
  stats: [1x1 struct]
  specdata: []
  specicaact: []
  splinefile: []
  icasplinefile: ''
  dipfit: []
  history: ''
  saved: 'yes'
  etc: [1x1 struct]
  datfile: 'eeglab_data_epochs_ica.fdt'
  run: []

```

Figure 15: Example of EEG structure. The data should have this particular structure in order to be uploaded to EEGLAB.

```
>> EEG.chanlocs

ans =
1x32 struct array with fields:
    theta
    radius
    labels
    sph_theta
    sph_phi
    sph_radius
    X
    Y
    Z
```

Figure 16: Example of EEG.chanlocs structure. The electrodes data should have this particular structure in order to be uploaded to EEGLAB.

```
>> EEG.event

ans =
1x157 struct array with fields:
    type
    position
    latency
    urevent
    epoch
```

Figure 17: Example of EEG.event structure. The event data should have this particular structure in order to be uploaded to EEGLAB.

To handle this data, I created the following script in Matlab:

```
%% Load file .set
load('-mat','eeglab2022.0\output_PAZ1.set')

%% Data preparation
eventi_struct=cell2struct(Events,{'type','latency','urevent'},2);
EEG.event=eventi_struct;

Electrodes_struct=cell2struct(Electrodes,{'theta','radius','labels','sph_theta',
'sph_phi',
'sph_radius','X','Y','Z'},2);
EEG.chanlocs=Electrodes_struct;

%% Save modified file
save output_PAZ1.set

%% Open eeglab interface
[ALLEEG EEG CURRENTSET ALLCOM] = eeglab;
```

This script is similar to the previous one with the difference that it obviously handles SET extension files and, with the last line of code, automatically starts the EEGLab tool by opening its main interface (fig. 18).



Figure 18: Main interface of EEGLAB that is displayed when the tool is started via Matlab. The instant before the data file is loaded is shown in the figure

To get the useful information exported from the TRC file and to populate the EEG structure created in C++, we need to load the output file. To do this I click on:

File → load existing dataset → output.set → ok

as shown in fig. 19.

In this way the main information (number of channels, sampling rate, number of events) appears on the main screen (fig. 20). At the same time the EEG structure is populated with the fields we expected (fig. 21).

In EEG we will find in the Event and Chanlocs fields, structures identical to those seen in fig. 13 and fig. 14.

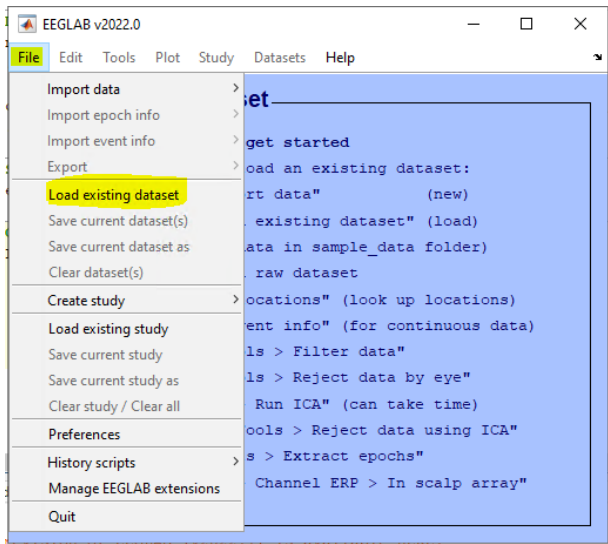


Figure 19: Uploading the SET file, obtained through the export feature, through the EEGLab interface

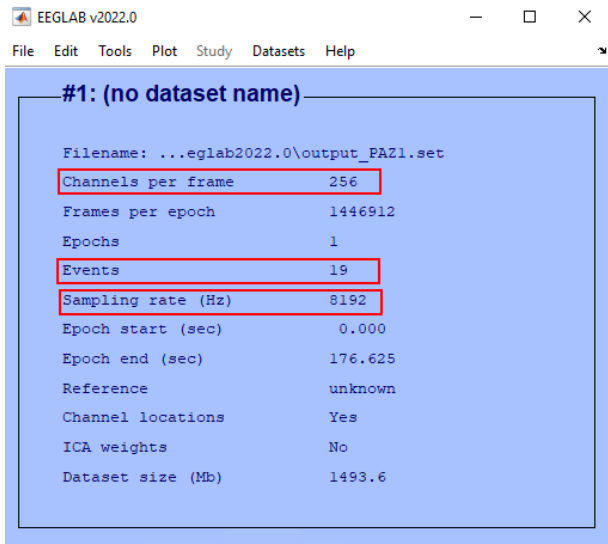


Figure 20: Main exported information that can be viewed on the EEGLAB interface: number of channels used for acquisition, number of events on the trace, sampling rate at which the trace was acquired

Field	Value
setname	''
filename	'output_PAZ1.set'
filepath	'C:\Users\am_mari\Desktop\eeeglab2...
subject	''
group	''
condition	''
session	[]
comments	''
nbchan	256
trials	1
pnts	1446912
srate	8192
xmin	0
xmax	176.6249
times	1x1446912 double
data	256x1446912 single
icaact	[]
icawinv	[]
icasphere	[]
icaweights	[]
icachansind	[]
chanlocs	256x1 struct
urchanlocs	[]
chaninfo	1x1 struct
ref	'common'
event	19x1 struct
urevent	1x1 struct
eventdescription	1x3 cell
epoch	[]
epochdescription	0x0 cell
reject	1x1 struct
stats	1x1 struct
specdata	[]
specicaact	[]
splinefile	''
icasplinefile	''
dipfit	[]
history	1x27 char
saved	'yes'
etc	1x1 struct
urchanloc	[]
...	...
spectdata	[]
run	[]
datfile	''

Figure 21: Example of a real data structure exported to Matlab with the main values of interest highlighted in red: filename (header of the output file from the Matlab plugin), nbchan (number of channels with which the trace was acquired), srate (sampling rate), data (number of channels per number of samples), event (number of events on the trace)

Chapter 5

Testing Stage

Tests on the Matlab Plugin were performed in Micromed's testing laboratory. The main steps followed are:

- Preparing a DLL (Dynamic Link Library) for the execution of C++ code and drafting a user guide;
- Drafting the official test activity document (test report) and defining the conditions under which acquire the EEG traces being exported;
- Preparation of Micromed instrumentation useful for acquisition;
- Preparation of an Excel spreadsheet in which to collect the results.

