UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA

POSTGRADUATE PROGRAM IN ELECTRICAL ENGINEERING

PEDRO TENDOLIN DE LIMA COSTA

Design of an Electrooculography-based interface for people with severe motor disabilities

São Paulo

2024

PEDRO TENDOLIN DE LIMA COSTA

Design of an Electrooculography-based interface for people with severe motor disabilities

Corrected Version

(The original version can be found on the unit that hosts the Postgraduate Program)

Dissertation presented to the Escola Politécnica of the Universidade de São Paulo for a Master of Science degree from the Electrical Engineering Postgraduate Program

Research Area: Biomedical Engineering

Advisor: Prof. Dr. Idágene Aparecida Cestari

São Paulo
2024

Catalogação-na-publicação

Costa, Pedro Tendolin de Lima
Design of an Electrooculography-based interface for people with severe motor disabilities / P. T. L. Costa -- versão corr. -- São Paulo, 2024.
130 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Telecomunicações e Controle.

1.PESSOAS COM DEFICIÊNCIA FÍSICA 2.SISTEMA DE COMUNICAÇÃO 3.ELETROCULOGRAFIA 4.INTERFACE HOMEM-COMPUTADOR 5.TECNOLOGIAS DA SAÚDE I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Telecomunicações e Controle II.t.

Nome: COSTA, Pedro Tendolin de Lima

Título: Design of an Electrooculography-based interface for people with severe motor disabilities

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Ciências.

Aprovado em: 27/02/2024

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to all those who have contributed, encouraged, and advised me since the beginning of this journey. It has been both challenging and rewarding, but I am certain that I would not have accomplished it without all the support.

A special acknowledgment goes to my advisor, Prof. Dr. Idágene Aparecida Cestari, who guided me during this journey, encouraging me to strive for excellence. I want to thank her patience, dedication, insights, and especially, the mentorship offered during these years.

I am also grateful to the members of my dissertation committee for their valuable feedback, critical reviews, and suggestions, which have contributed significantly to this dissertation.

I would also like to acknowledge the Lucy Montoro Rehabilitation Institute, particularly Prof. Dr. Linamara Rizzo Battistella, Denise Rodrigues Tsukimoto, occupational therapist, Daniel Seiei Uehara Tamashiro, biomedical engineer, and Milton Seigui Oshiro, for all the collaboration and support, and for allowing me to carry on the clinical protocol at the institute.

I also want to thank the Heart Institute (InCor), Bioengineering Division, for providing the resources for this study, especially, Marcello Mazzetto, biomedical engineer, for the support and instructions during this study.

I thank the Escola Politécnica of the Universidade de São Paulo and their staff and professors for all the opportunities and knowledge provided.

I would like to thank Paulo Camilo, biomedical engineer, for participating in this journey toward obtaining a master’s degree by sharing his experiences and knowledge while working together in the same company.

Special thanks to Fátima Mari and M. Cristina V. Borba for reviewing this dissertation for the English language and for the knowledge they provided.

I want to extend my appreciation to all my family for their unconditional support and love. Their understanding and belief in me provided me with the strength to complete this work. I am deeply thankful to my father, Antonio, my mother, Rosa and my stepmother, Sandra, as well as my brothers, sisters, and aunts. Special thanks to my beloved partner, Bianca, for her love, patience, encouragement, and support in each decision and step during these years.

Resumo

Mais de um bilhão de indivíduos no mundo convivem com alguma forma de deficiência. Em 2010, aproximadamente 3,7 milhões de brasileiros apresentavam graus severos de deficiência motora que ocorrem por traumas físicos ou doenças tais como a esclerose lateral amiotrófica (ELA). ELA e outras doenças ou traumas físicos podem resultar em paralisias motoras severas com exceção dos movimentos dos olhos, que motivam o desenvolvimento de tecnologias assistivas controladas pelos olhos. A eletrooculografia (EOG) é uma técnica que mede biopotenciais gerados pelos movimentos oculares e que já foi aplicada no controle de cadeiras de rodas motorizadas, controle de drones e interfaces homem-máquina para fins de comunicação. Esse estudo tem como objetivo desenvolver um interface home-máquina baseada em eletro-oculografia visando oferecer um método de comunicação conveniente, eficiente e prático para pessoas com mobilidade reduzida e impossibilidade da fala. Um estudo piloto foi desenvolvido a partir de um sistema analógico de aquisição de EOG para detectar movimentos horizontais dos olhos e traduzi-los em respostas a perguntas pré-programadas mostradas em uma interface gráfica. Os resultados desse estudo inicial e discussões com os pesquisadores do Instituto de Reabilitação Lucy Montor sugeriram a viabilidade de se desenvolver um sistema de comunicação baseado em EOG acessível. Foi desenvolvido um sistema de comunicação EOG de canal único que usa apenas piscadas de olhos, em sincronia com o flash do botão alvo, para controlar uma interface gráfica de usuário (GUI). A GUI foi projetada com opções para informar suas necessidades e um teclado virtual para escrita. O software desenvolvido neste estudo executa a interface e detecta quando o usuário pisca. Dois testes online diferentes foram realizados com participantes que não apresentavam deficiência motora. Os resultados obtidos mostram que o sistema desenvolvido tem exatidão de 89,38% para detecção de piscar de olhos, 89,91% para uso do teclado virtual e 8,26 s de tempo de resposta, demonstrando o bom desempenho do sistema projetado. Resultados similares foram obtidos em quatro participantes com deficiência atendidos no Instituto Lucy Montoro com acurácia de 85,62% na detecção do piscar de olhos, 91,97% no uso do teclado virtual e tempo de resposta de 9,79 s. Os resultados mostraram que a velocidade máxima de interação dependente do intervalo entre flashes e que o sistema requer treinamento e adaptação do usuário. O sistema desenvolvido tem menor complexidade de processamento e consumo de energia comparado a outros descritos na literatura e pode ser utilizado como plataforma para o desenvolvimento de sistemas portáteis.

Palavras-chave: Eletro-oculografia (EOG), Interfaces Homem-máquina (HCI), tecnologias assistivas, piscada, Comunicação Aumentativa e Alternativa (CAA).

Abstract

Over one billion people worldwide deal with a type of disability. In 2010, approximately 3.7 million Brazilians showed severe motor impairments from physical traumas or diseases such as amyotrophic lateral sclerosis (ALS). ALS and other diseases or physical traumas can cause severe impairment excluding the movements of the eyes, which motivates the development of assistive technologies controlled only by eye movements. Electrooculography (EOG) measures biopotential signals generated by eye movements and has been used for wheelchair control, drone control, and human-machine interfaces for communication purposes. This study aims to develop a human-machine interface based on electrooculography to provide an efficient, practical, and convenient communication method for people with mobility and speech impairments. A pilot study was conducted to develop an analog EOG communication interface based on the detection of horizontal movements of the eyes and translate them into answers to pre-programmed questions shown on a graphical interface screen. The initial results and discussions with researchers of the Instituto de Reabilitação Lucy Montoro (IRLM) suggested the feasibility of creating an affordable EOG communication device. A single-channel EOG communication system was developed that uses only eye blinks, in synchrony with the flash of the target button, to control a graphical interface. A graphical user interface (GUI) was designed with options for users to inform their needs and a virtual keyboard for writing. The software developed in this study runs the interface and detects when the user blinks. Two different online tests were conducted with participants with no disabilities, and the results showed 89.38% accuracy of blink detection, 89.91% virtual keyboard use, and 8.26 s of response time (RT), demonstrating a good performance of the system designed. The system was also evaluated at the Lucy Montoro Institute. Four participants with disabilities were included and the results were similar regarding accuracy in blink detection (85.62%), use of virtual keyboard (91.97%), and response time (9.79 s). Overall, the results show that the speed of the interactions depends on the interval between the flashes, and the system requires the user’s training and adaptation. The system designed has lower processing complexity and power requirements compared to previous studies reported in the literature and may be further developed into a wearable EOG acquisition system.

Keywords: Electrooculography (EOG), human-computer interface (HCI), assistive technology, eye blink, augmentative and alternative communication (AAC).

List of Figures

List of abbreviations and acronyms

ADC	Analog-to-Digital Converter
BCI	Brain-Computer Interface
CRPD	Convention on Rights of Persons with Disabilities
FFT	Fast Fourier Transform
GUI	Graphical User Interface
IDE	Integrated Development Environment
LSL	Lab Streaming Layer
PC	Personal Computer
SNR	Signal-to-Noise Ratio
SOA	Stimulus Onset Asynchrony
SVM	Support Vector Machine
IRLM	Instituto de Reabilitação Lucy Montoro

Table of Contents

1. INTRODUCTION

1.1 Contextualization

More than one billion people around the world are affected by some type of disability, with 200 million experiencing significant functional limitations (WORLD HEALTH ORGANISATION, 2011). In Brazil, approximately 3.7 million people were reported to have severe motor impairments in 2010 (INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA - IBGE, 2010); these impairments can impact a person's ability to interact with the world, even with perfect cognitive and social skills (USAKLI; GURKAN, 2010).

Access to technology has been enshrined in the Brazilian Constitution. In March 2007, Brazil signed the United Nations Convention on Rights of Persons with Disabilities (CRPD) in New York, which was later enacted through Decree 6,949 in August 2009. Article 9 of the CRPD requires that States Parties take appropriate measures to promote the development and distribution of accessible technologies for persons with disabilities. Following the CRPD, Federal Law 13.146 was enacted in July 2015 (PERUZZO; FLORES, 2021).

Motor neuron diseases (MND), including amyotrophic lateral sclerosis (ALS), are progressive neurodegenerative disorders that result in the loss of body functions such as speech and breathing. Degeneration caused by this disease primarily affects the cells of the nervous system, except for the eye muscles, which are typically the last to be affected. In addition to ALS, other types of diseases or physical trauma can cause severe impairments, except for eye movements and mental activity, such as the locked-in syndrome. This motivates the development of assistive technologies that are only controlled by eye movements (USAKLI et al., 2018; USAKLI; GURKAN, 2010). These assistive technologies can improve people’s functional skills and provide better life prospects by improving access to health, education, transport, and the labor market (WORLD HEALTH ORGANISATION, 2011).

The methods currently employed to track eye movements encompass a range of techniques, including the scleral search coil method, infrared oculography, electrooculography (EOG), and video-oculography (VOG) (CHANG, 2019). Among these, EOG and VOG stand out for their suitability for assistive technologies (HORI; SAKANO; SAITOH, 2006; SINGH; SINGH, 2012). VOG presents certain performance challenges related to camera quality and ambient light, requiring more advanced signal processing, EOG offers a robust technique for measuring the corneal-retinal potential (MALMIVUO; PLONSEY, 1995; SINGH; SINGH, 2012). This steady spontaneous bioelectric signal, ranging from 0.4 to 1 mV, can be harnessed to detect eye movements and blinks (MALMIVUO; PLONSEY, 1995).

EOG-based systems have been shown to be effective in providing support to people with mobility impairments. For instance, EOG has been used to control a wheelchair (BAREA et al., 2002) and navigate virtual interfaces, including keyboards and menus (USAKLI; GURKAN, 2010). Recent advances, such as those described by Li and He's single EOG channel system (HE; LI, 2017), have enabled faster and more efficient communication methods, boasting accuracy and speed.

Moreover, the integration of EOG with other control signals, such as steady-state visual-evoked potentials (SSVEP) (DURAISAMY; REDDY, 2021; SARAVANAKUMAR; RAMASUBBA REDDY, 2020) or P300 event-related potentials (ERP) (POSTELNICU; TALABA, 2013; YU et al., 2019), can enhance the accuracy and information transfer rates in hybrid brain-computer interfaces (HBCIs). Recent studies have also focused on developing lightweight, flexible, and wireless EOG systems tailored for real-life usage (DEBBARMA; BHADRA, 2021, 2022), further expanding the accessibility and comfort of assistive technologies.

In light of these advancements and challenges, research in the field of EOG-based assistive technologies remains important. By addressing technical limitations and exploring new applications, researchers can further empower individuals with disabilities and promote their participation in various aspects of life.

1.2 Objectives

The objective of this research is to develop an efficient and convenient communication method for people with mobility and speech impairments, using electrooculography (EOG) signals to control a graphical user interface (GUI).

1.2.1 Specific Objectives:

a) To review the EOG literature and its applications in communication interfaces;
b) To develop an EOG system to detect and process EOG blink signals;
c) To develop a GUI that can be accurately controlled by EOG blink signals;
d) To evaluate the performance and user experience of the EOG system with people with and without disabilities.

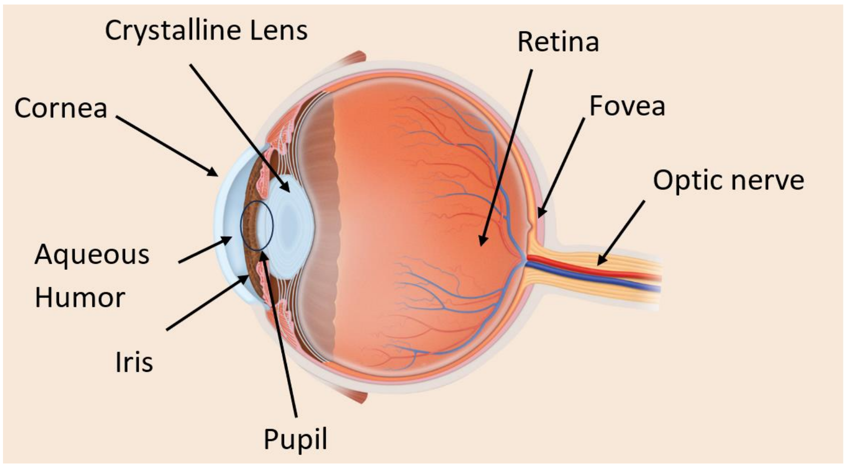
2. LITERATURE REVIEW

Different methods have been studied to improve the interactions of people with disabilities, such as those affecting movement and speech. Among these, eye movements have emerged as an option for users to communicate with their smartphones and laptops. This section reviews the anatomy and physiology of the human eyes and eye movement detection techniques, focusing on Electrooculography and its applications.

2.1 Anatomy and Physiology of the Human Eye

The human eye is an extension of the brain, which is protected by facial tissues and bones. The eyeball is spherical (Figure 1), and its outer layer, the cornea, acts as a protective barrier. Light enters the eye through an aperture called the pupil, which is adjusted by a thin muscular tissue called the iris, acting as a diaphragm. The iris can contract and expand the pupil to adjust the amount of light entering the eye. After entering the eyes through the pupil, the light passes through the crystalline lens, which, together with the cornea, adjusts the light rays to form the optical image on the retina, centered on the foveal region, where the visual accuracy is the highest. The retina then transforms the received light waves into electrical signals that are transferred to the brain (visual cortex) through the optic nerve (IRSCH; GUYTON, 2009; LUKANDER, 2003).

Figure 1- The eye structure



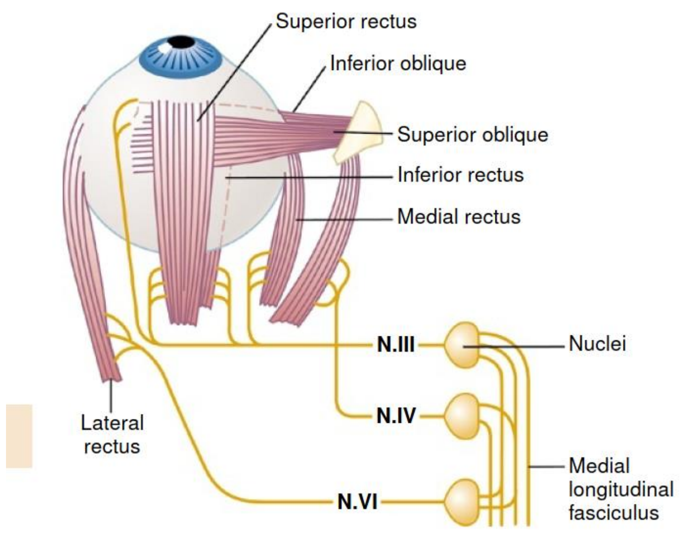
Source: Author.

The aqueous humor (Figure 1) fills the chamber that is located between the cornea and iris, as well as in the posterior chamber, which is situated between the iris and the crystalline lens. It works with the vitreous humor to fill the cavity between the crystalline lens and the retina, maintain intraocular pressure, preserve the shape of the eyeball, and supply nutrients and oxygen. The sclera, which is the external fibrous layer surrounding the eyeball, is commonly referred to as the "white" of the eye. Finally, the layer situated between the retina and sclera is called the choroid, comprising small arteries and veins forming a dense capillary plexus (IRSCH; GUYTON, 2009).

2.2 Eye Muscles

Three pairs of muscles are responsible for controlling eye movements (Figure 2): the superior and inferior oblique, inferior and superior rectus, and lateral and medial rectus. They allow the eyes to move from side to side, up and down, and rotate, controlling the horizontal and vertical motion of the eye and the gaze direction (GUYTON; HALL, 2011; LUKANDER, 2003).

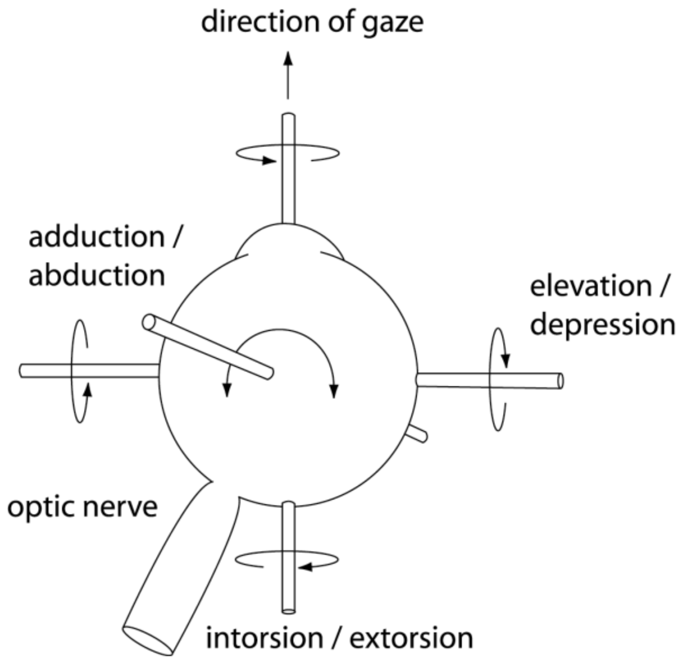
Figure 2 - The muscles of the eye



Source: Guyton and Hall (2011).

Figure 3 shows that these three sets of muscles provide six degrees of freedom for the eye. They are also reciprocally innervated, which means that when one pair contracts, the other relaxes (GUYTON; HALL, 2011). The medial and lateral rectus muscles contract to move the eyes toward or away from the nose, the superior and inferior rectus contract to move the eyes upwards and downwards, and the oblique muscles rotate, controlling the intorsion and extorsion of the eye (LUKANDER, 2003).

Figure 3 - Eye degrees of freedom



Source: Lukander (2003).

2.3 Eye Movements

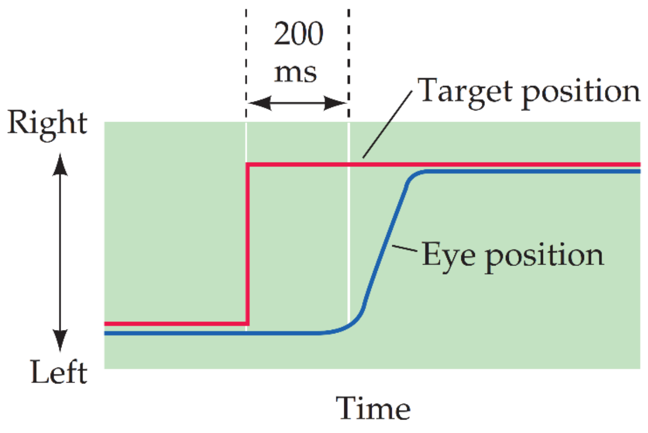
There are five main types of eye movements: saccades, fixation, pursuit, vergence, nystagmus, and vestibular-ocular system movements (GUYTON; HALL, 2011; LUKANDER, 2003). Saccadic and smooth pursuit are the movements used for human-machine interfaces, yet other types of eye movements should be understood to classify them as artifacts and remove them from electronic readings (CHANG, 2019).

2.3.1 Saccades

Saccades are either voluntary or reflexive eye movements that abruptly change the direction of gaze to a new target. They can occur very rapidly and can reach 40,000 °/s^2 of peak accelerations and 400-600 °/s of peak velocity, which vary with the saccade amplitude but are significantly superior in speed when compared to other resources of human interactions (LUKANDER, 2003; YOUNG; SHEENA, 1975).

Figure 4 illustrates the time response of the saccadic movement. After a change in the target position, there is a 200-ms delay before the eye movement starts. To focus on the next position, the eye takes 15-100 ms, which depends linearly on the distance between the previous and future targets (LUKANDER, 2003; PURVES, 2004). This type of eye movement is also called ballistic because the system that generates the saccadic movements cannot respond to subsequent changes in the target position. Consequently, the eye target must be selected before the saccade starts, and if the target moves during this time, another saccadic movement is required to correct the position (PURVES, 2004).

Figure 4 - Time response for saccadic movements



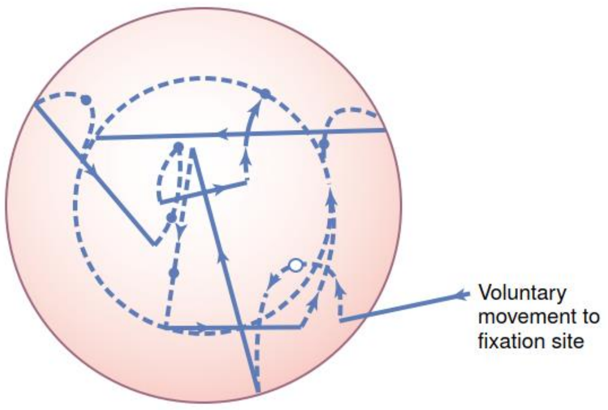
Source: Purves (2004).

2.3.2 Fixation

Fixation, which lasts between 100 and 1000 ms, is the primary method for gathering visual information (LUKANDER, 2003). Essentially, fixation involves the eyes "locking" onto the object of interest and stabilizing the image on the retina. During this process, three nearly imperceptible movements occur: continuous tremor, drift, and flicking movements. The continuous tremor refers to a distinct type of high-frequency oscillatory movement that occurs from 30 to 80 Hz, with velocities of only a few º/s. This type of movement is often considered noise in the oculomotor system (GUYTON; HALL, 2011; LUKANDER, 2003).

The drift movement is a slow, random eye motion that can move the image focus away from the foveal region, which accounts for high-resolution vision. In contrast, the flicking is a sudden reflex movement that quickly moves the image back to the center of the fovea. Figure 5 illustrates the fovea as a pink circle and shows the movement of a spot of light over the fovea. The random slow drifting (represented by dashed lines) moves the spot of light to the edges of the fovea, whereas the flicking (solid lines) attempts to maintain the image within the region of the fovea (GUYTON; HALL, 2011).

Figure 5 - Spot of light movements over the fovea during fixation



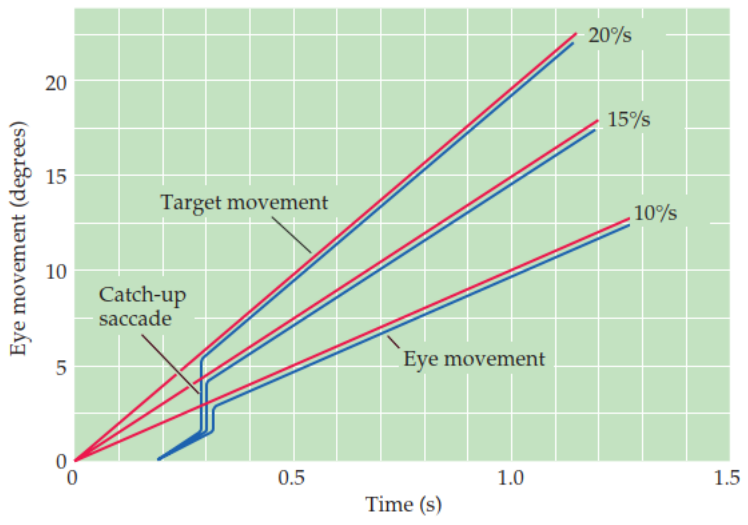
Lines with dashes indicate slow drifting movements, whereas solid lines are sudden flicking movements.

Source: Guyton and Hall (2011).

2.3.4 Pursuit Movements

The ability of the eye to keep a fixed gaze on a moving object is known as pursuit movement. This movement occurs due to a cortical mechanism that can automatically detect the movement course of an object and then move the eyes following the object trajectory, stabilizing the image of a moving target on the retina (GUYTON; HALL, 2011). This type of movement can be considered voluntary because the subject can choose whether to track an object; however, the tracking occurs through automatic subconscious ability. Figure 6 illustrates three examples of pursuit movements at different velocities (blue lines) compared to the target (object) movements in red lines (PURVES, 2004).

Figure 6 - Three types of pursuit movements (target versus eye movement)



Source: Purves (2004).

This type of movement is usually slow and can track objects moving within the range of 1 to 30 °/s (YOUNG; SHEENA, 1975). This movement is limited in velocity and acceleration; hence, the eye usually uses saccades to catch up with the moving object, maintaining the object tracking (PURVES, 2004; YOUNG; SHEENA, 1975).

2.3.5 Vergence Movements

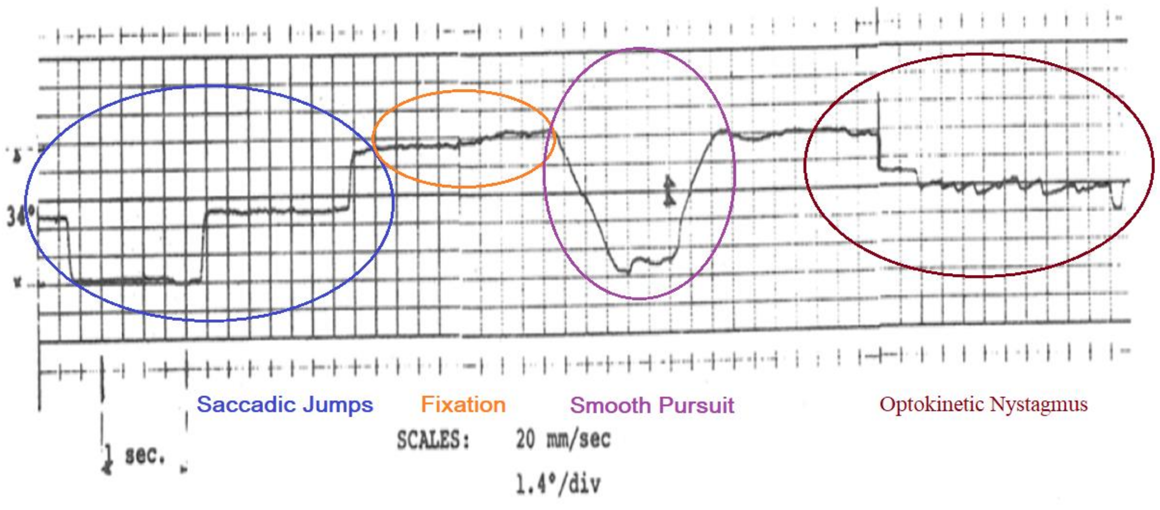
Vergence is a slow eye movement given by each eyeball moving in different directions simultaneously. It is either convergence, when an object in front of the subject is getting closer, and the eyes need to move towards each other to adjust the focus; or divergence, when the object is getting farther away, and the eyes move away from each other to adjust the focus. Contrary to saccades that can reach up to 600 °/s, vergence movements are slow, and their velocity is around 10 °/s over a range of nearly 15 ° (LUKANDER, 2003; PURVES, 2004; YOUNG; SHEENA, 1975).

2.3.6 Nystagmus

Nystagmus is a rhythmic, involuntary, and oscillating movement that can occur in one or both eyes. It can cause horizontal, vertical, or rotating movements in response to patterns in the visual field (optokinetic nystagmus) or motions of the head (vestibular nystagmus) (LUKANDER, 2003; YOUNG; SHEENA, 1975).

The appearance, amplitude, and frequency of optokinetic and vestibular nystagmus are similar, with amplitudes varying between 1 and 10 °. The former typically occurs when a target object moves out of the visual field, while the latter results from head rotation with respect to inertial space. Both nystagmus movements consist of two phases: a slow phase, which can be explained as a pursuit movement, and a fast phase, similar to a saccade. Figure 7 displays different types of eye movements recorded, including nystagmus with a sawtooth pattern characteristic (YOUNG; SHEENA, 1975).

Figure 7 - Movements of the eyes recorded using a photoelectric monitor



It shows saccadic jumps, fixation, smooth pursuit, and optokinetic nystagmus.

Source: Adapted from Young and Sheena (1975).

2.4 Excitable Cells and The Corneoretinal Potential

Most cells in the body have an electrical potential across their membranes. This potential, known as the membrane potential, arises from the unequal distribution of cations and anions (mainly sodium Na^+, Potassium P^+, and chlorine Cl^-) on the opposite sides of the membrane. The cytoplasm inside the cell is negatively charged relative to the extracellular fluid, ranging from -50 to -200 mV (CAMPBELL; REECE, 2008; GUYTON; HALL, 2011). This imbalance of charged particles is described by the Nernst equation and is caused by four factors: the chemical force caused by the ion concentration gradient, the inwardly directed electric field (effect of the membrane potential on ion movement), the membrane pores that allow different types of ions to pass through, and the active transport of ions (CAMPBELL; REECE, 2008; WEBSTER, 2010).

The microscopic electrochemical activity of cells can cause a macroscopic potential distribution on the surfaces of body tissues, commonly known as bioelectric potentials. This potential is generated in excitable cells, found in nervous, muscular, and gland tissues. These cells have two potential states: the resting potential, which is the steady electrical potential maintained between the internal and external environments, and the action potential, caused by an adequate stimulus. When the cell is in the resting potential state, it is called polarized, with the internal medium ranging from -40 to -90 mV relative to the external medium. A decrease in voltage is called depolarization magnitude, while an increase in magnitude is called hyperpolarization (WEBSTER, 2010).

The human eye contains excitable cells that produce bioelectric potentials. The retina contains cells that are sensitive to light and, like other excitable cells, generate receptor potentials. This behavior can be measured by electrodes placed either on the inner surface of the retina or on the cornea, and another electrode placed elsewhere (e.g., forehead). The recorded signal is known as electroretinogram (ERG), which is the electrical response of the retina to light and may be used for both scientific studies and clinical diagnoses (MALMIVUO; PLONSEY, 1995; WEBSTER, 2010).

Contrary to the bioelectric signals produced by excitable tissues, the corneoretinal potential is a spontaneous signal that arises from the increased metabolic rate of the retina. Emil du Bois-Reymond was the first to observe this potential in 1848, and it is often called steady, in the range of 0.4 to 1 mV, but it may vary slowly during the day (MALMIVUO; PLONSEY, 1995). The corneoretinal potential behaves as a single dipole oriented from the retina to the cornea and is the basis of electrooculography, whose main application is to measure eye movements.

2.5 Eye Blinks

There are three types of blinks: reflex, voluntary, and spontaneous (KANEKO; SAKAMOTO, 1999). Eye blinks are one of the fastest movements of the human body, occurring approximately 13,500 times a day, which is more than required for ocular lubrication (HÖMKE; HOLLER; LEVINSON, 2018). The main function of a blink is to re-form the tear film layer of the cornea to maintain a clear and healthy eye (DOANE, 1980). However, blinks also have a reflex protective function and may serve as an index of cognitive load.

2.6 Eye Tracking

The history of eye tracking dates back from the 18th century, when researchers first used “afterimages”, which are images that remain in the eyes after exposure to the original image, to describe eye movements (DREWES, 2010). Javal (1879) and Lamare (1892) were the researchers who observed eye movements in the 19th century and introduced the term "saccade" to describe the sudden movements of the eyes. The first recorded attempts to measure movements of the eye were made by Ahrens (1891), Delabarre (1898), and Huey (1898), who used small levers attached to the eye to track movements on a surface covered with soot (DREWES, 2010; SINGH; SINGH, 2012).

Later in 1939, Jung applied electrodes to the skin near the eyes to measure the electric fields generated by eyeball dipole during vertical and horizontal movements. Using analog electronics, this research enabled the development of the first real-time processing of gaze data. However, only in the 1980s did small computers become powerful enough to perform real-time eye tracking, which opened the possibility of video-based eye trackers (Video-oculography) (SINGH; SINGH, 2012).

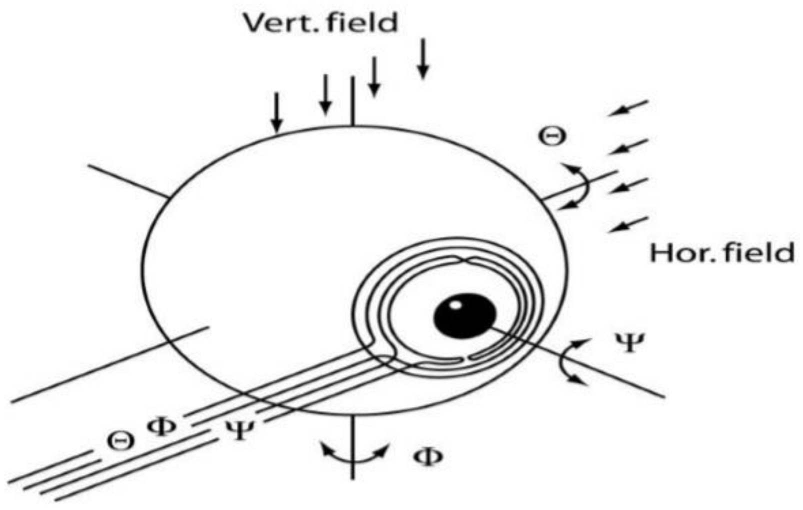
Research in the field of eye tracking uses either eye localization in the image, or gaze estimation. Eye localization involves detecting the presence and location of the eyes in an image, typically through the pupil or iris center. In turn, gaze estimation involves estimating and tracking where a person is looking in 3D space or determining their line of sight (HANSEN; JI, 2010).

The following methods can be used for gaze tracking: scleral search coil method, Infrared oculography (IROG), Electrooculography (EOG), and Video-oculography (VOG). The following topics briefly review the aforementioned eye-tracking techniques.

2.6.1 Scleral Search Coil Method

Created in the 1960s, the search coil method remains one of the most precise eye movement techniques (FANG; SHINOZAKI, 2018). This method applies the principle that when a coil of wire is moved through a magnetic field, a voltage is induced in the coil. To determine the eye position, a coil of wire is placed by using a contact lens, and the voltage induced by two or three external oscillating or revolving magnetic fields is measured (SINGH; SINGH, 2012). For example, Figure 8 illustrates two magnetic fields generated outside the head that induce a voltage on the coil placed on the eye. The eye orientation can then be calculated by demodulating the components of the different magnetic fields (SINGH; SINGH, 2012).

Figure 8 - Magnetic fields in the scleral search coil method



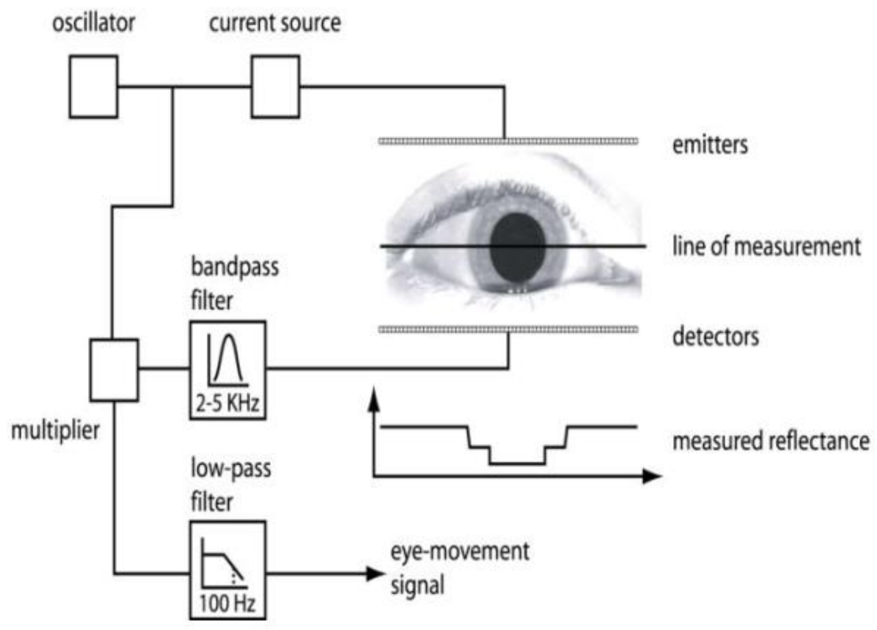
Source: Singh and Singh (2012).