5.1 DLL and User Guide

In testing phase, it was not possible to physically add the entire BQ suite, therefore a dynamic link library (DLL) was made to simulate the Export function. A DLL is a library that contains code and data that can be used by more than one program at the same time. Any program can use the functionality contained in this DLL to implement a dialog box. This promotes code reuse and efficient use of memory.

Then a folder (fig. 22) was created containing the useful programs for conducting the tests; this folder, along with the user guide, was also handed over for validation to an external agency as we will see later.

In addition to the executable file, which precisely executes C++ commands, inside the folder is also contained a file with a JSON (JavaScript Object Notation) extension, which is a format suitable for data interchange between client/server applications. As can be seen in the guide, through this file the user can enter the path of the TRC and EVT files he wants to export to matlab and in addition he can decide where to save the output file but also with which extension to save it (MAT or SET) (fig. 23).

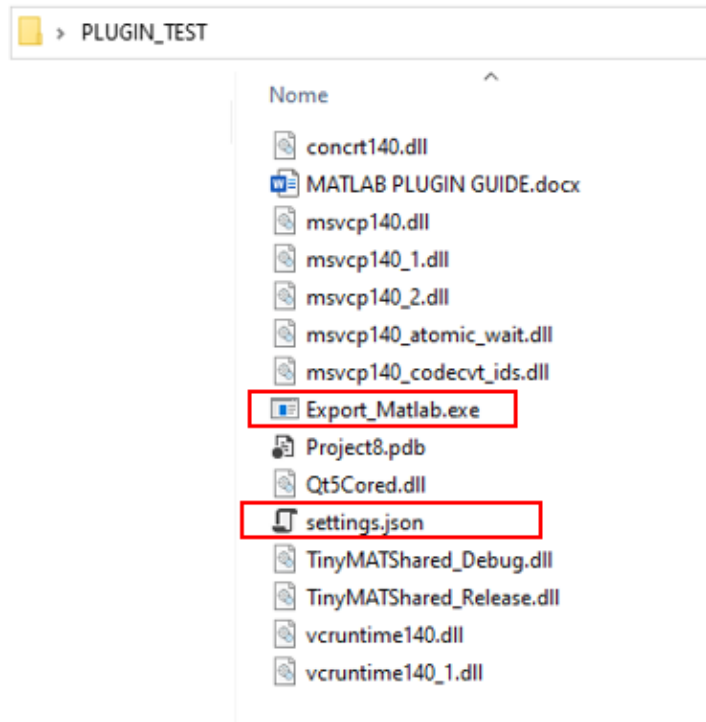


Figure 22: Folder for testing the Matlab plugin containing, in particular, a JSON file to define the paths to the different files and an .exe file to simulate the export activity of the Matlab plugin

```

1 {
2   "trcPath": "C:\\Users\\am_mari\\Desktop\\PAZIENTI TEST\\EEG_xxx.TRC",
3   "evtPath": "C:\\Users\\am_mari\\Desktop\\PAZIENTI TEST\\EXT_xxx.evt",
4   "outputPath": "C:\\Users\\am_mari\\Desktop\\FILE DI OUTPUT\\output.set",
5   "startSample": 0,
6   "lengthSamples": -1
7 }
8

```

Figure 23: Contents of the JSON file: the user can insert the path to the TRC and Evt file he wants to export; he can also decide where to save the output file by entering the path and with which extension (.mat or .set) to save it

As anticipated, in order to be able to perform the tests through third parties as well, a guide for using the plugin has been created. Below are the two main parts of this guide (fig. 24, fig. 25):

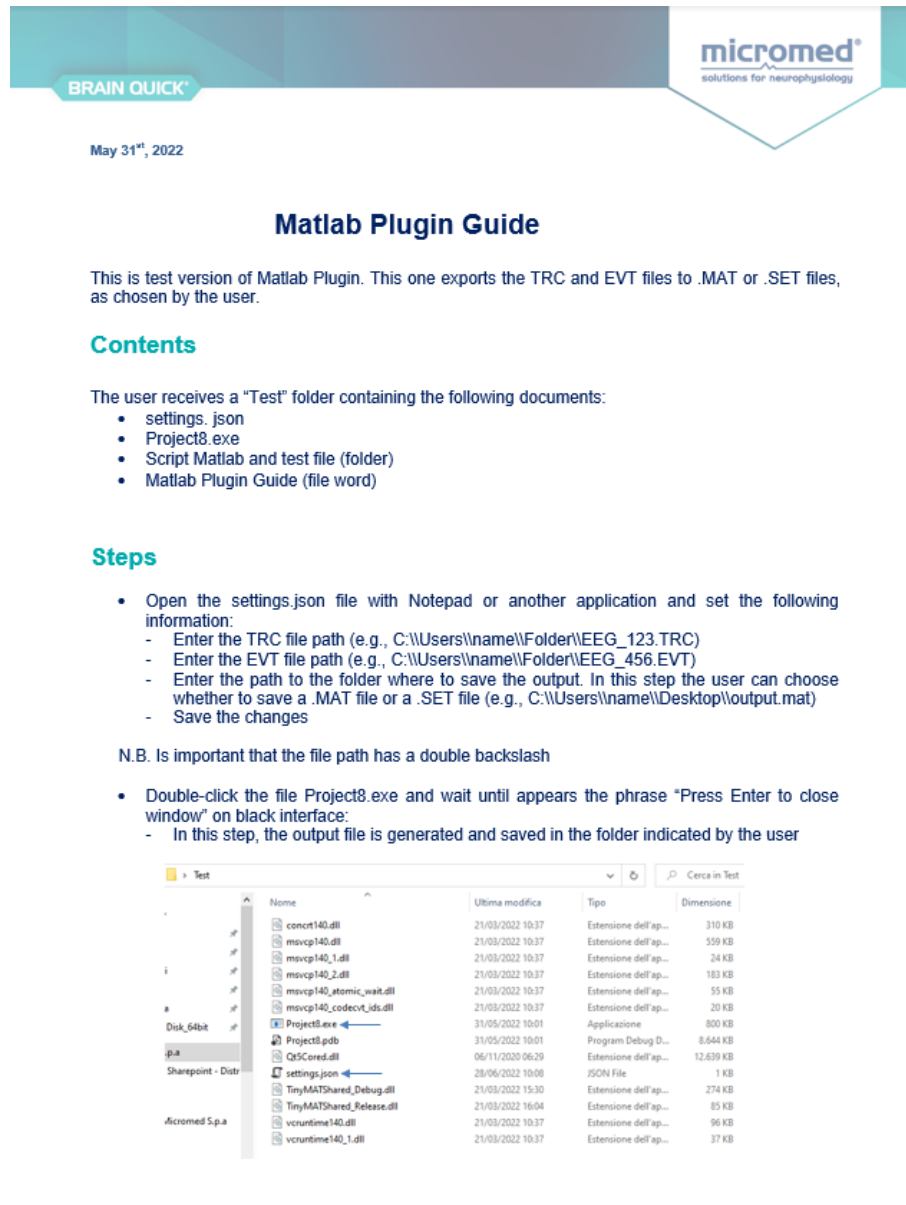


Figure 24: Matlab plugin guide: part 1

- Open Matlab and create folder that contains the following information:
 - Script Matlab (data preparation)
 - Eeglab plugin (If you want to use the interface)
 - Output file
- At this point there are two options:
 1. If you exported a .set file, then you will have to open it only through the eeglab interface; in this case:
 - Run the script Matlab (Script_for_set_file.m) for preparing the data and open the interface
 - On eeglab interface click: File → Load existing dataset → Open "output.set" file
 - You are ready for any processing through interface: plot → Channel data(scroll) or Edit → Channel location
 2. If you exported a .mat file, then you will have to open it only through the Workspace in Matlab; in this case:
 - Run the script Matlab for preparing the data structure (easily readable)
 - Double-click the output.mat and visualize the data on workspace

N.B. In both cases, the EEG structure contains all data information after running the script Matlab

Figure 25: Matlab plugin guide: part 2

5.2 Test report with definition of acquisition conditions

In order to perform tests in the business environment, appropriate documentation must be drawn up. For this reason, a report was created before the testing phase. First, “test environment” was defined, that consist in:

- Operating System: Windows 10 Pro
- Software:
 - Matlab_plugin.exe
 - MATLAB / EEGLAB
 - BRAIN QUICK

A detailed test plan was then written:

1. Export function functionality (fig. 26)
2. Accuracy of export function for Events (fig. 27)
3. Accuracy of conversion of EEG files recorder with different parameters (fig. 28)
4. Correct display of exported data in EEGLAB (fig. 29)

Step	Description	Expected Result	Result
1	- Open settings. json	- User shall be able to open the file correctly	
2	- Follow the instruction and Set file path - Choose whether to save a .MAT file or a .SET file (It is recommended to save the file in the current folder of Matlab) - Click on Save and then on Exit	- User shall be able to insert the file path in the correct fields and shall decide where save the output file - Click on Save and then on Exit	
3	- Execute the project8.exe	- The Project8.exe program shall run correctly and has no errors	
4	- Creation of new output file	- The output file shall be generated and saved in folder indicated by the user	
5	- Open Matlab and verify that output file saved in the current folder of Matlab	- User shall open Matlab ad visualize in current folder previously created output file	
6	- Prepare the exported data in Matlab by running the appropriate script	- User shall run one of the provided scripts in Matlab and shall view data in workspace or in eeglab	

Figure 26: Export function functionality Test Case

Step	Description	Expected Result	Result
1	- The user shall be sure to do the test case " Export function functionality Test Case " first.		
2	- Verify the correct export of electrode information	- The software shall identify the stored electrodes, their order and number. It shall be possible to view the electrodes registered and associated with the correct channel	
3	- Verify the correct export of events information	- The software shall show in the correct position events entered in recording and review with Brain Quick. The software shall show also all events contained in the EVT files.	
4	- Verify the correct export of notes information	- The software shall show in the correct position the notes included in the recording and review steps.	
5	- Verify the correct export of flags information	- The software shall show in the correct position the red, blue and green flags placed during recording and review.	
6	- Verify the correct export of digital trigger information	- The software shall show in the correct position the digital triggers placed during recording and review.	

Figure 27: Accuracy of export function for Events Test Case

Step	Description	Expected Result	Result
1	- Verify the correct export of EEG file recording up to 32 channels	- The software shall convert recorded EEG files up to 32 channels.	
2	- Repeat the previous step with all possible channel numbers (32 to 256)	- The software shall convert recorded EEG files with a number of channels from 32 to 256.	
3	- Verify the correct export of EEG file recording up to 256 Hz	- The software shall convert EEG files recorded up to 256 Hz.	
4	- Repeat the previous step with all possible rate sampling (64Hz, 128Hz, 256Hz, 16KHz)	- The software shall convert recorded EEG files up to 16KHz.	
5	- Verify the correct export of EEG trace with several overlapping signals (sine wave, square wave); parameters: 1 Hz – sampling rate, 1 mV – amplitude	- The software shall convert recorded EEG trace with overlapping signals.	