The advantages of this method are its high accuracy and good time resolution, which allows the detection of small eye movements. However, this method is only used in research and clinical settings since the contact lenses require a small wire connected to an external device (SINGH; SINGH, 2012).

2.6.2 Infrared Oculography

This method uses infrared light to detect the boundary between the sclera and the iris. In this technique, an infrared light source is directed at the eye, and photodetectors are positioned to receive the reflected light from the frontal surface of the eyeball (Figure 9). From the infrared light reflected, the system tracks the boundary between the sclera and the iris or the boundary of the pupil and the iris to measure the eye position (SINGH; SINGH, 2012).

Figure 9 - IROG measurement technique



Source: Singh and Singh (2012).

Since infrared light is not visible, it causes neither distraction nor discomfort, nor does the lack of ambient light affect the measurements. This technique provides a good spatial resolution of up to 0.1 º and a temporal resolution of 1 ms. However, it is not suitable for tracking vertical movements as the eyelids block the upper sclera and the iris boundaries (SINGH; SINGH, 2012).

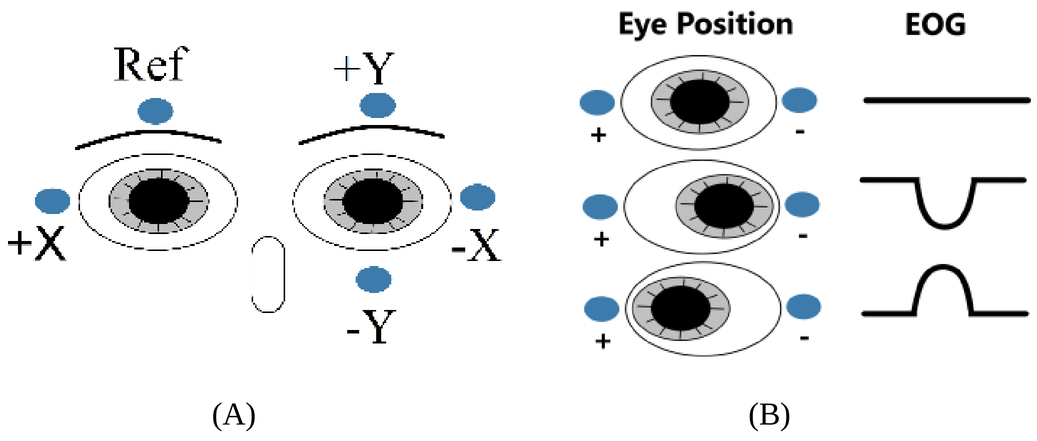
2.6.3 Video-oculography (VOG)

Video-oculography uses multiple cameras to estimate the eye position relative to the head or surroundings. It is a noninvasive method, not requiring wearable devices and providing good accuracy. This method relies on ambient light, camera quality, and advanced signal processing, which may result in higher user costs (SINGH; SINGH, 2012).

2.6.4 Electrooculography

Whereas the main sources of bioelectric signals are those produced by excitable tissues (e.g., nerve and muscle cells), electrooculography (EOG) is based on the corneoretinal potential, which is a static electric polarization of the eye. The potential between the cornea (positive pole) and retina (negative pole) can be represented as a spherical battery (dipole), with potentials varying from 0.4 to 1 mV (FANG; SHINOZAKI, 2018; MALMIVUO; PLONSEY, 1995). This dipole of the ocular globe generates microcurrents in the conductive tissue around the eyes that can be measured by skin surface electrodes placed near the eyes. As shown in Figure 10A, to capture eye signals, five electrodes are placed on the face: two for detecting horizontal movements, two for vertical movements (+X, –X and +Y, –Y), and a reference electrode.

Figure 10 - (A) Electrode placement (B) changes in the electrical field generated by eye movements



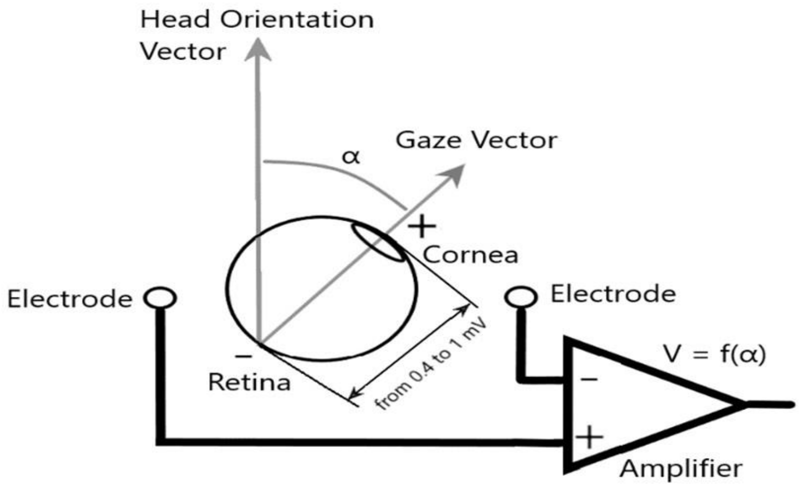
Source: Adapted from Doyle, Kucerovsky and Greason (2006).

Figure 10B illustrates the voltage behavior of the electrodes placed on the face due to horizontal movements of the eyes. The left movement creates a similar waveform to the right movement, but with the opposite amplitude, which allows identifying the direction of the gaze. Each degree of eye movement generates about 14 to 20 µV, with a complete movement registering between 50 to 3500 µV and frequencies of 0-100 Hz (SINGH; SINGH, 2012;

USAKLI; GURKAN, 2010). Additionally, EOG can accurately record movements of up to 70°, with an accuracy of 1.5° to 2° (DOYLE; KUCEROVSKY; GREASON, 2006).

The measurement of EOG signals is usually performed using operational amplifiers connected to a band-pass filter (SINGH; SINGH, 2012), as illustrated in Figure 11. For example, obtaining the horizontal signal involves subtracting the signal of +X from that of -X (following Figure 10A notation), or vice versa (CHANG, 2019). An advantage of EOG is that horizontal and vertical eye movements are easily distinguished, even in temporal series, which allows simple processing systems to be used (USAKLI; GURKAN, 2010). The main objective of processing EOG signals is to eliminate noise and artifacts from signals related to saccades and smooth pursuit movements. For example, median and low-pass filters are used to remove high-frequency EMG artifacts, whereas high-pass filters remove ocular drift (CHANG, 2019).

Figure 11 - Scheme of EOG electrical activity recording



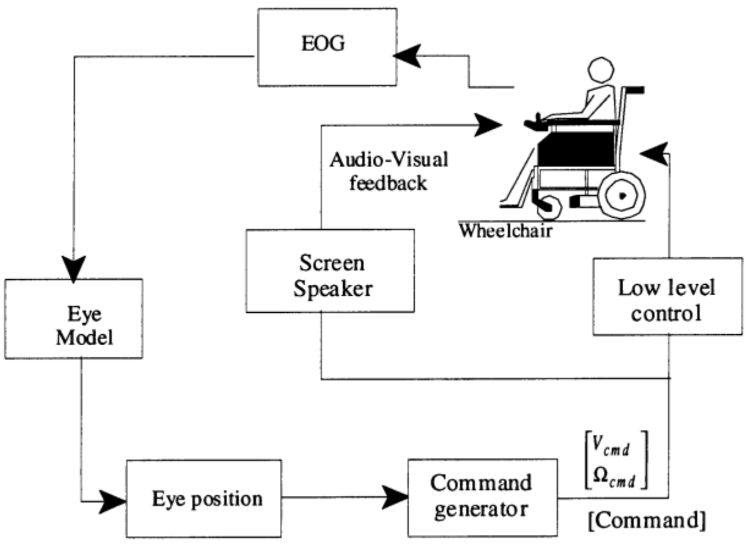
Source: Adapted from Chen and Newman (2004).

Eye blinks can also be detected by using EOG. During a blink, the eyeball experiences an upward movement called Bell's phenomenon, which was identified by Sir Charles Bell (IWASAKI et al., 2005). Electrooculogram recordings of eye blinks traditionally place electrodes in a bipolar montage above the eyebrow and above the malar prominence in a vertical plane in line with the pupil. This results in electrical potentials similar to those observed during vertical saccades (DENNEY; DENNEY, 1984; IWASAKI et al., 2005; MATSUO; PETERS; REILLY, 1975).

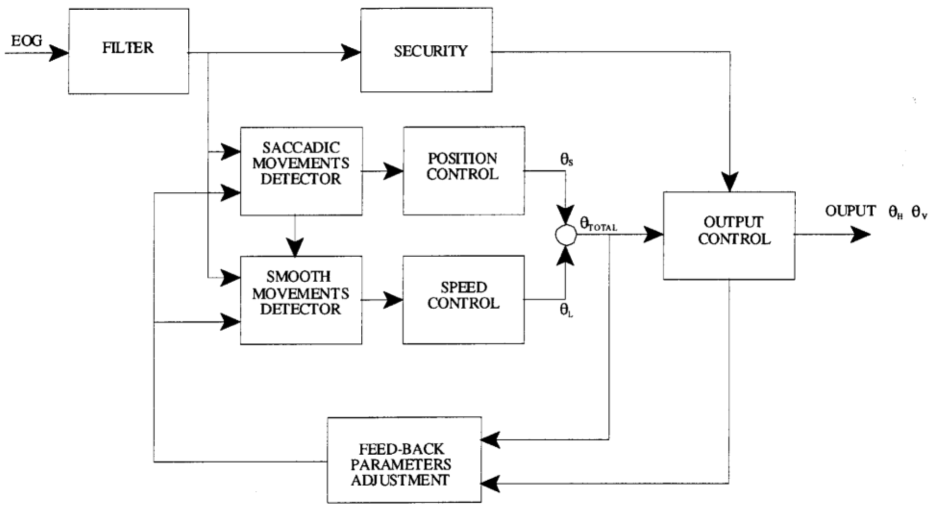
2.7 EOG-based devices

EOG has been used for developing an automatic wheelchair controlled by eye movements, as shown in Figure 12A (BAREA et al., 2002). For this application, a model was developed to determine the gaze direction by using recorded EOG data (Figure 12B). It filters out other biopotentials, detects when the eyelids are open (security block), classifies whether it is a saccadic or smooth movement, and outputs the deviation angles $\theta_{Horizontal}$ and $\theta_{Vertical}$ with respect to the central position of the eye. In addition, the model outputs are sent to a high-level controller that generates the linear and angular speed commands of the wheelchair ($[V_{cmd}\ \Omega_{cmd}]^T$). The high-level controller also uses the eye position to navigate a graphical interface (similar to a computer mouse) and to select between a rotational and a linear movement of the wheelchair, which is controlled by a state machine.

Figure 12 - Wheelchair controlled by EOG



(A)

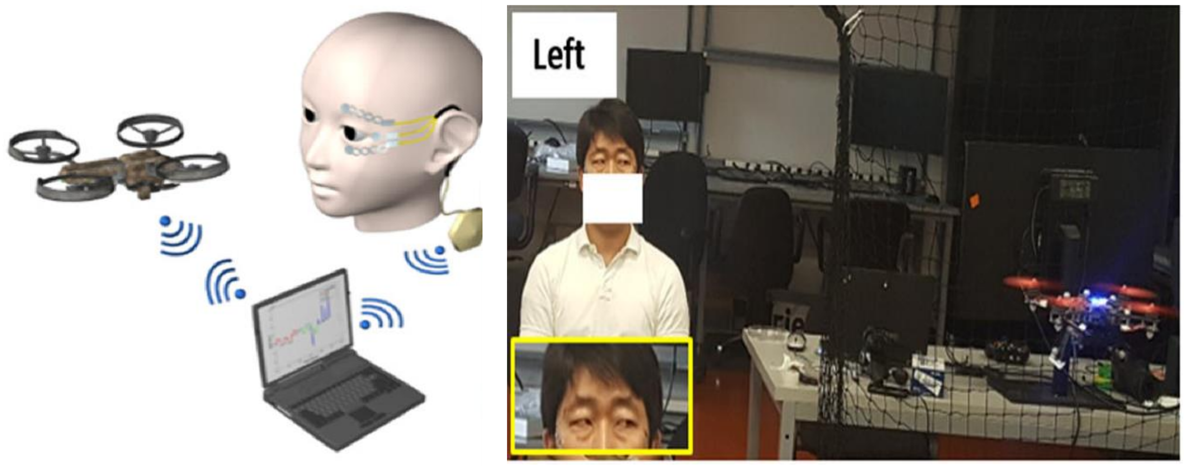

EOG
FILTER
SECURITY
SACCADIC MOVEMENTS DETECTOR
POSITION CONTROL
θ_S
θ_{TOTAL}
OUTPUT CONTROL
OUPUT θ_H θ_V
SMOOTH MOVEMENTS DETECTOR
SPEED CONTROL
θ_L
FEED-BACK PARAMETERS ADJUSTMENT

(B)

Source: Barea et al. (2002).

EOG electrodes were developed by using graphene electronic tattoos (GET), which are extremely thin, soft, transparent, and breathable, and which have shown improved skin contact, reduced size, high signal-to-noise ratio (SNR), and lower motion artifacts. These GET EOG electrodes were connected to a wireless transmitter to control the movements of a drone, as shown in Figure 13 (AMERI et al., 2018).

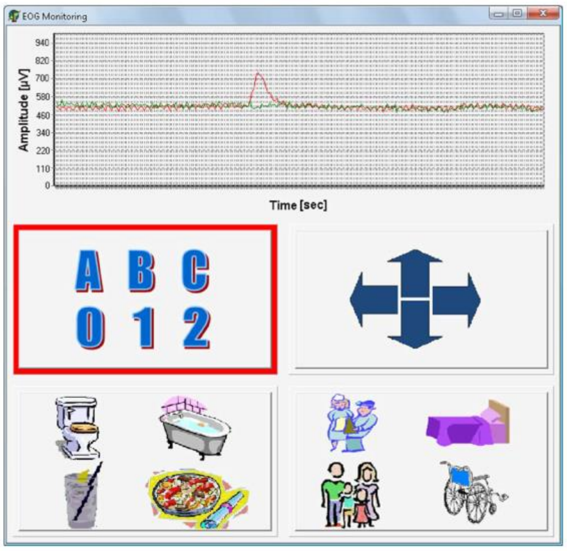
Figure 13 - Drone system control using graphene electrodes



Source: Ameri et al. (2018).

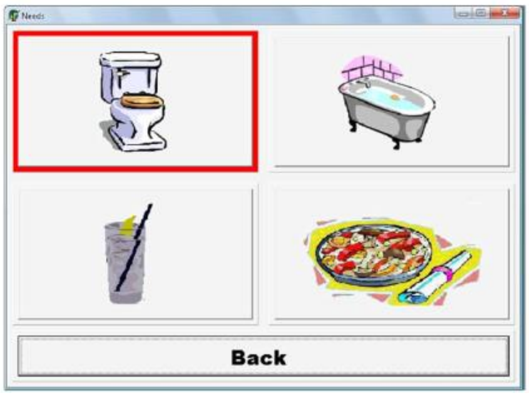
EOG has been widely used in human-machine interfaces (CHEN; NEWMAN, 2004; FANG; SHINOZAKI, 2018; USAKLI et al., 2018; USAKLI; GURKAN, 2010), in which the movement of the eyes is used to control a computer cursor, write words, and navigate graphical interfaces. For example, Usakli and Gurkan (2010) developed an EOG system to navigate a menu with options for the user, including a virtual keyboard that allowed the user to write a 5-word letter in 25 s (Figure 14).

Figure 14 - (A) Graphical interface developed for the EOG system (B) EOG acquisition system connected to a notebook (C) interface submenu (D) virtual keyboard

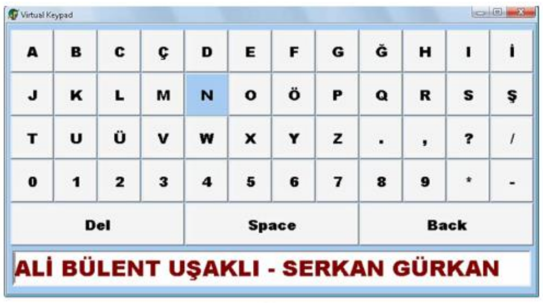


(A)

(B)



(C)



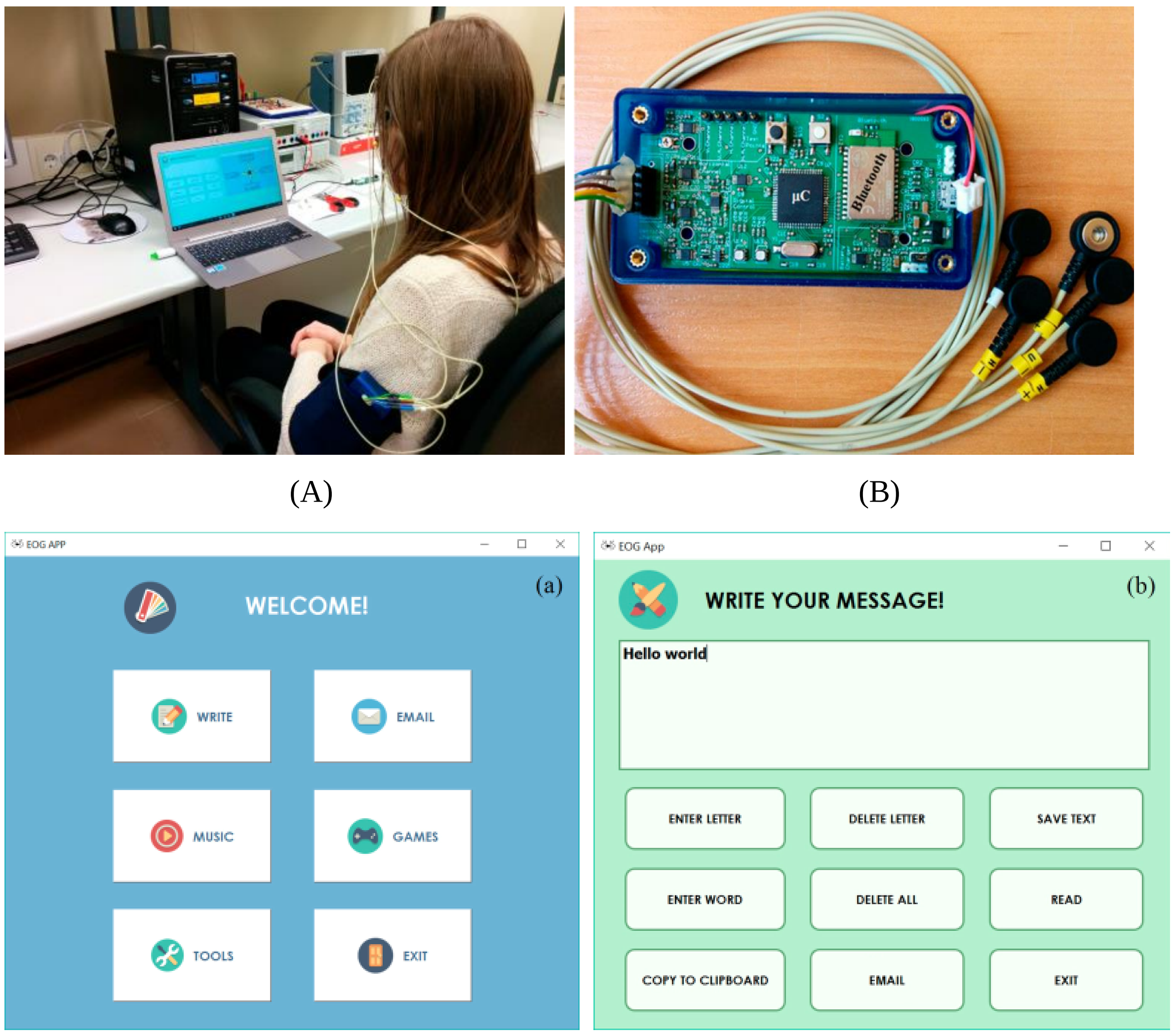
ALİ BÜLENT UŞAKLI - SERKAN GÜRKAN

(D)

Source: Usakli and Gurkan (2010a).