Figure 28: Accuracy of conversion of EEG files recorder with different parameters Test Case

Step	Description	Expected Result	Result
1	- Save the output file in current folder on Matlab and, in the same folder, save EEGLAB plugin	-	
2	- Run "Script_for_set_file.m" contained in current folder	- The software shall prepare the data for eeglab interface and open that interface	
3	- Load output file on eeglab interface with several command: Load → Existing data set → output.set	- User view most important information on eeglab interface (filename, number of channels, number of events, Sampling rate)	

Figure 29: Correct display of exported data in EEGLAB Test Case

5.3 Micromed equipment for the acquisition of EEG traces

In order to acquire EEG traces in the test environment, the Micromed proprietary LTM 32/64 PLUS SD headbox (fig. 30) was used.

SD LTM 32/64 PLUS amplifier is a portable device for acquisition of electroencephalographic signals. It is intended to be used in the diagnosis of neurological diseases characterized by episodic alteration of EEG parameters or evoked potentials evaluation. The device is particularly suitable for prolonged analysis (Long-Term EEG Monitoring). The device performs the typical EEG amplifiers and recorders function, the acquisition of bioelectrical signals. The signals are amplified, converted in digital form, formatted by a programmable logic device and transferred to the microcontroller. This component stores them on a RAM memory and afterward at defined times in the internal SDHC (Secure Digital High Capacity) memory. The acquired signals are directly shown on the headbox display. The PC is used to manage, via the software, the storage and the review of the data transmitted on-line or off-line from the memory support. SD LTM 32/64 PLUS models are specifically intended to be used both as amplifier and as ambulatory recorder; they have the possibility of working mode completely autonomous from the PC also for the start of the acquisition, thanks to expanded setting features included in the menus, SD LTM 32/64 PLUS is endowed of 8 differential channels and 24/56 common reference channels. The differential channels can be used through electronic switches as common reference channels. The device is endowed also of 2 DC channels and 3 channels for data coming from the external oximeter. Up to four SD LTM 64 PLUS can be used in a synchronized mode (fig. 31), allowing the acquisition of up to 256 channels at the same time.

On the front of the recorder, the LCD display allows the view of the menu for the device setting. Moreover, this display supplies indications on recording mode, acquisition time, total memory availability, impedance measure and allows a rough view of acquired traces, in real time or in review.

Communication with the PC was via the BQ USB PLUS (fig. 32) interface which is connected to the PC.

The device can be used as a data source for the Micromed acquisition and review software (SystemPlus EVOLUTION). The SW interface module with the recorder manages both the acquisition of data in real time and the reading of the data contained in the memory support. The acquired data will be then saved in the standard Micromed EEG format and displayed during the direct review via the review functions included in the EEG part of the software. So, after properly connecting the headbox and interface to the PC following the company guidelines (fig. 33) to simulate a real acquisition situation, software has been prepared [12].



Figure 30: Example of SD LTM 64 PLUS Headbox: portable device for acquisition of electroencephalographic signal



Figure 31: Example of synchronization of four headboxes to obtain up to 256 active channels for EEG acquisition



Figure 32: Example of BQ USB PLUS interface to connect headbox to PC for EEG trace acquisition

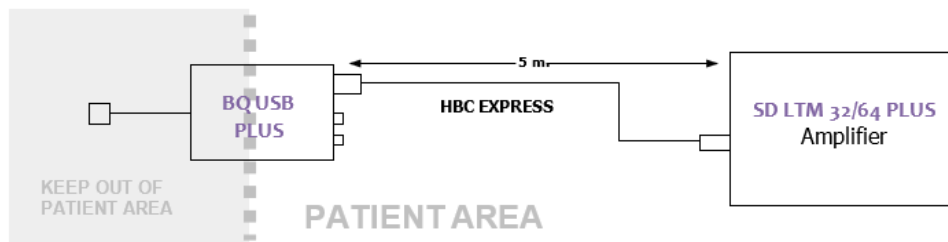


Figure 33: Connection diagram of the several parts: for correct acquisition of EEG tracing with Micromed devices, it is necessary to follow the connection scheme described in the figure

Regarding the SystemPLUS Evolution software that underlies the acquisitions with BrainQuick, first, a new resource was created in which to collect data. Resources are workspaces that identify the environment in which the data is located; specifically, resources define where the data is physically located and the referenced DB [13]. In this case, a local-type resource pointing to a specific folder on the PC was used (fig. 34).

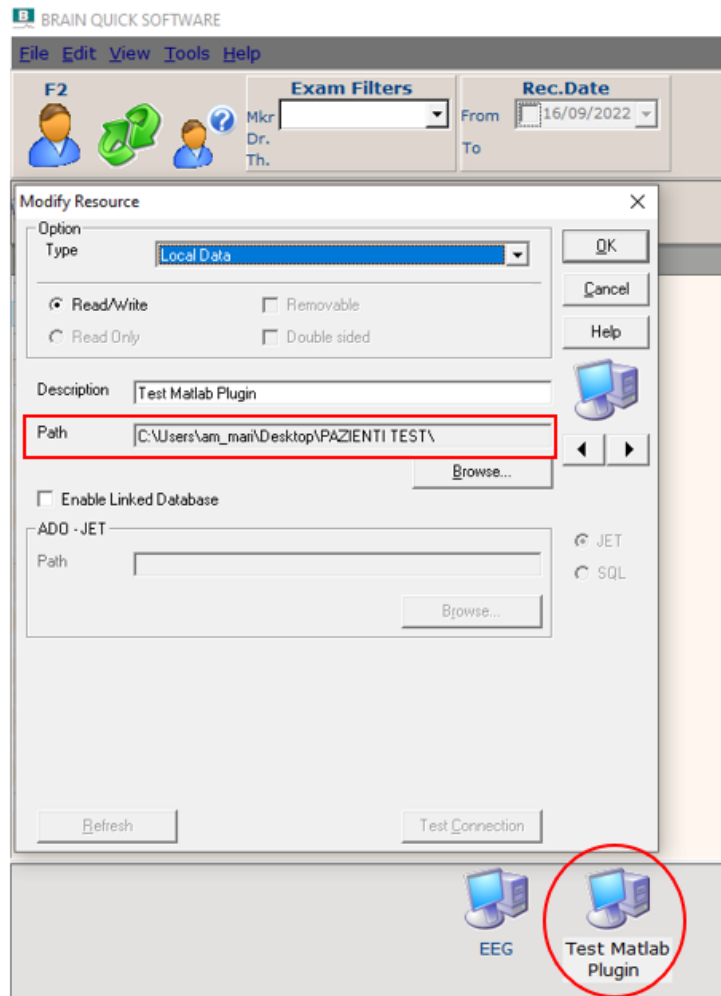


Figure 34: Creating local resource “Test Matlab plugin” containing all EEG traces acquired with Micromed equipment

Then new patients were created and EEG traces were acquired by setting the different conditions defined beforehand. In this way, the local resource in Brain Quick was populated. In addition, EEG traces of real patients recorded during demonstration sessions by Micromed staff were also imported fig. 35.

At this point the export of the resulting traces via the Matlab Plugin began following the instructions in the previous paragraphs.

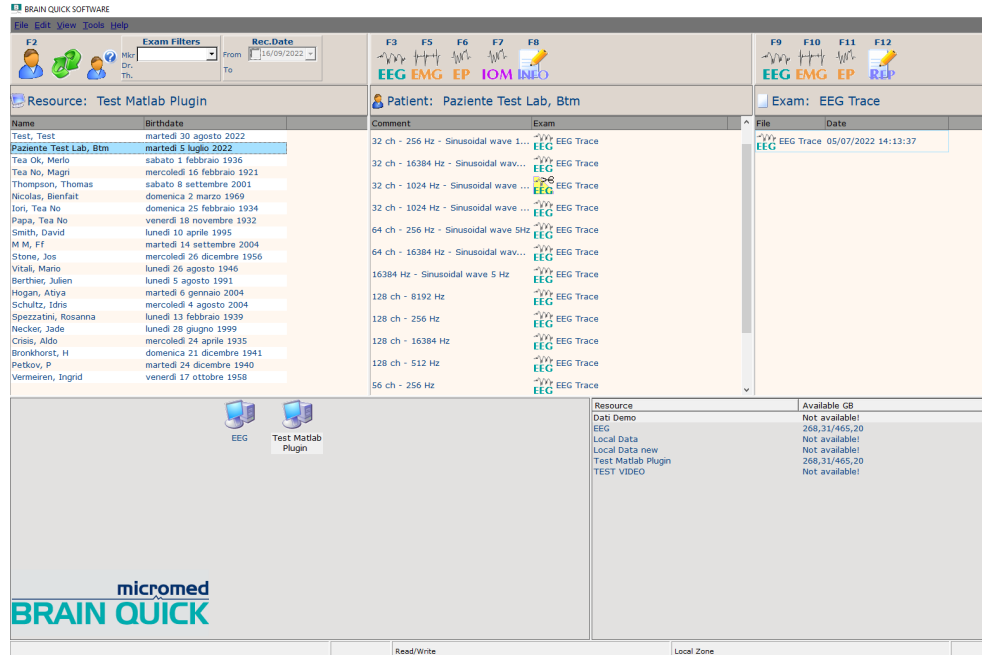


Figure 35: EEG traces acquisition environment. The main screen of the acquisition software is shown in the figure. In the first column on the left is the list of patients; for each patient, the middle column shows the list of exams performed, and for each exam, the right column shows the corresponding files

5.4 Test Performance and Results

To report the testing activity and collect data efficiently, an Excel file was created. As an initial step, was verified that the main values of interest were optimally exported to Matlab. The focus was on:

- Number of channels with which the EEG tracing was acquired
- Number of events posted on the trace during acquisition or review
- Sampling rate value used to acquire the EEG trace

From an initial visual survey it is clear that export is taking place with 100% efficiency, this is then confirmed by an analytical investigation showing in the following graphs (fig. 36, fig. 37. fig. 38).

Then the focus was on the export of events. In BrainQuick the list of events posted to the plot is returned with the corresponding location in the format hh:mm:ss.SSS (fig. 39). So it was important to verify that also in Matlab the exported events were of the same type, i.e., had the same label and were placed at the right time instant. As the data extrapolated from the TRC were structured, latency was reported in Matlab, i.e., the sample at which the event was placed and not the time instant.

Then, by adding lines of code in Matlab, the time instant was obtained. Nevertheless, the first tests failed (fig. 40, fig. 41) because Matlab was inclined to round milliseconds. To solve this problem, The Matlab code has been modified by adding the correct format for the time instant:

```
n=length(eventi_struct);
lat_millis=zeros(length(eventi_struct),1);
for i=1:length(eventi_struct)
lat_millis(i)=1000.0*eventi_struct(i).latency/EEG.srate;
end

t=datetime(2022,07,06,15,07,44,000);
t.Format='uuuu/MM/dd HH:mm:ss.SSS';

for i=1:length(eventi_struct)
    position(i) = t + milliseconds(lat_millis(i));
end
```

In this way, an export of events and positions was obtained that was perfectly congruent with Brainquick, as can be seen from the intuitive tables obtained in Excel (fig. 42).