In 2017, Lopéz et al. created a computer writing system (Figure 15A) that used eye movements and was composed of: a wireless high-performance analog front-end (LÓPEZ et al., 2020) (Figure 15B) for acquiring and transmitting the signals generated by EOG; a software application designed for data processing, mapping, and signals classification; and a GUI (Figure 15C) that, besides other functions, allowed the user to write a text.

Figure 15 - (A) System in use (B) hardware device (C) graphical interface

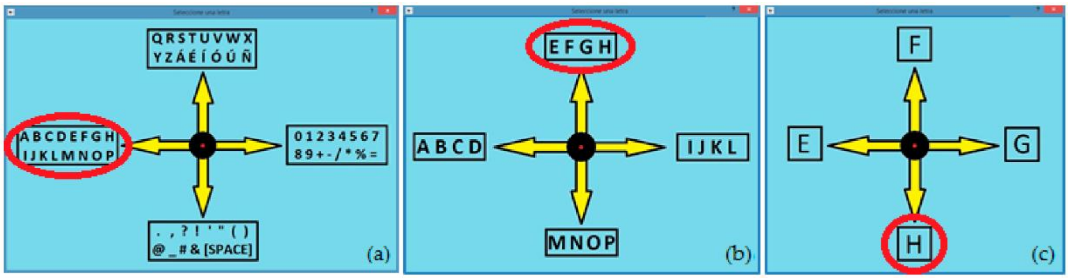


(C)

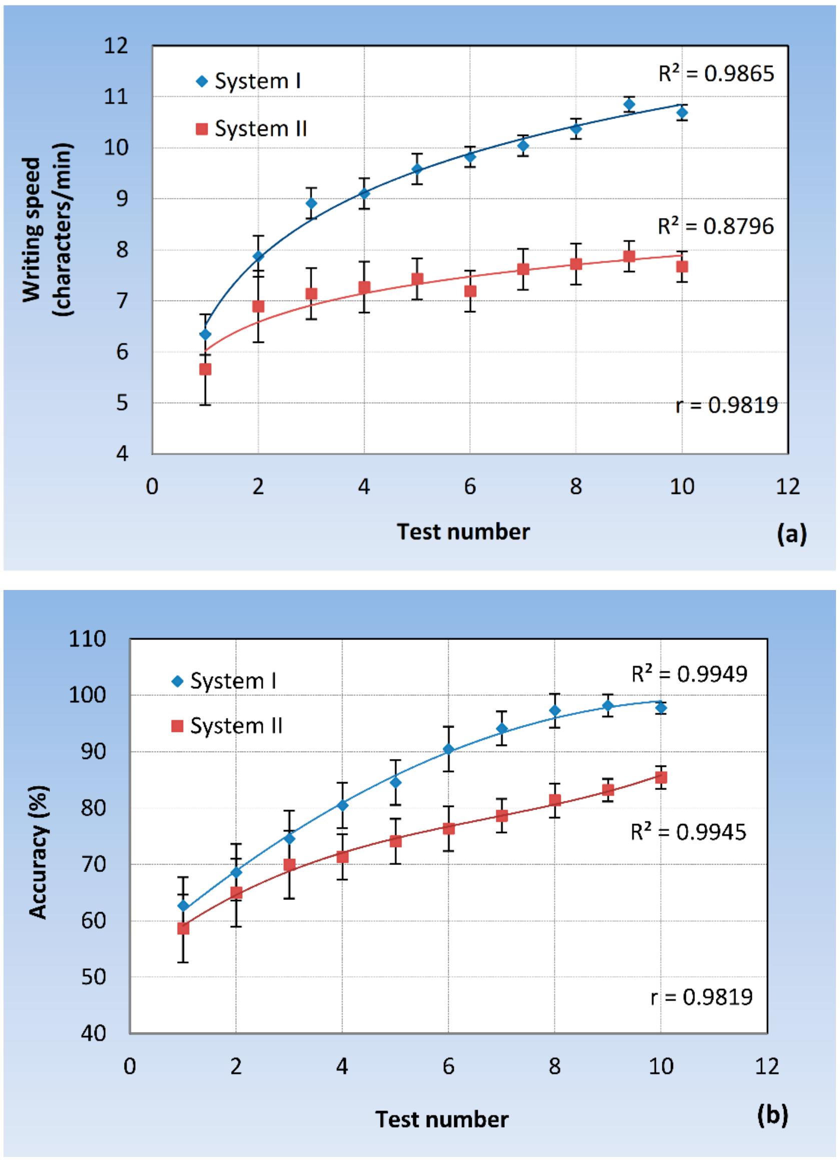
Source: López et al. (2017).

To type using eye movements, a virtual keyboard based on groups of letters was developed (Figure 16A). For selecting one of the 64 of the alphanumeric characters, the user had to perform three simple eye movements. For instance (Figure 16A), to type the letter “H”, the user needed to look left, select the first group of letters, look upwards, and finally look down. In the same study (LÓPEZ et al., 2017), they compared the proposed group typing system (System I) with a conventional virtual keyboard typing system (System II), in the same test environment and hardware. The results showed (Figure 16B) that the writing speed and accuracy of “System I” was always greater than those of “System II”, and with training, the performance difference tended to increase.

Figure 16 - (A) Selecting a character in the virtual keyboard using three basic eye movements (B) Evolution of writing speed and accuracy of System I and System II with the number of tests



(A)

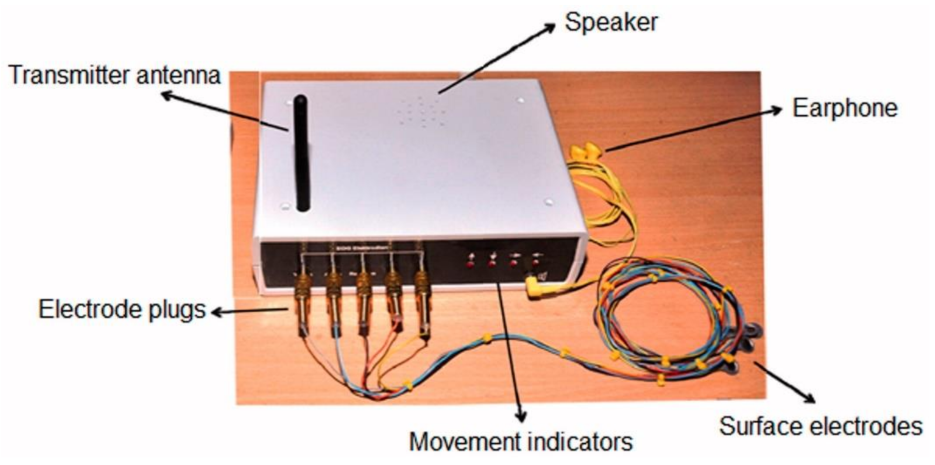


(B)

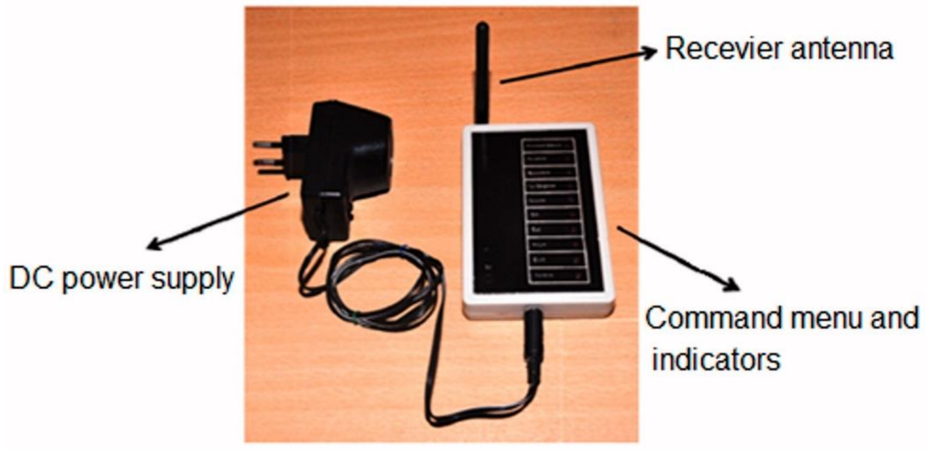
Source: López et al. (2017).

Recently, Usakli and Gurkan developed a low-cost and stand-alone EOG system (USAKLI et al., 2018) for users to communicate remotely using 10 pre-programmed messages transmitted wirelessly, without the need of a personal computer or display. The system consisted of two distinguishing devices. The first one, shown in Figure 17A, acquires and processes the EOG signals, and transmits one of the 10 messages pre-programmed. The second device, shown in Figure 17B, receives the information through a radio frequency and communicates it with a speaker or LEDs (USAKLI et al., 2018).

Figure 17 - Complete EOG communication system: (A) acquisition, processing and transmission device (B) receptor system for remote monitoring



(A)



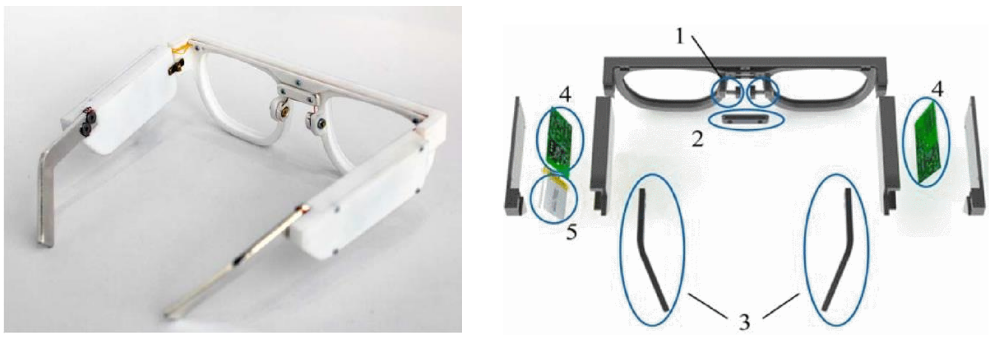
(B)

Source: Usakli et al. (2018).

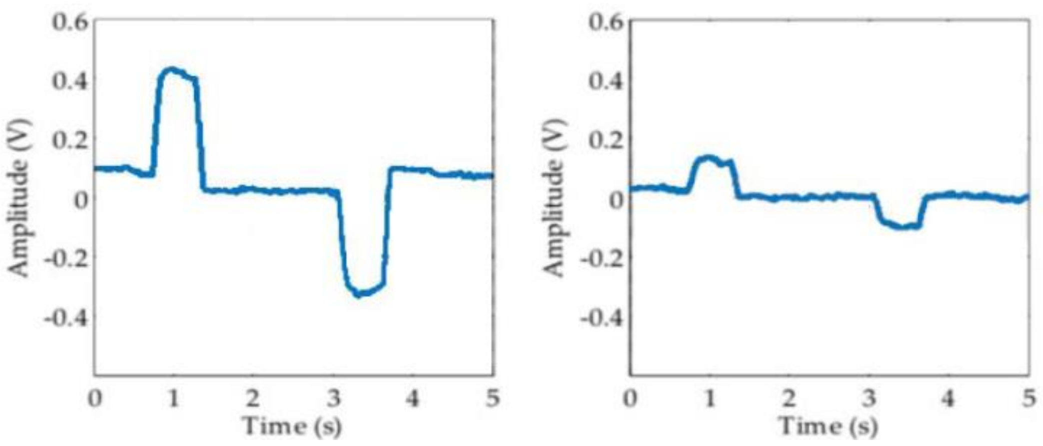
Most EOG projects reviewed use wet electrodes, which involve applying a conducting gel or solution between the skin and the electrode to improve conduction. However, this method can result in decreased conductivity and skin irritation over time (CHANG, 2019). Additionally, most EOG-based systems are attached to the head using headbands, head caps, or frames of glasses with wires, which can be uncomfortable for everyday use (KOSMYNA *et al.*, 2019), developed for investigation only and not as a product (CHANG, 2019).

In contrast, the wireless eyeglass AttentivU (Figure 18A) has a socially acceptable and comfortable design. It has embedded electronics that allow it to acquire and process EEG and horizontal EOG signals to analyze the level of user attention, providing audible feedback when the attention level drops. Although AttentivU uses dry electrodes, the signal can still be clearly distinguished in temporal series, as shown in Figure 18B (KOSMYNA et al., 2019).

Figure 18 - (A) AttentivU: Eyeglasses with embedded EOG acquisition and processing system. (B) comparison of the EOG signals obtained using dry and wet electrodes



(A)



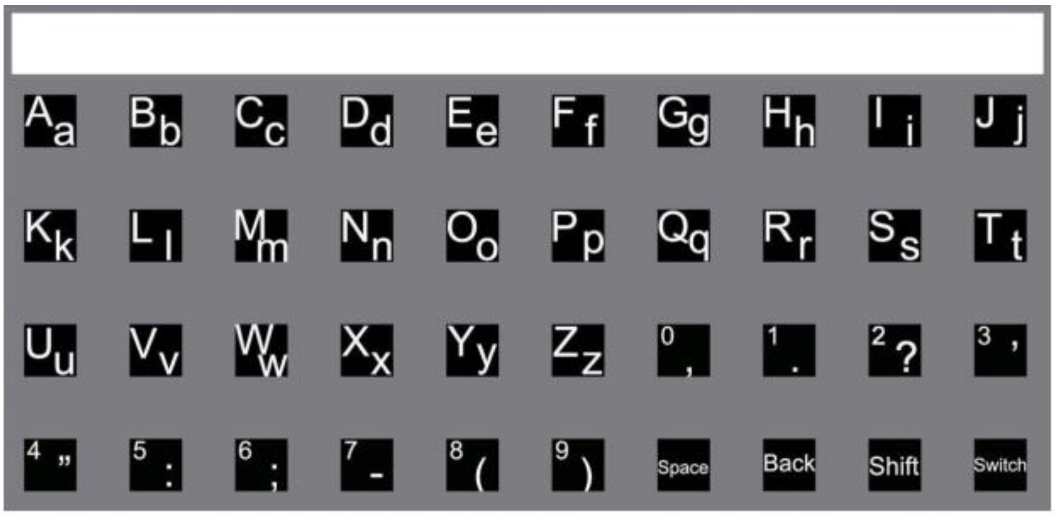
(B)

Source: Kosmyna et al. (2019).

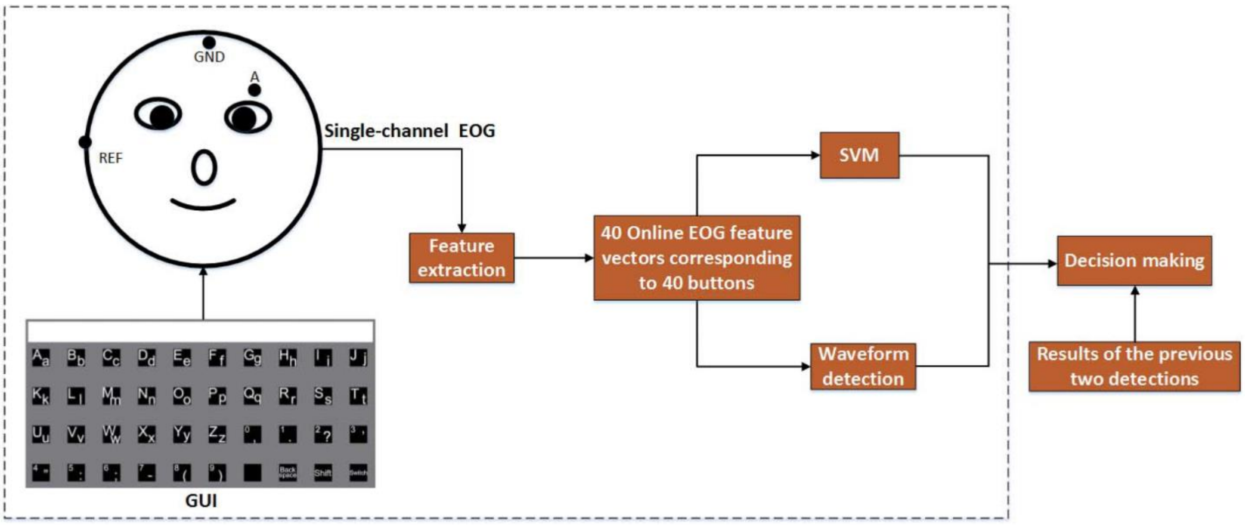
In 2017, Li and He proposed a single EOG channel system (HE; LI, 2017) to write on a virtual keyboard using only eye blinks. This enabled fast communication (4.18 s per character) by only blinking in synchrony with the flashes of the buttons of the virtual keyboard. The efficiency of the system was due to the rapid flashing of the buttons combined with an effective blink detection method. The authors later applied a similar technique to control wheelchair (HUANG et al., 2018), a virtual reality environment (XIAO; QU; LI, 2019) and an asynchronous hybrid brain-computer interface (BCI) (HE et al., 2020).