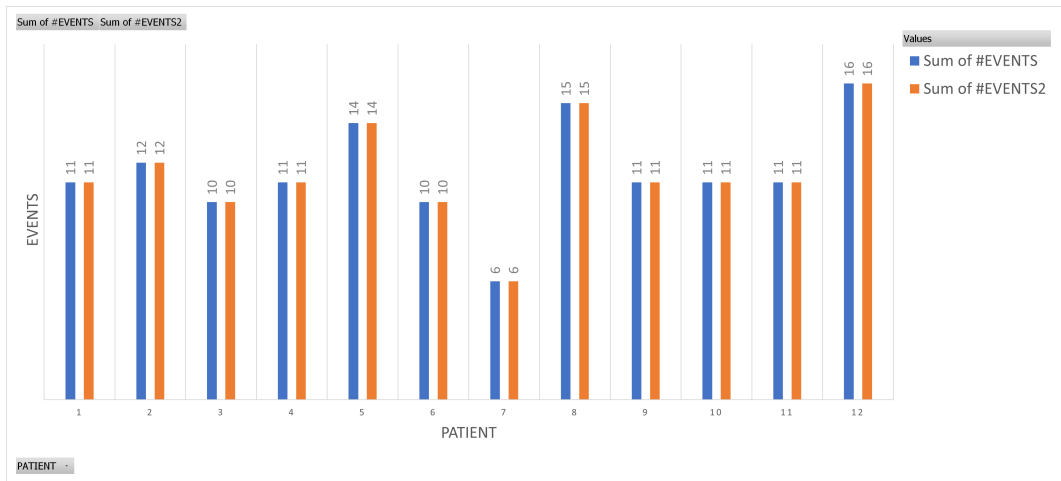


Figure 36: Comparison between the number of BrainQuick events (blue) and the number of Matlab events exported (orange)

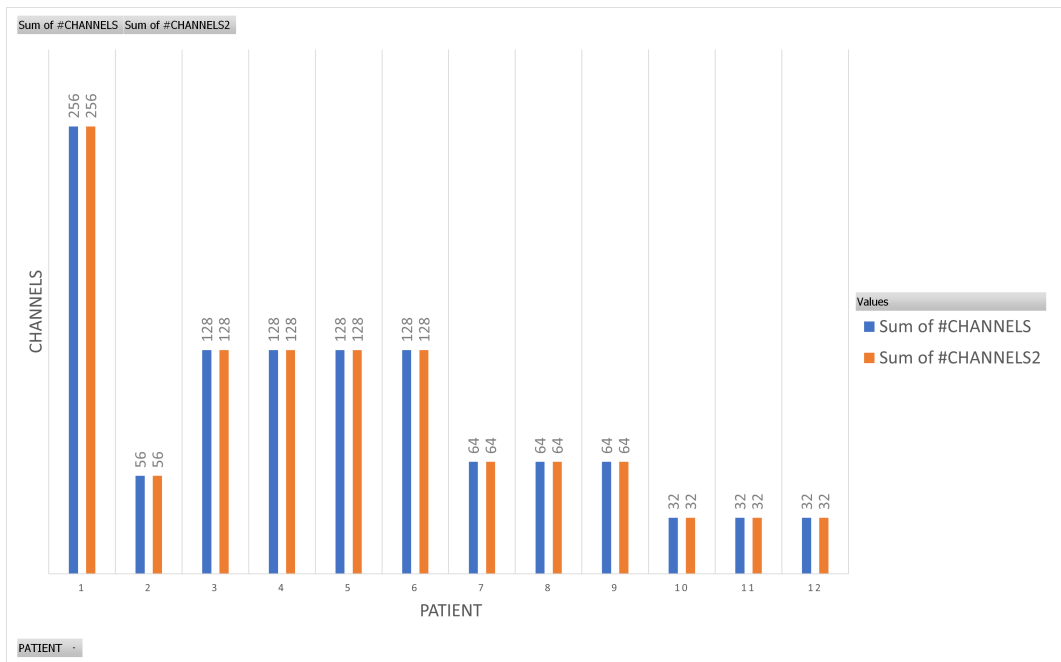


Figure 37: Comparison between the number of BrainQuick channels (blue) and the number of Matlab channels exported (orange)

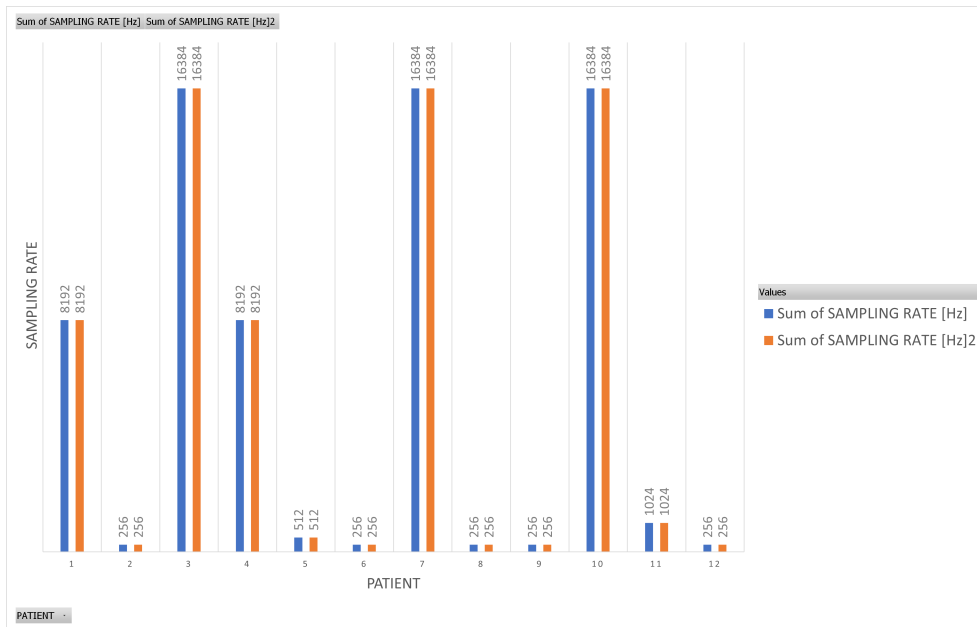


Figure 38: Comparison of BrainQuick's sampling rate (blue) and that exported to Matlab (orange)


















Events (16 objects)			Search 
Type	Text	Position	
 EEG selection		14:13:38.445	
 HFO ripple		14:16:26.679	
 Note	stimulation	14:16:32.546	
 Hyperventilation		14:16:49.242	
 Digital trigger	7	14:16:53.726	
 Eyes open		14:17:06.066	
 Eyes closed		14:17:06.480	
 Eyes open		14:17:07.668	
 Eyes closed		14:17:08.209	
 Note	eyes opened	14:17:26.656	
 HFO fast ripple		14:17:29.941	
 Digital trigger	8	14:17:38.574	
 Note	eyes closed	14:17:54.574	
 Video selection		14:18:19.167	
 HFO spike		14:18:52.370	
 Note	photic	14:19:07.562	

Figure 39: Example of event list in BrainQuick: for each event is shown the type (label), text (comment) and location (data time)

TEST T1				
BRAIN QUICK		MATLAB		DIFF
EVENT	POSITION	EVENT	POSITION	POSITION
HFO ripple	15:07:46.436	HFO ripple	15:07:46.000	FAILED
EEG selection	15:07:46.725	EEG selection	15:07:46	FAILED
Nota (prova)	15:07:47.965	Nota (prova)	15:07:47	FAILED
Digital trigger	15:07:56.910	Digital trigger	15:07:56	FAILED
Photic selection	15:08:39.036	Photic selection	15:08:39	FAILED
Hyperventilation	15:09:48.718	Hyperventilation	15:09:48	FAILED
Movement	15:10:16.862	Movement	15:10:16	FAILED
HFO gamma	15:10:18.010	HFO gamma	15:10:18	FAILED
Annotation	15:10:19.194	Annotation	15:10:19	FAILED
Eyes closed	15:10:22.696	Eyes closed	15:10:22	FAILED
Digital trigger	15:10:33.017	Digital trigger	15:10:33	FAILED

Count of POSITION3

T1



Total

POSITION3

Figure 40: Test 1 failed as a result of millisecond rounding in Matlab. Comparison of the data from the two software showed a mismatch in milliseconds

TEST T2				
BRAIN QUICK		MATLAB		DIFF
EVENT	POSITION	EVENT	POSITION	POSITION
Video selection	14:56:09.882	Video selection	14:56:09.000	FAILED
Digital trigger	14:56:15	Digital trigger	14:56:15	VERIFIED
Photic selection	14:57:39.578	Photic selection	14:57:39	FAILED
Annotation	14:57:39.728	Annotation	14:57:39	FAILED
HFO ripple	14:59:03	HFO ripple	14:59:03	VERIFIED
Analysis selection	14:59:55.716	Analysis selection	14:59:55	FAILED
Annotation	15:00:57.518	Annotation	15:00:57	FAILED