The first study (HE; LI, 2017) presented a GUI with 40 buttons (Figure 19A) that flashed randomly to reduce interference from adjacent buttons. The user had to blink in synchrony with the flashes of the desired button. Each flash lasted 100 ms with a 30-ms interval between successive flashes, resulting in only 1.2 s for one entire round of flashes. To detect the eye blink, two processing operation ran in parallel: the waveform detection and the support vector machine (SVM), as illustrated in Figure 19B.

Figure 19 - (A) The GUI of the EOG-based speller (B) flowchart of the detection algorithm


Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj
Kk Ll Mm Nn Oo Pp Qq Rr Ss Tt
Uu Vv Ww Xx Yy Zz 0 , 1 . 2 ? 3 '
4 " 5 : 6 ; 7 _ 8 (9)
Space
Back
Shift
Switch

(A)

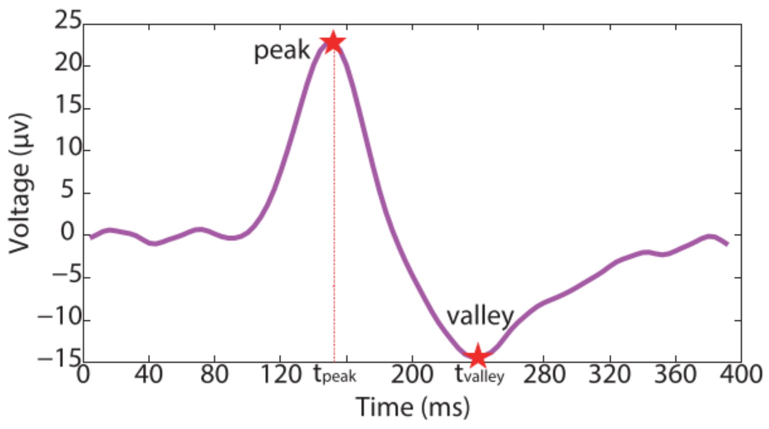

GND
A
REF
Single-channel EOG
Feature extraction
40 Online EOG feature vectors corresponding to 40 buttons
SVM
Waveform detection
Decision making
Results of the previous two detections
GUI

(B)

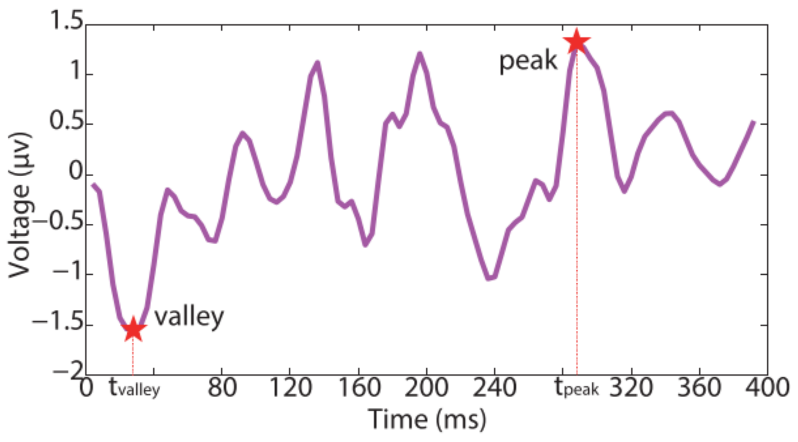
Source: He and Li (2017).

The detection algorithm (Figure 19B) extracts 40 feature vectors of the filtered and differentiated EOG signal, corresponding to 40-button flashes. These 40-feature vectors are then fed into an SVM classifier, resulting in 40 SVM scores for the 40 buttons. Simultaneously, the 40 online feature vectors are also processed for waveform detection. Figure 20 shows two waveforms of single feature vectors: Figure 20A corresponds to an eye blink, and Figure 20B corresponds to a non-blink.

Figure 20- (A) Single blink (B) and a non-blink differentiated EOG waveforms



(A)



(B)

Source: He and Li (2017).

The waveform detection uses thresholds to detect the eye blink: duration threshold (‘d’), which is the time difference between the peak and the valley and energy threshold (‘e’), both shown in Equation 01.

$$\begin{cases} d = t_{valley} - t_{peak} \\ e = \sum_{t=t_{peak}}^{t_{valley}} (x_t')^2 \end{cases} \quad (1)$$

These thresholds are determined by a calibration process, which uses 200 feature vectors, removing values that are either greater than $\mu + 3\sigma$ or less than $\mu - 3\sigma$ (μ represents the mathematical expectation and σ is the standard deviation). The energy threshold is then selected as the smallest value within the top 95% of the sorted energy values, whereas the duration thresholds are the smallest values within the top 95% and bottom 5% of the sorted duration values. Finally, the waveform detection selects the button according to the following criteria (Equation 02):

$$b_i = \begin{cases} 1, & if\ D_{min} \leq d_i \leq D_{max}\ and\ e_i \geq E \\ & 0,\ otherwise \end{cases} \quad (2)$$

The system selects the button based on the waveform detection and SVM classification results, referred to as "decision making" in Figure 19B. The selected button is the one designated as the potential choice chosen twice across the three consecutive detections (successive rounds of button flashes). The button with the highest final score is chosen as the target button for the current decision, and the character is inputted. This means that the decision step is concluded, and the system moves on to the detections and decisions of the following operation. Lastly, Li and Hi (2017) performed a clinical trial with eight individuals without disabilities to assess the system effectiveness, resulting in an average accuracy of 94.4% and a time of 4.14 s to select a character.

The following studies employed a similar technique to detect blinks, but with some modifications. Huang et al. (2018) did not use the SVM technique; instead, an EOG waveform was used with a few parameters in addition to the differential EOG parameters. Xiao, Qu and LI (2019) used only the parameters of the differentiated waveform and achieved an average accuracy of 95.25%. After that, both EEG and blink EOG signals were used, and the blink detection was achieved by combining SVM and the EOG differentiated waveform parameters (HE et al., 2020), similarly to (HE; LI, 2017).

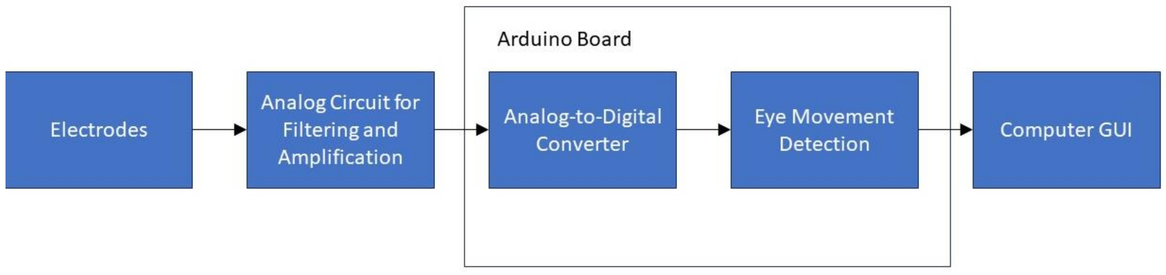
3. DEVELOPMENT

The following section describes the preliminary studies, the initial results, and the rounds of development based on the observations of researchers of the Instituto de Reabilitação Lucy Montoro. This section also highlights the development and testing of a prototype for the EOG system, which was evaluated with subjects with and without disabilities. This section concludes with a description of the tests of feasibility for developing a comfortable acquisition system.

3.1 Pilot Study

As a preliminary study, we developed an EOG communication system (COSTA et al., 2020) for people with severe motor disabilities who are unable to communicate by speaking, writing, or gesturing. Figure 21 illustrates the block diagram of the system. The system was used to acquire, filter, and process the EOG signals for detecting horizontal movements of the eyes, which were translated into answers to questions by an interface that ran on a personal computer (PC).

Figure 21 - System block diagram


Arduino Board
Electrodes
Analog Circuit for Filtering and Amplification
Analog-to-Digital Converter
Eye Movement Detection
Computer GUI

Source: Author.

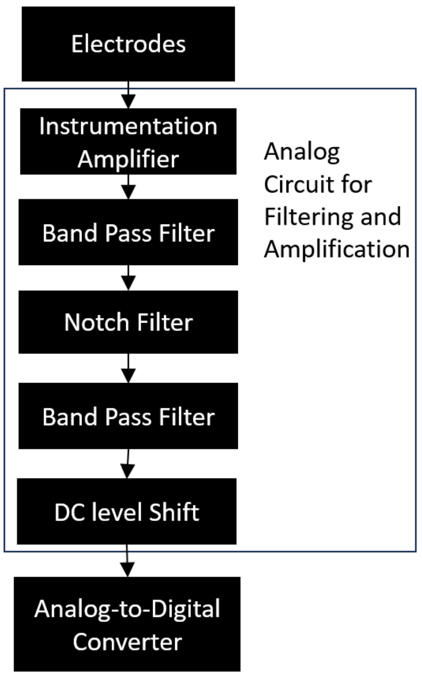
3.1.1 Acquisition Circuit

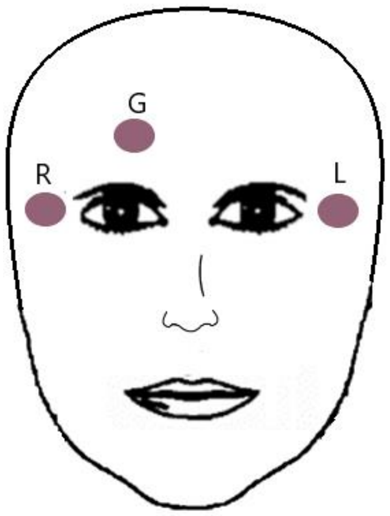
The block diagram of the acquisition stage is illustrated in Figure 22A. The system was designed to use three silver chloride electrodes (Ag/AgCl), two electrodes on the right and left of the outer sides of the eyes (R and L in Figure 22B), and one reference electrode (G in Figure 22B) on the forehead.

Based on previous research (CHANG, 2019; USAKLI; GURKAN, 2010), we designed the amplification and filtering circuit (Figure 23) to eliminate noise and enhance the signal of interest, which are primarily saccades and smooth pursuit movements. The typical amplitude of the signal of interest is less than 500 µV, and the frequencies are typically up to 5 Hz. To eliminate high and low frequency noise, the previous studies reviewed in (CHANG, 2019) set the cutoff frequency of the low-pass filters between 10 and 60 Hz, and the high-pass filters between 0.05 and 0.1 Hz. Based on these considerations, we applied a final amplification gain of 74.15 dB ($51 \times 10 \times 10 = 5100$ V/V) and filtered the frequencies below 0.16 Hz and above 10.6 Hz.

To achieve this high gain, two stages of band-pass filters were required because a higher gain in the instrumentation amplifier would decrease the SNR, and a single 2nd-order band-pass filter would not provide the attenuation required to remove high-frequency noise. Lastly, the signals obtained after amplification and filtering were sampled by an Arduino UNO R3 board.

Figure 22 - (A) EOG acquisition circuit block diagram (B) electrodes placement


Electrodes
Instrumentation Amplifier
Band Pass Filter
Notch Filter
Band Pass Filter
DC level Shift
Analog Circuit for Filtering and Amplification
Analog-to-Digital Converter

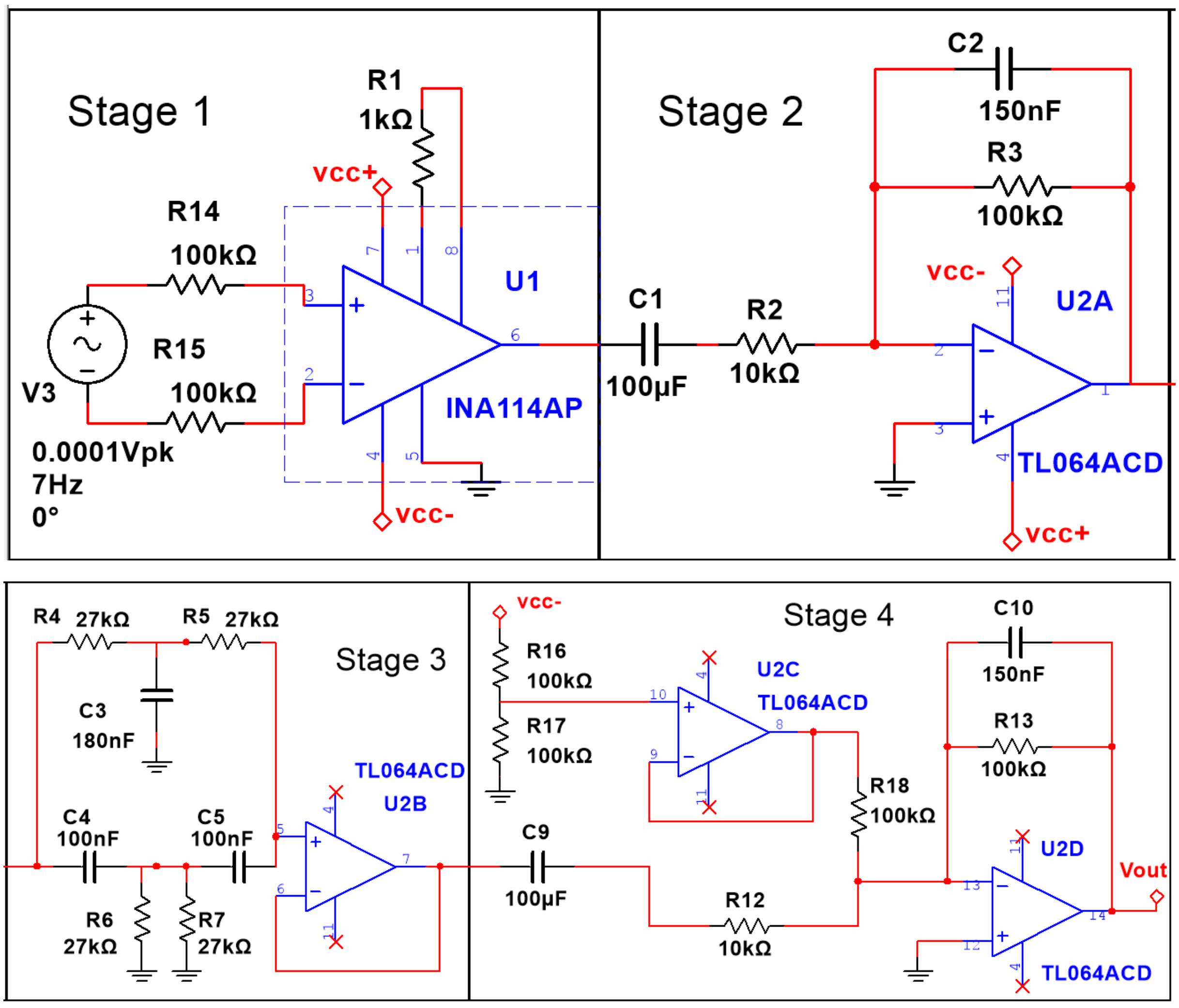


Source: Author.

The amplification and filtering circuit was implemented using a +/- 5 V symmetrical direct current (DC) power supply and the TL064 operational amplifier. The circuit was divided into four stages, as described below and shown in Figure 23:

a) stage 1: an instrumentation amplifier (INA114AP) of approximately 34.15 dB (~51 V/V), with high input impedance (100 GΩ) and high common noise rejection (115 dB). The input of the INA114AP uses two input resistors of 100 kΩ to protect the user in case of circuit failure;
b) stage 2: an active 2nd-order band-pass filter with 20 dB gain (10 V/V), low-pass cutoff frequency of 10.61 Hz, and a high-pass cutoff frequency of 0.16 Hz. The inverting amplifier was used to obtain a higher SNR with gain lower than one in the rejection band;
c) stage 3: a notch filter to remove the 60 Hz power-line noise captured by the cables of the electrodes;
d) stage 4: similar filter of the second stage, with a 2.5 V DC level shift, to allow the Arduino to read the total excursion of the signal. The output of the last stage of the circuit was connected to a 10-bit analog-to-digital converter (ADC) of the Arduino board.

Figure 23 - Schematic of the signal acquisition circuit

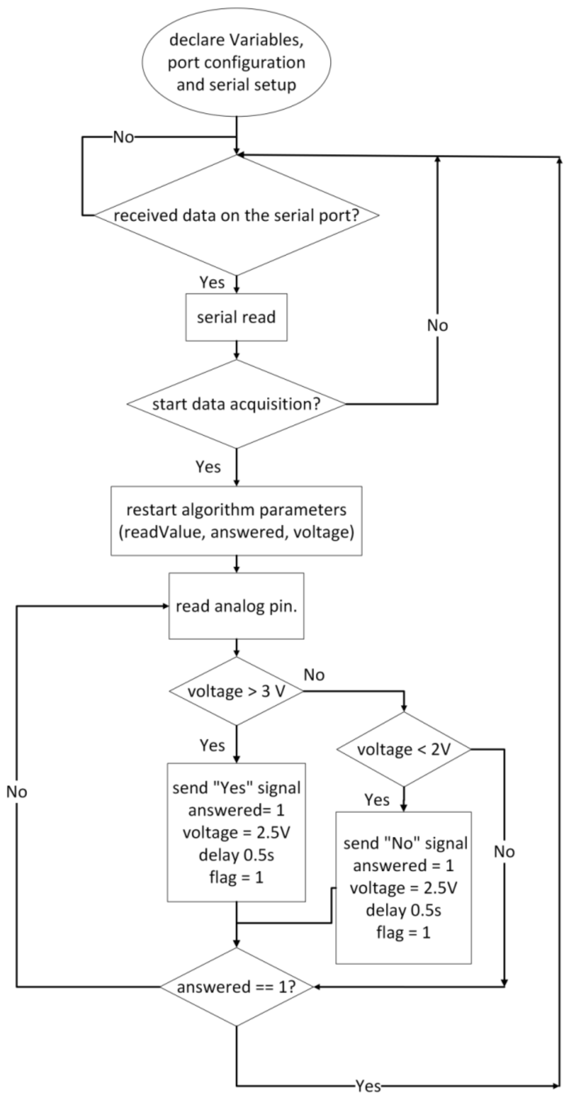


Source: Author.

3.1.2 Processing Algorithm

The eye movement detection was performed by using a level detection algorithm implemented in the Arduino board, which was connected to the analog acquisition circuit and to the PC by using the Arduino USB serial port. Figure 24 illustrates a flowchart of the firmware algorithm. When a question is generated by the graphical interface, the computer interface sends a command ("char") for the Arduino to start reading the ADC output. If this output is above 3 V, the Arduino sends a signal to the interface, representing the "Yes" answer. If the voltage is below 2 V, the Arduino sends a signal to the interface, representing the "No" answer. After an answer is displayed, there is a delay of 0.5 s to avoid multiple answers in case the interface sends another signal to start the acquisition immediately after receiving an answer from the Arduino.

Figure 24 - Level detection algorithm

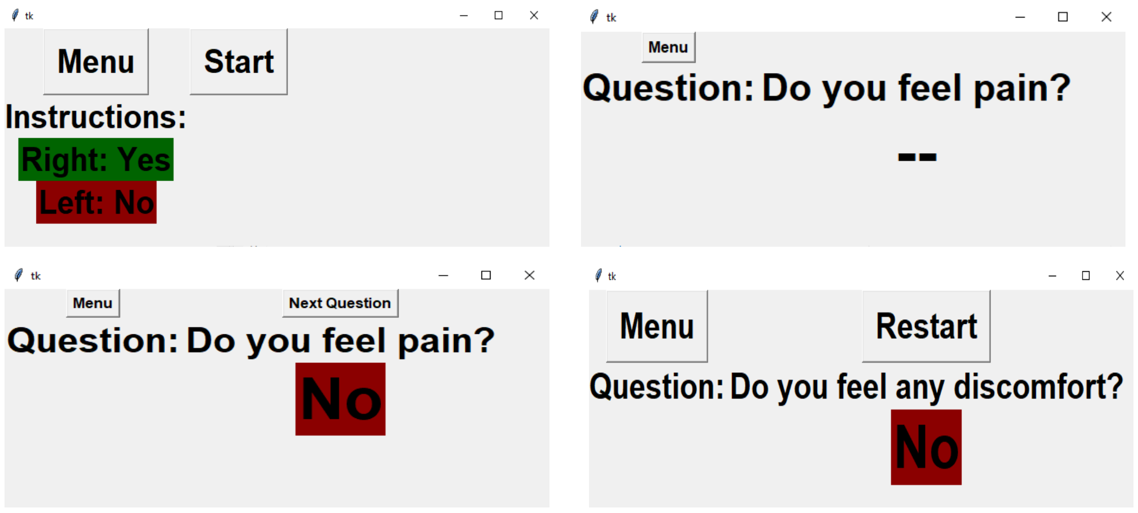


Source: Author.

3.1.3 Graphical Interface

A graphical user interface (Figure 25) was developed in Python to translate the user's eye movements acquired by EOG electrodes into answers to four pre-programmed questions. If the Arduino detects a movement to the right, the interface prints "Yes". Otherwise, if the user looks to the left, it prints "No".

Figure 25 - Printed screens of the graphical interface



Instructions:
Right: Yes
Left: No

Question: Do you feel pain?
--

Question: Do you feel pain?
No

Question: Do you feel any discomfort?
No

Source: Author.

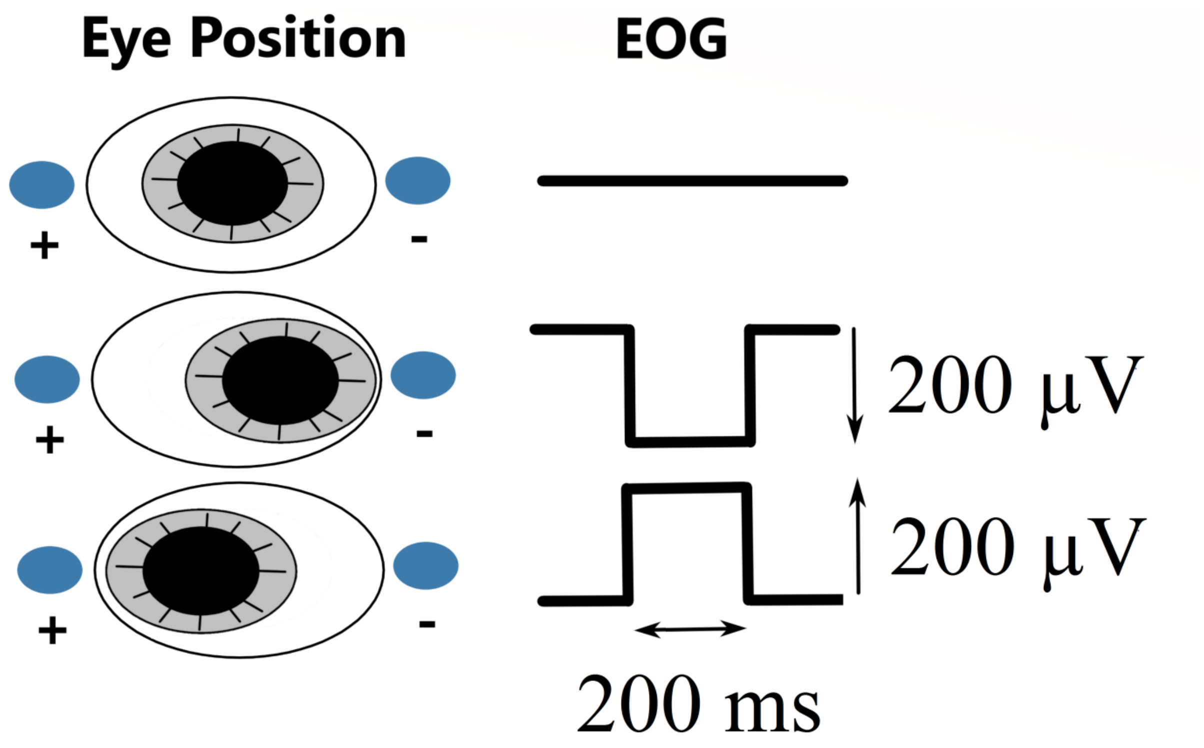
The interface requires a helper to assist in starting and advancing the sequence of questions using a computer mouse or keyboard. Thus, if the user selects "start", the first question appears, and the Arduino starts the EOG readings. The algorithm then detects an eye movement and displays the answer. After answering the question, the participant's assistant can select "next question", triggering a new Arduino acquisition. This process continues until the system reaches the final question and can be restarted.

3.1.4 Experimental Protocol

To evaluate the performance of the system, we created synthetic signals with morphology similar to those generated by eye movements (CHANG, 2019; USAKLI;

GURKAN, 2010). The signals generated by eye movements to the left and to the right were represented by a waveform consisting of two square pulses with opposite amplitude of 200 µV and 200 ms width, as illustrated in Figure 26. The waveform was generated by using a function generator (DSOX1102G, Keysight Technologies, USA) with amplitude reduced by a voltage divider to 200 µV, which is the amplitude of an EOG signal (USAKLI; GURKAN, 2010).

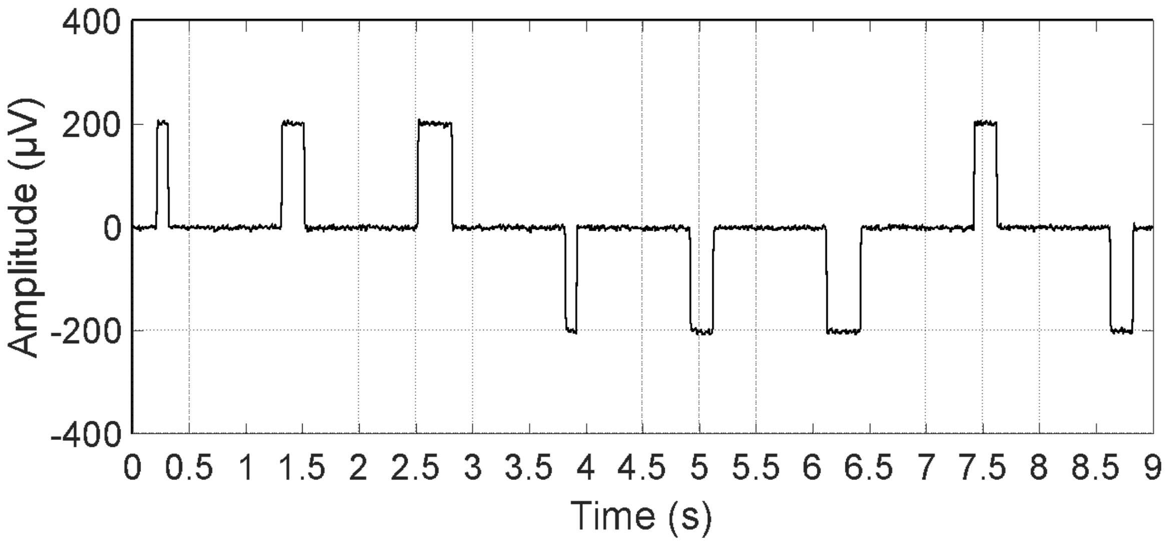
Figure 26 - Eye movement scheme



Source: Author

Experiments were also carried out to test circuit accuracy, using the sequence of pulses illustrated in Figure 27, representing eye movements to the left and to the right. This waveform was generated by a function generator (DG1022, RIGOL Technologies, China) and a voltage divider (to obtain 200 µV) with pulse widths of 100, 200 and 300 ms. This signal was then applied to the input of the first stage of the acquisition circuit (“Vin”) and with the Arduino configured to send a signal to the serial port when it detected a signal corresponding to a look to the right or to the left.

Figure 27 - Sequence of pulses simulating eye movements

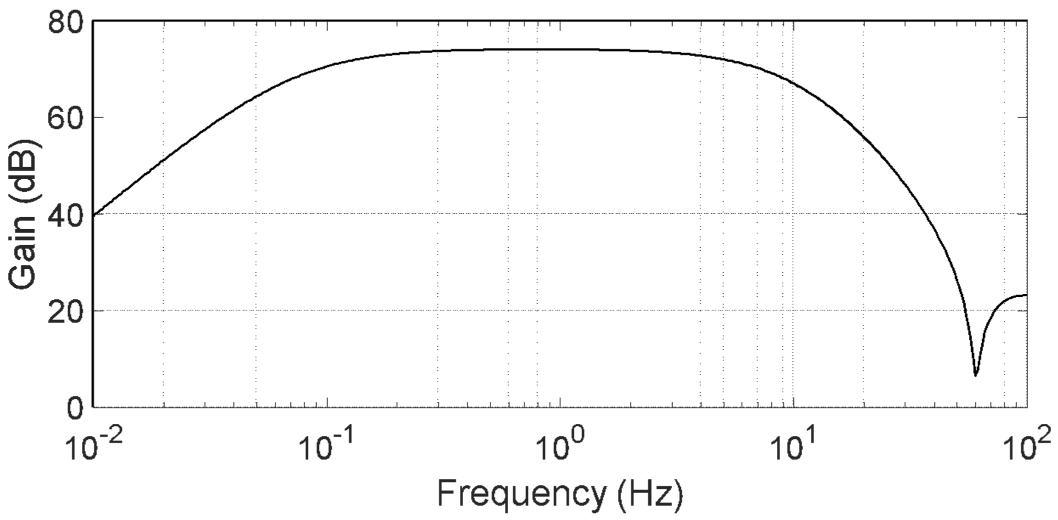


Source: Author.

3.1.5 Results of the pilot study

Figure 28 illustrates the frequency response of the acquisition circuit with 74 dB gain in the pass band and with 66 dB in the cutoff frequency of the low-pass filter (10.61 Hz). Together with the band-pass filters, the notch filter provided a final system gain of 6.4 dB at 60 Hz.

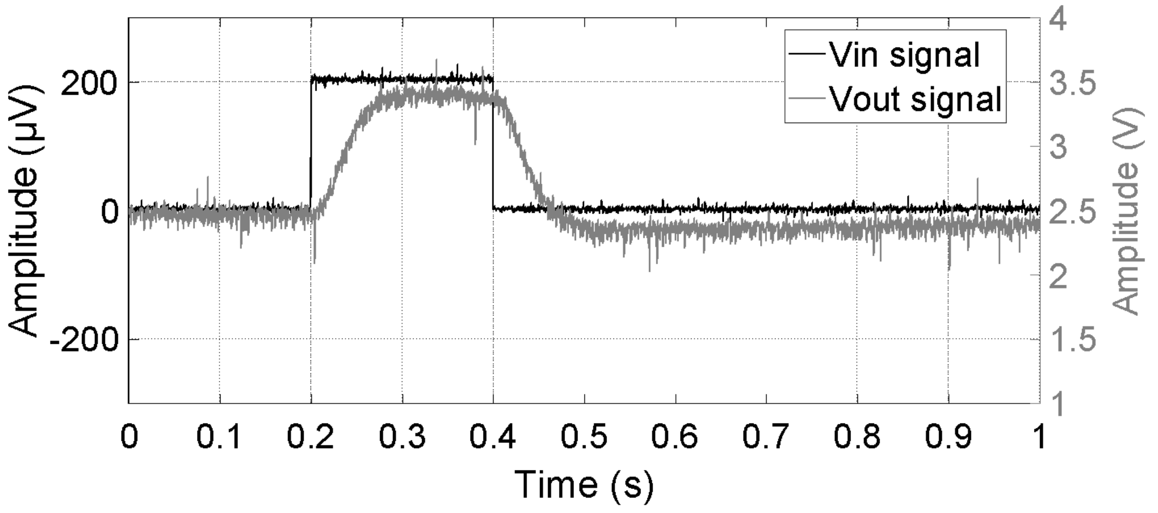
Figure 28 - Frequency response of the acquisition circuit



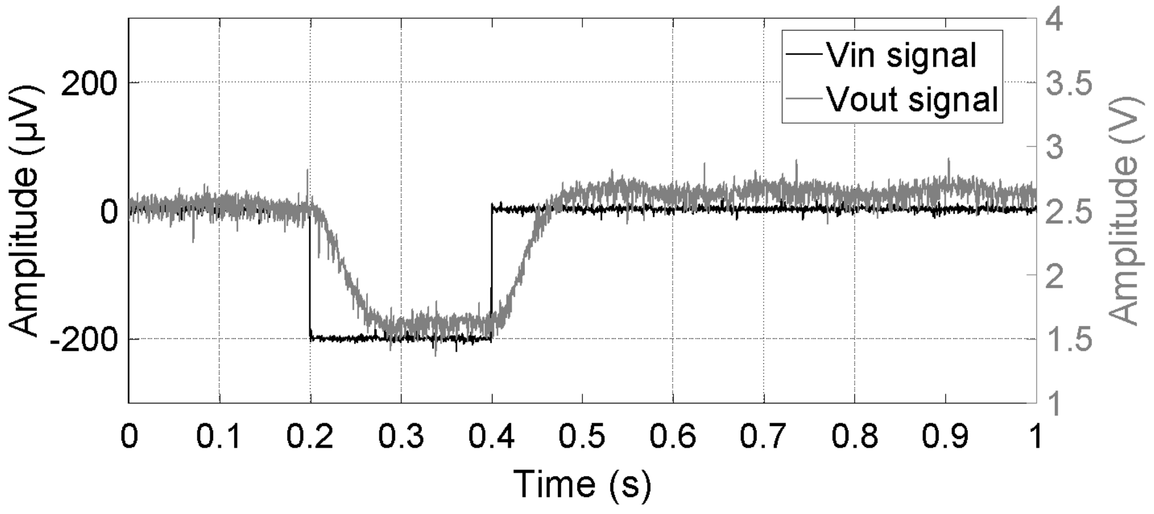
Source: Author.

Figure 29 shows a time-series signal obtained by using the pulse wave generated to simulate eye movements. Figure 29A shows the simulation of the movement of the eyes to the right, whereas Figure 29B shows the movement to the left. The final gain of the circuit was approximately 74bB, as defined, and the signals could be easily distinguished in time series, as also described by (CHANG, 2019; USAKLI; GURKAN, 2010).

Figure 29- Acquisition circuit response to synthetic eye movements: (A) left eye movements (B) right eye movements



(A)



(B)

"Vin" refers to the first stage of the acquisition input whereas "Vout" refers to the last stage (following Figure 23 notation)

Source: Author.

The results showed that the designed acquisition system could successfully record artificial EOG signals with high SNR and that the synthetic pulses were correctly classified by the algorithm. However, the main limitation of the design was that the person needed help to use the device, which could be eliminated by detecting the blink of the eyes as a trigger.

The use of synthetic signals did not allow the study of the influence of vertical movements on the horizontal electrodes, known as crosstalk (CHANG, 2019). Previous studies (CHANG, 2019; USAKLI; GURKAN, 2010) have shown that this issue can be addressed by ignoring EOG signals with low amplitudes, which was already performed by the algorithm used.

The total cost of the device designed as a proof of concept was approximately 35 USD, which was less expensive than the 200 USD complete system presented by Usakli (USAKLI et al., 2018). This excluded the costs of a PC because it was only used as a development tool and not part of the device. The initial findings seemed to indicate the feasibility of creating an affordable EOG communication device and suggest its promising potential as a mode of communication. The design described was a proof of concept of an EOG device, and as such, it had limitations.