EEG selection	15:01:35.183	EEG selection	15:01:35	FAILED
HFO gamma	15:03:13.525	HFO gamma	15:03:13	FAILED
Note	15:04:08.480	Note	15:04:08	FAILED
HFO fast ripple	15:04:35.623	HFO fast ripple	15:04:35	FAILED
Digital trigger	15:04:40.777	Digital trigger	15:04:40	FAILED

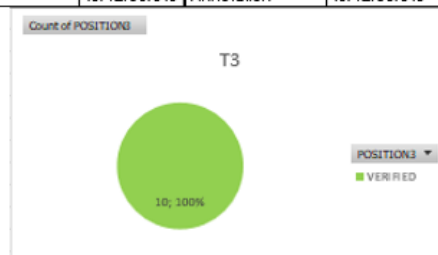
Count of POSITIONS

T2



Figure 41: Test 2 failed as a result of millisecond rounding in Matlab. During this test it became clear that the problem was the rounding of milliseconds in Matlab

TEST T3				
BRAIN QUICK		MATLAB		DIFF
EVENT	POSITION	EVENT	POSITION	POSITION
Video selection	16:41:28.473	Video selection	16:41:28.473	VERIFIED
Pain stimulus	16:41:23.680	Pain stimulus	16:41:23.680	VERIFIED
Note	16:42:41.273	Note	16:42:41.273	VERIFIED
Hyperventilation	16:42:21.593	Hyperventilation	16:42:21.593	VERIFIED
Hyperventilation	16:40:17.348	Hyperventilation	16:40:17.348	VERIFIED
HFO ripple	16:41:48.507	HFO ripple	16:41:48.507	VERIFIED
HFO gamma	16:40:33.177	HFO gamma	16:40:33.177	VERIFIED
Eyes closed	16:41:31.980	Eyes closed	16:41:31.980	VERIFIED
Artefact	16:40:38.560	Artefact	16:40:38.560	VERIFIED
Annotation	16:42:59.018	Annotation	16:42:59.018	VERIFIED



TEST T4				
BRAIN QUICK		MATLAB		DIFF
EVENT	POSITION	EVENT	POSITION	POSITION
HFO ripple	15:07:46.436	HFO ripple	15:07:46.436	VERIFIED
EEG selection	15:07:46.725	EEG selection	15:07:46.725	VERIFIED
Digital trigger	15:07:56.910	Digital trigger	15:07:56.910	VERIFIED
Photic selection	15:08:39.036	Photic selection	15:08:39.036	VERIFIED
Hyperventilation	15:09:48.718	Hyperventilation	15:09:48.718	VERIFIED
Movement	15:10:16.862	Movement	15:10:16.862	VERIFIED
HFO gamma	15:10:18.010	HFO gamma	15:10:18.010	VERIFIED
Annotation	15:10:19.194	Annotation	15:10:19.194	VERIFIED
Eyes closed	15:10:22.696	Eyes closed	15:10:22.696	VERIFIED
Digital trigger	15:10:33.017	Digital trigger	15:10:33.017	VERIFIED
Note	15:07:47.965	Note	15:07:47.965	VERIFIED

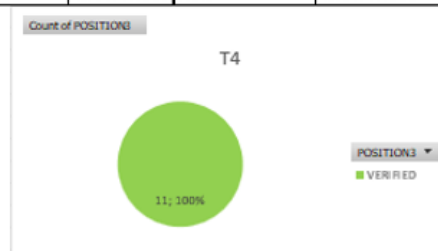


Figure 42: Examples of successful tests. The milliseconds are rounded correctly and thus the position (time) corresponds to the position in Brain Quick

Chapter 6

Validation and Future Developments

The validation phase was handled by Professor Stefano Seri of Birmingham Children's Hospital, Micromed's partner for many years. Professor Seri is a Doctor of Medicine and Surgery specializing in Developmental Neuropsychiatry. His medical practice focuses on the research and treatment of epileptic events in pediatric patients.