3.2 Design specifications

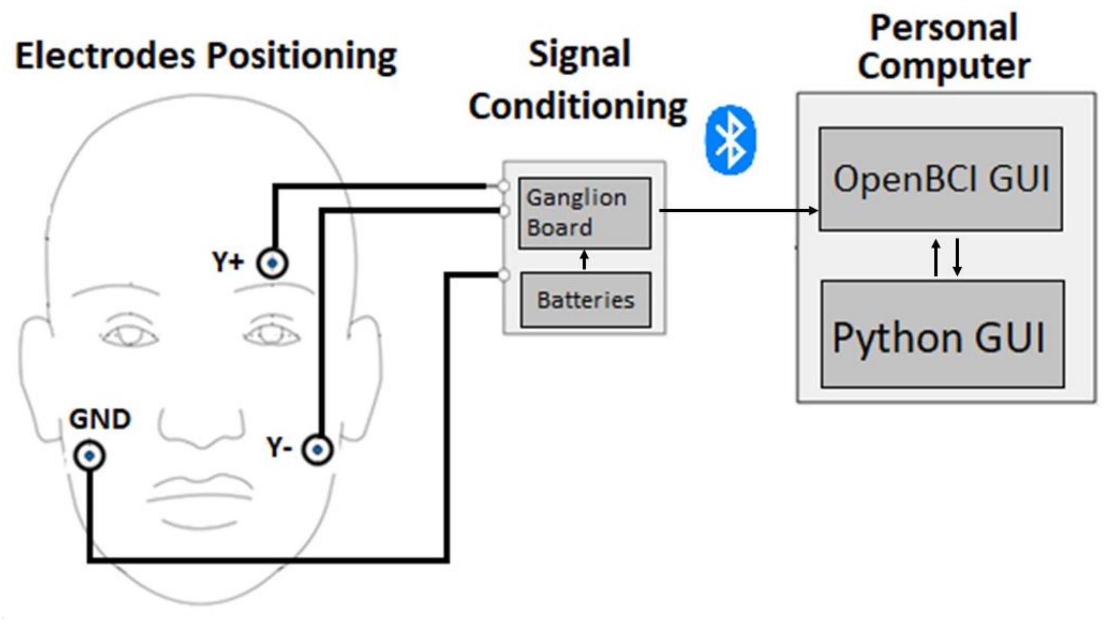
During the development of the project, meetings with the Biomedical Engineering and the Occupational Therapy teams of the Instituto de Reabilitação Lucy Montoro were conducted to discuss patients' needs and define the specifications of the project. The Occupational Therapy team reported that interfaces that depended on the user selection in synchrony with the target button were well accepted by patients despite not having previous experience with interfaces based on eye blinking.

During this stage, we studied the case of a patient with severe mobility and speech impairments but with intact head movements. To communicate, she used a laser pointer fixed to her eyeglasses and head movements to point out letters and numbers on paper. In addition to this method, she had the software on her computer (Camera Mouse, Boston College, USA) that detected the tip of her nose by using a webcam and served as the mouse cursor. The patient could select items using either a cursor dwell timing or an enter button press on the PC. However, she reported neck fatigue when using either of these methods for an extended period. The United Nations CRPD encourages the participation of individuals with disabilities during the early stages of development so that the technology becomes accessible and possibly available at a minimum cost.

3.3 Proposed System

Based on the literature review, pilot study, and feedback from the IRLM team, we proposed the communication shown in Figure 30 (COSTA et al., 2022). This system uses the user’s eye blink, in synchrony with the flash of the target button, to control a GUI. The development of the system was divided into three stages: the design of the acquisition system, the creation of a blink detection algorithm, and the development of a GUI. The system is used to acquire, filter, and process the EOG signals to detect eye blinks, which are then translated into commands of a GUI running on a PC.

Figure 30 - EOG-based system representation



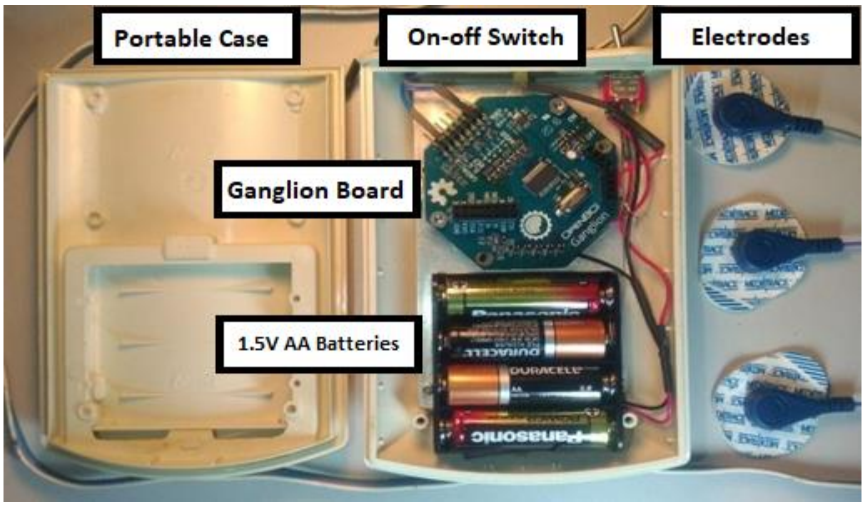
Source: Author.

3.3.1 Data Acquisition

In this study, EOG signals were acquired and preprocessed using a customized EOG device (Figure 31). The blink EOG acquisition involved positioning three commercial adhesive silver chloride electrodes (Ag/AgCl), as shown in Figure 30: the EOG signal on the forehead (+Y above the left eyebrow), the reference electrode on the left mastoid (-Y), and the ground on the right mastoid (GND). The signal acquisition was accomplished using a commercial electronic board (Ganglion, OpenBCI, USA), which transmitted the data via Bluetooth to a PC.

The electrodes were connected to the input of the instrumentation amplifier (AD8237) of the Ganglion board. Figure 31 shows the electrodes connected to the channel 1 amplifier and the D_G pin of the board. The output of the AD8237 was connected to an analog high-pass filter of 0.3 Hz and a 24-bit delta-sigma ADC. After being filtered, the signals were converted by using the 24-bit ADC, sampled at 200 Hz, and transmitted through an embedded low-energy Bluetooth 4.0 module. Additionally, we mounted the board in a portable case and connected to four 1.5 V AA batteries that provided approximately 160 h of autonomy (considering 15 mA board consumption when streaming data). The portable case also had an externally mounted switch that allowed turning on/off the board when the lid was closed.

Figure 31 - Photo of the acquisition device

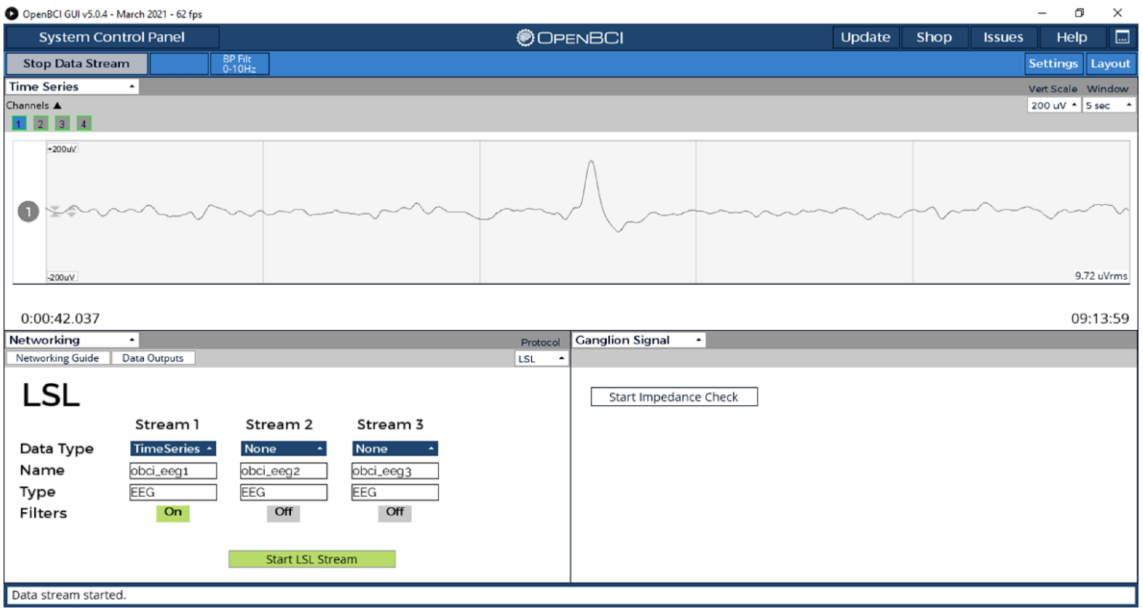


Source: Author.

The Ganglion board was compatible with free open-source software (OpenBCI GUI, OpenBCI, USA). The software interface is illustrated in Figure 32, which shows the time series of the acquired signal, the electrode impedance, and the transmission widget. We used this software to receive EOG data from the Ganglion board, apply digital filters, and transmit the data to the GUI developed specifically for this study. In addition, we modified the source code of the OpenBCI GUI to include a 2nd-order Butterworth filter with a low cutoff frequency of

0.1 Hz and a high cutoff frequency of 10 Hz. This filter was implemented using Brain Flow, which is a library created for EEG, EMG, and ECG signal processing.

Figure 32 - The OpenBCI GUI



	Stream 1	Stream 2	Stream 3
Data Type	TimeSeries	None	None
Name	obci_eeg1	obci_eeg2	obci_eeg3
Type	EEG	EEG	EEG
Filters	On	Off	Off

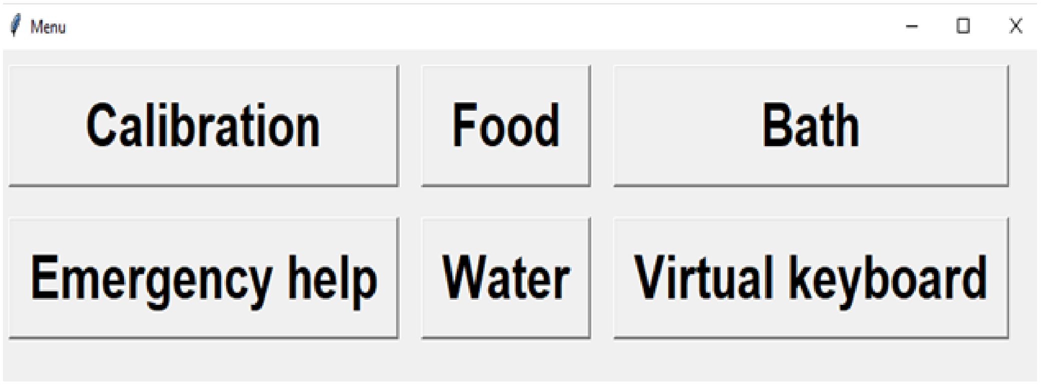
Source: Author.

The networking widget used the Lab Streaming Layer (LSL) protocol to transmit the EOG signal between the OpenBCI GUI and the GUI developed in this study. The LSL comprises a set of libraries and tools for real-time data streaming. Figure 32 shows that we configured the widget to stream the data from Channel 1 and to apply the created band-pass filter. A test routine (Appendix A) was executed to validate the network functionality and to confirm that the EOG data were correctly received. The program recorded streamed data for 2 s and then calculated the valid samples and the average sampling rate. The test results showed that the average rate of valid samples remained close to 200 Hz and did not compromise the EOG acquisition.

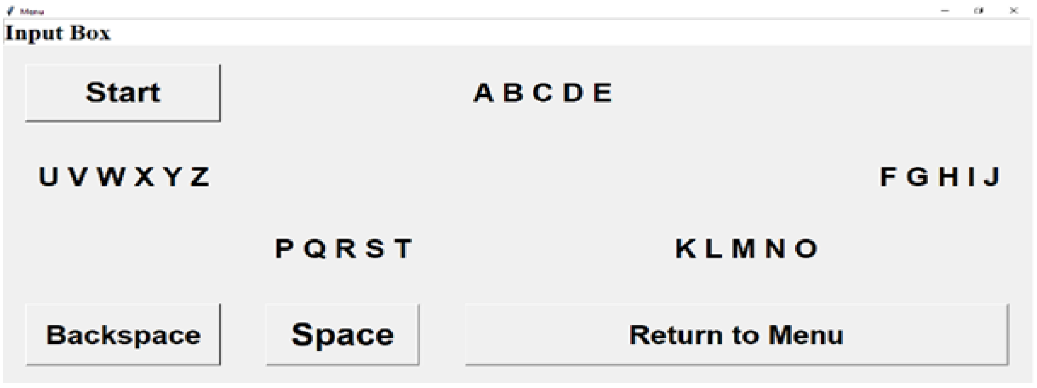
3.3.2 Development of the Graphical User Interface

The GUI developed is shown in Figure 33. First, it displays the main menu (Figure 33A), with options for the user to communicate his/her needs, such as food, bath, water, and emergency help. Additionally, there is an option of a writing application that uses a virtual keyboard illustrated in Figure 33B and Figure 33C.

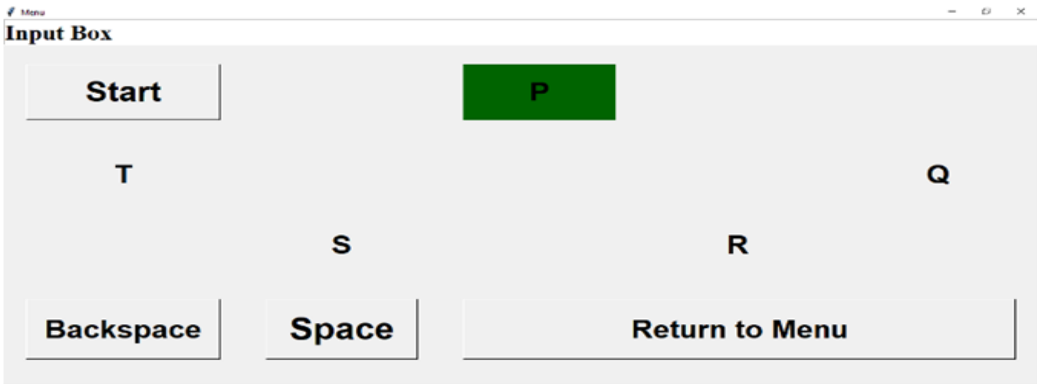
Figure 33 – GUI proposed: (A) main menu (B - C) virtual keyboard



(A)



(B)



(C)

Source: Author.

When the interface is running, it sequentially flashes its buttons in green, and the user can select the target button by synchronously blinking with the flash. The GUI also works in asynchronous mode, in which the user chooses a convenient time to select a target button (selfpaced) instead of having the algorithm set the time to select each operation — a synchronous mode program.

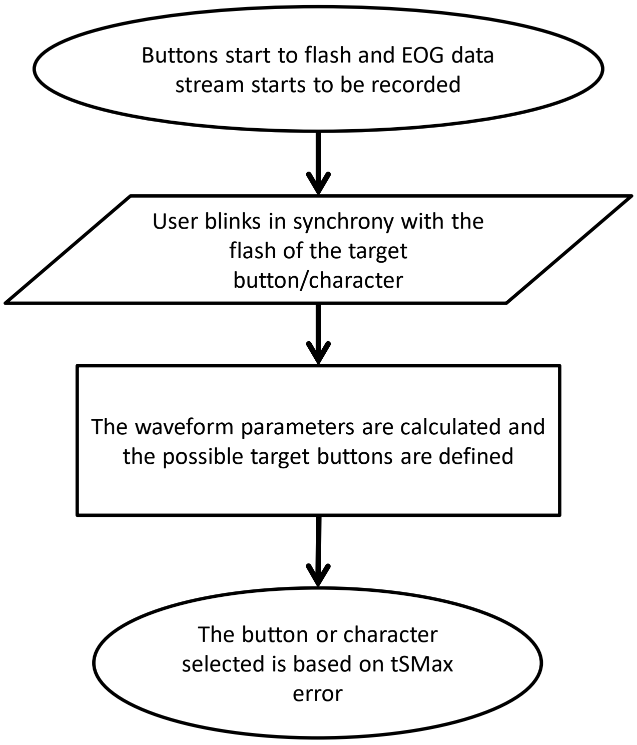
The GUI was developed in Python 3.8 (code in Appendix B), using the Eclipse Integrated Development Environment (IDE). The GUI software was programmed to communicate with the OpenBCI software, to control the graphical interface, and to run the blink detection algorithm. We used the Tkinter library for the interface design, and the PyLSL library to receive the EOG stream via the LSL protocol.

3.3.3 Blink detection and button selection algorithm

The blink detection implemented in this study was based on previous works (HE et al., 2020; HE; LI, 2017; HUANG et al., 2018; XIAO; QU; LI, 2019). This method was chosen because it was a more straightforward approach than other complex techniques, such as the SVM technique and other pattern recognition methods, which still produced comparable results.

Figure 34 shows the flowchart of the button selection algorithm. The interface contains buttons that flash in sequence, and the user must blink synchronously with the flash of the desired button to execute a command. When the selection routine starts, each button flashes in a sequence, with 100 ms of flash duration and 250 ms of the interval between the flashes of each adjacent button.

Figure 34 - Flowchart of button selection algorithm


Buttons start to flash and EOG data stream starts to be recorded
User blinks in synchrony with the flash of the target button/character
The waveform parameters are calculated and the possible target buttons are defined
The button or character selected is based on tSMax error

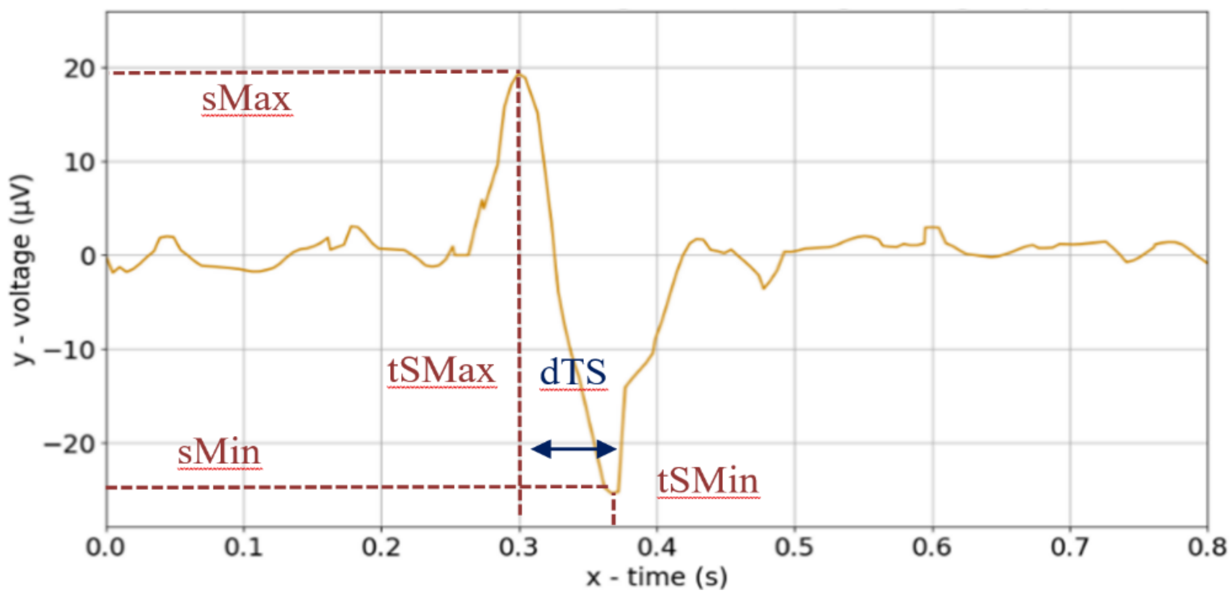
Source: Author.