The possibility of manipulating data and identifying events easily through the export to Matlab that I created allows him to efficiently view, edit and archive his patients' trace data as he also has the ability to export even small parts of Long Term Monitoring.

In addition, by exporting to Matlab he was able to do quick analysis and draw up a trend of seizure events of patients whose large numbers of tracings had been acquired. In fact, through the plugin and to the availability of events and their positions over time in tables, it is possible to perform simple statistics of relevant events of a patient at different times of hospitalization.

Dr. Seri not only gave us positive feedback regarding the usability of the Matlab Plugin but also put attention on what could be future developments.

The main suggestions concern the following points:

- Implementation of export of data obtained by intracranial stimulation;
- Ability to create groups of montages and visualize them in Matlab;
- Obtaining correct date and time format of events without using scripts in Matlab;
- Obtaining correct date and time format of events without using scripts in Matlab;
- Reduce export times, which to date are estimated to be about 3 minutes for monitoring of a few hours and about 10 minutes for Long Term Monitoring.

6.1 Conclusion

Micromed is among the few manufacturers in the area of electroencephalography to collaborate with researchers around the world to help advance the neuroscience and neurophysiology fields.

The main focus of this thesis work was to create a Plugin that would make data formats acquired with Micromed systems fully compatible and viewable in the Matlab environment and through the EEGLAB tool to support research.

During the internship activity in Micromed, which lasted about five months, the planned goal was achieved. Through the implementation of code in C++ and the use of specific libraries, it was possible to export files containing EEG traces recorded with Micromed devices with 100% efficiency.

An intensively testing phase provided evidence that the export was a useful tool for clinicians and researchers to efficiently visualize data by creating statistics and trends of epileptic events.

Validation at an accredited center such as Birmingham Hospital confirmed that the possibility of viewing data in EEGLAB will facilitate analysis activities on large groups of patients and exams.

The Matlab Plugin created, will enrich the new Brain Quick Software to be released by Micromed's research and development team in the next few months.

Bibliography

- [1] Theoretical-Practical Manual of Electroencephalography.
Oriano Mecarelli - Department of Neurological Sciences Sapienza University of Rome Policlinico Umberto I, Rome
- [2] American Clinical Neurophysiology Society. Guideline 5: Guidelines for standard electrode position nomenclature. *J Clin Neurophysiol* 2006;23(2):107-110
- [3] Product Development Protocol - Private Documentation by Micromed
- [4] <https://docs.microsoft.com/it-it/cpp/ide/using-the-visual-studio-ide-for-cpp-desktop-development?view=msvc-170> - Microsoft's official website
- [5] TYPE "4" - EEG FILE STRUCTURE DESCRIPTION - Private Documentation by Micromed
- [6] BRAIN QUICK INTERFACE FOR EXTERNAL PROGRAM - Private Documentation by Micromed
- [7] <https://support.microsoft.com/it-it/office/informazioni-di-base-su-xml-a87d234d-4c2e-4409-9cbc-45e4eb857d44> - Microsoft's official website
- [8] <https://github.com/jkriege2/TinyMAT>
- [9] MAT-File Format/www.mathworks.com
- [10] www.filetypeadvisor.com/it/extension/set - General guide on file formats
- [11] <https://scn.ucsd.edu/eeglab/index.php> - Swartz Center for Computational Neuroscience
- [12] SD LTM 32/64 PLUS User and Technical Manual - Private Documentation by Micromed
- [13] SystemPlus EVOLUTION Advanced Installation Guide - Private Documentation by Micromed

Ringraziamenti

Nelle prossime poche righe vorrei rivolgere i miei più sentiti ringraziamenti alle persone che, in tanti modi diversi, hanno contribuito a rendere questo periodo speciale.

Al Professor Marco Castellaro per essere stato sin da subito disponibile ed avermi guidato in questo percorso che, per me, è stato di grande crescita personale e professionale.

A Cristiano Rizzo, Nicola Rizzo, Raffaele Orsato per avermi dato l'opportunità di vivere questa esperienza in Micromed.

A Giulia Pellegrino per aver fatto il tifo per me sin dal primo giorno: "Giulia, dopo aver conosciuto te e tutte le donne di Micromed ho imparato che possiamo essere davvero una forza della natura".

Ad Alberto Pellizzon per aver reso il mondo della programmazione più leggero per me, per avermi aiutata senza sosta e per aver condiviso attimi in "delirio di onnipotenza" ma anche con la "sindrome dell'impostore". Per la tua gentilezza e per le risate dopo una improvvisata lezione di napoletano: "Grazie Albe!".

A tutto il team di Ricerca e Sviluppo, per avermi fatto sentire sin da subito parte del gruppo ed aver reso divertenti anche le giornate più uggiose.

A Silvia Bellio, Raffaele Giordano, Massimo Tarolli, Garay Pashayev per essere per me un reale "supporto" nella vita di tutti i giorni: "mi avete insegnato cosa vuol dire essere un team e che, soprattutto, è possibile in questo mondo spesso così ostile, essere delle anime gentili".

A tutti voi, grazie !