Simultaneously with the start of the flash sequence, the recording of the EOG data stream starts, and it is divided into multiple segments of 800 ms. Each extracted segment refers to the flash of a different button and is composed of two vectors: the EOG signal and its respective timestamp.

Next, each EOG segment is differentiated, and a moving average filter of three samples is applied. Figure 35 shows the resulting waveform used to verify whether the user blinked. The algorithm then calculates the waveform parameters described below and illustrated in Figure 35.

a) sMax: the maximum value of the differentiated EOG signal;
b) tSMax: the time that sMax occurs;
c) sMin: the minimum value of the differentiated EOG signal, within the period of 125 ms after sMax occurs;
d) tSMax: the time that sMax occurs;
e) tSMin: the time that sMin occurs;
f) dTS: the time difference between tSMax and tSMin.

Figure 35 – Parameters: (A) EOG blink segment (B) parameters of the differentiated EOG blink



Source: Author.

The detection of eye blink occurs when the parameters read from the differentiated and filtered EOG waveform reach a threshold obtained in the calibration: SmaxThreshold, DTSLTheshold, DTSHThreshold, tpSmaxLThreshold, and tpSmaxHThreshold. The successful blink detection must satisfy the following inequalities (Equation 03):

$$\begin{gathered} sMax \geq SMaxThreshold \\ TSMaxHThreshold \geq tSMax \geq TSMaxLThreshold \\ DTSHThreshold \geq dTS \geq DTSLThreshold \end{gathered} \tag{3}$$

However, as the time between the flashes of the adjacent buttons is very short, the program may detect more than one button selection in one round of button flashes. Therefore, there is a final step called decision making, in which the button selected is the one that presents the smallest error (Equation 04) between tSMax and TSMaxMean.

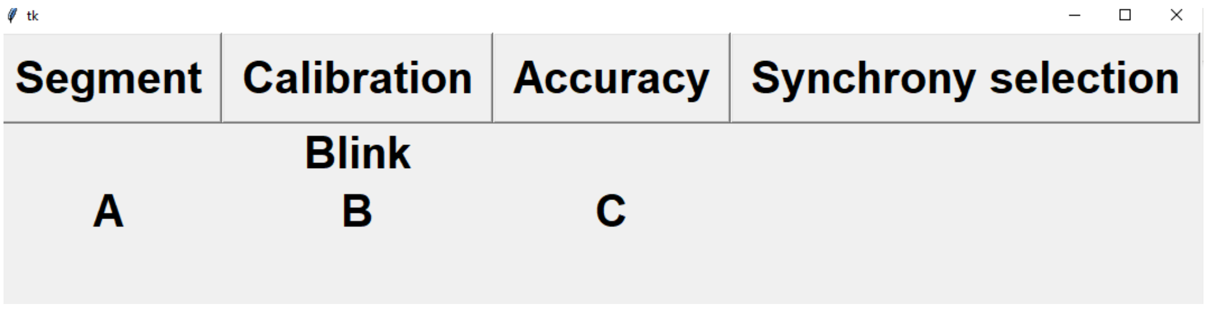
$$e = |tSMax - TSMaxMean| \tag{4}$$

Once a button is selected as the target, all buttons stop flashing, and the respective command is executed. However, if there is no candidate button, the buttons start flashing again (after a short delay). The system then proceeds to the next stage of detection and decision making.

3.3.4 Protocol for testing the interface

To test the described method of blink detection and button selection, we developed a testing interface (Figure 36) in Python 3.8 (code in Appendix C). The GUI includes four buttons ("Segment", "Calibration", "Accuracy", and "Synchrony Detection") that must be selected by using a computer mouse. After the button is selected, the interaction is performed only by blinking. The detailed function of each button is described below:

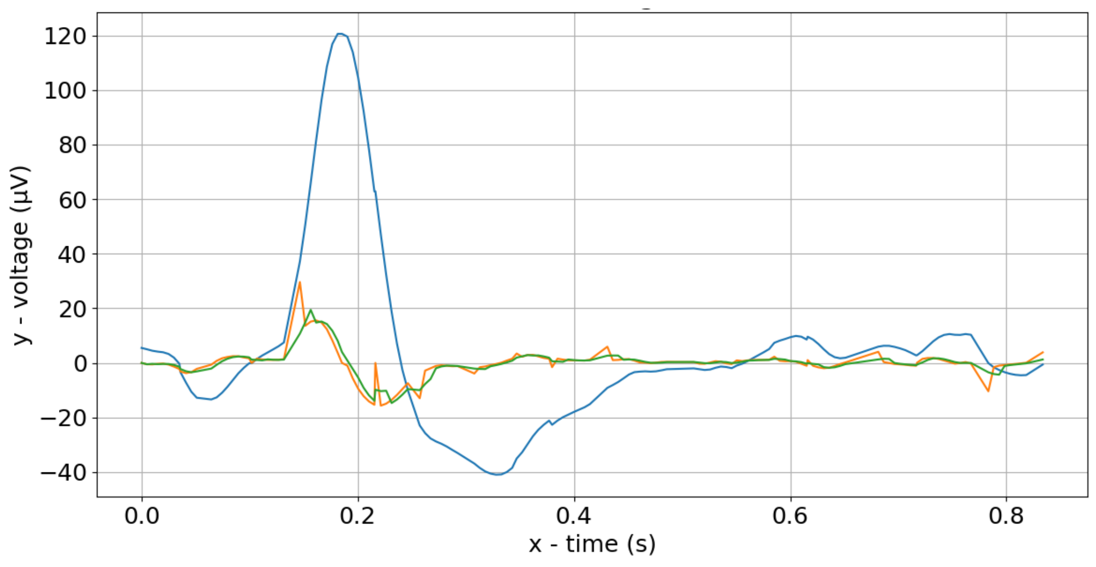
Figure 36- Testing GUI for calibration, accuracy measurement and character selection



Source: Author.

a) **segment**: this button was created to plot a segment of the blink EOG signal. When this button is selected, the program starts to record the EOG data stream, and the word "Blink" flashes for 100 ms. The user must blink synchronously with the button flash. After 800 ms of EOG acquisition, the algorithm calculates the first-order difference and then applies a moving average filter of three samples to remove possible Ganglion transmission errors. The program then plots the blink segment, as shown in Figure 37;

Figure 37 - Blink EOG Segment



Blue line: EOG signal (filtered with 0.1 to 10 Hz band-pass filter); Orange Line: differentiated EOG signal; Green Line: moving average applied to the differentiated EOG signal. Source: Author.

b) **calibration:** this button runs the calibration routine that must be executed before the user starts controlling the GUI because it sets the parameters thresholds. Similar to the segment button routine, the word “Blink” flashes, and the system records the blink EOG signal. However, the calibration routine repeats 20 times, every 2 s, with one break in the middle to rest (after 10 flashes). This results in 20 segments of the differentiated and filtered EOG signal recorded for each flash. The calibration program collects the parameters of the 20 blink segments, and the blink segments with dTS, sMax, and tSMax parameters smaller than $\mu - 2\sigma$ or larger than $\mu + 2\sigma$ are removed (μ represents the mathematical expectation and σ is the standard deviation). Next, the remaining collected segments are used to calculate the thresholds, as described below (Equation 5).

$$\begin{gathered}
\text{SMaxThreshold} = \mu(\text{sMax}) - 2\sigma(\text{sMax}) \\
\text{DTSLThreshold} = \mu(\text{dTS}) - 2\sigma(\text{dTS}) \\
\text{DTSHThreshold} = \mu(\text{dTS}) + 2\sigma(\text{dTS}) \\
\text{TSMaxLThreshold} = \mu(\text{tSMax}) - 2\sigma(\text{tSMax}) \\
\text{TSMaxHThreshold} = \mu(\text{tSMax}) + 2\sigma(\text{tSMax}) \qquad (5)
\end{gathered}$$

c) **accuracy**: this button activates the accuracy routine. This routine is similar to the calibration in the sense that the user has to blink in synchrony with the flash of the word "Blink". However, unlike the calibration process, there is no interruption in the sequence of 20 button flashes. The program then uses the thresholds obtained during calibration to determine the number of times the user blinked during the 20 button flashes. After the 20 button flashes, the program displays the number of blinks that it accurately detected;
d) **synchrony selection:** this mode executes the button selection algorithm. When this button is pressed, the three letters ("A", "B", or "C") flash in sequence, and the user must blink in synchrony with the flash of the desired letter. This mode was created as a preliminary test for the blink selection algorithm (described in section 3.3.3), and the results confirmed the feasibility of this method. The software was then expanded to a complete virtual keyboard, as described in section 3.3.2. The synchrony selection mode was not used in the system assessment described below (section 3.4).

3.4 System performance assessment

To assess the effectiveness of the system designed, we conducted online tests involving both volunteers without disabilities and volunteers with disabilities from the IRLM, following a clinical protocol that we submitted to the Ethics Committee of the Clinics Hospital of the University of Sao Paulo Medical School (CAAE: 62589722.3.0000.0068 and ethical committee approval number: 5.647.107). Prior to participation, all subjects signed a consent form for both the experiments and the publication of individual information.

Table 1 outlines the main steps performed by each subject during the study. For the tests, we used both the testing interface (described in section 3.3.4) and the proposed GUI (section 3.3.2). Initially, the participants received instructions and an overview of the system. Next, the skin area where the electrodes were to be placed was cleaned by using alcohol swabs, and the electrodes were positioned as shown in Figure 30. After a brief waiting period for the electrode-skin interface to stabilize, we checked if the impedance was below 10 kΩ and verified the data stream by using the "segment" button of the testing interface. If the software correctly acquired the EOG data, the participant proceeded to the calibration session for threshold determination. Subsequently, the participants underwent an accuracy test to confirm whether the system could detect their blinks by using the obtained thresholds. Subjects were then given time to familiarize themselves with the system, and finally, they were asked to write a predefined text.

Table 1 - Steps of tests with subjects

Steps	Description
01	Subject receives instructions and an overview of the system
02	Electrode placement, wait for the electrode-skin interface stabilization
03	Check impedance and verify stream of data (both OpenBCI GUI and testing interface)
04	Calibration (testing interface or proposed GUI)
05	Accuracy Test (testing interface)
06	System familiarization (proposed GUI)
07	Write a predefined text (proposed GUI)
08	Remove electrodes

Source: Author.

After applying the described steps, we used the NASA-TLX protocol (HART; STA, 1988) to evaluate our system. This protocol, widely recognized and cited in over 5500 studies, was developed by a NASA research group to assess the subjective workload and satisfaction of tasks or systems. The method employs a multidimensional analysis based on six factors: mental demand (MD), physical demand (PD), temporal demand (TD), own performance (OP), effort (EF), and frustration level (FR). Each factor is scored on a scale of 0 to 100, which are used to calculate the final task workload index. Higher NASA-TLX scores and final indexes indicate greater subjective workload. However, in this workload analysis, it is important to consider not only the final index but also variations in the scores of different factors (HART; STA, 1988).

An example of the NASA-TLX application form is included in Attachment A, and an adapted free translation version is available in Attachment B (applied version). The individuals without disabilities were asked to complete the protocol, whereas the IRLM participants answered during an interview. One IRLM participant answered the interview by speaking, whereas the other two used alternative communication methods.

The basic safety standards were applied during the tests. According to the IEC 60601-1 standard, the maximum direct current allowed under normal conditions is 10 µA and under a single fault condition is 50 µA (LÓPEZ et al., 2020). However, the current that can flow through the subject is far below 10 µA because of the high input impedance (100 MΩ) of the instrumentation amplifiers in the input channels of the Ganglion board. Moreover, the Ganglion is equipped with an isolated power supply of only 6 V, which means that any single fault condition will provide a current significantly lower than 50 µA.

3.4.1 Application of the tests in subjects without disabilities

Two online tests were conducted with the subjects without disabilities. Eight subjects without disabilities participated in Test I. Next, six of the eight subjects without disabilities participated in Test II. The tests were performed as follows.

Test I: We applied the accuracy routine of the testing interface to assess the blink detection. Accuracy was determined by dividing the number of blink detections by the number of flashes.

Test II: Participants interacted with the virtual keyboard to spell the name "Pedro Costa". The initial text selection was arbitrary considering the project context; however, we opted not to alter it to maintain consistency across all participants. The total number of operations required to type the name was 21. The following parameters have been widely adopted in previous studies (HE et al., 2020; HE; LI, 2017; HUANG et al., 2018; XIAO; QU; LI, 2019) to assess user performance and system efficiency, and they were used to evaluate our proposed system and method:

a) time — the total time required to spell the name;
b) operations — number of operations generated by the participant;
c) command Ratio — the ratio of the operations issued to the minimum number of operations required;
d) accuracy — the ratio of the number of correctly inputted letters to the total number of inputted letters;
e) response time (RT) — the number of operations issued over the total time required to spell the name.

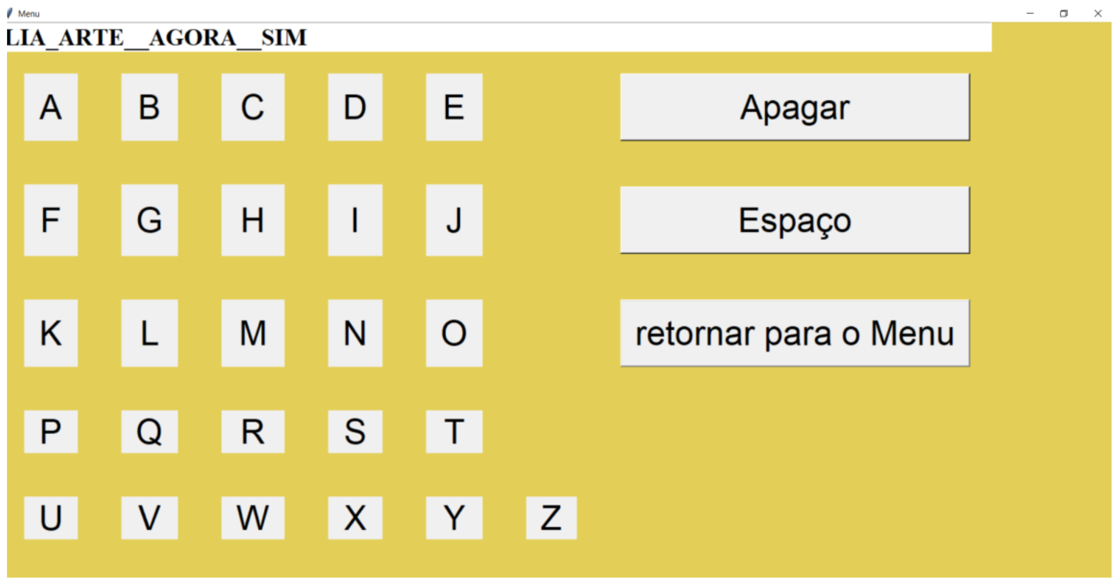
3.4.2 IRLM participants

Four participants from IRLM with severe motor disabilities who were literate volunteers with preserved cognition were selected for the tests. To expand the pool of potential participants, it was not necessary to have communication disabilities. The IRLM also ensured the best accessibility conditions for participants, and the tests preferably occurred before or at the beginning of each scheduled therapy session so that the participants were less fatigued.

The testing procedure for IRLM participants closely mirrored that of subjects without disabilities, with the distinction that IRLM participants underwent at least five sessions of 20 min each, whereas subjects without disabilities participated in a single session. Test I was the same as Test I conducted with subjects without disabilities, but the accuracy was determined as the mean accuracy across these multiple sessions.

In Test II, the IRLM participants interacted with the virtual keyboard. However, two participants required significantly more time to adapt. In one case, 10 sessions and interface adjustments were required to enable this participant to effectively write a full sentence. Customized adjustments (Figure 38) were made to address this participant's specific needs, which significantly improved her system usage. However, these adaptations were not found to enhance the system performance of other participants.

Figure 38 - Adapted system layout



Source: Author.

The adaptations were as follows:

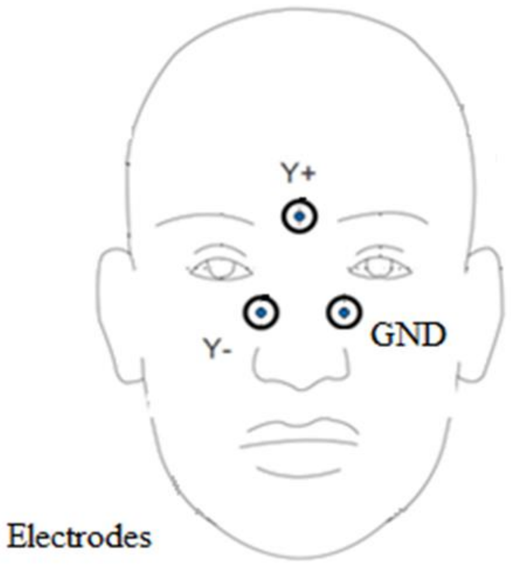
a) The letters layout – the letters were displayed in lines and columns, with all the letters on a single screen;
b) The time between flashes was increased to 900 ms;
c) The font size was increased;
d) The background color was changed to improve the contrast.

In contrast, the remaining two participants were able to start typing during their first session. Over a series of five sessions, these participants were asked to type the predefined sentence: “Treino para comunicar com as pessoas ao meu redor”. This sentence was collaboratively reformulated with the input from a participant from IRLM. It was chosen to contain more characters than the initial test applied to individuals without disabilities and to be a sentence that participants could type within the allotted time, featuring familiar words and holding meaning relevant to the project.

3.5 Tests for a convenient and comfortable acquisition system

To detect eye blinks, the placement of electrodes can vary. In this study, we based our electrode placement on previous studies (HUANG et al., 2018; XIAO; QU; LI, 2019). However, we also tested a different electrode placement, as shown in Figure 39, to provide a more convenient and comfortable acquisition system that would enhance real-life use. We used silver chloride electrodes (Ag/AgCl), with +Y located on the glabellar area (EOG signal), -Y on the right nose pad (reference electrode), and ground (GND) on the left nose pad. The impedance of the electrodes was maintained below 10 kΩ.

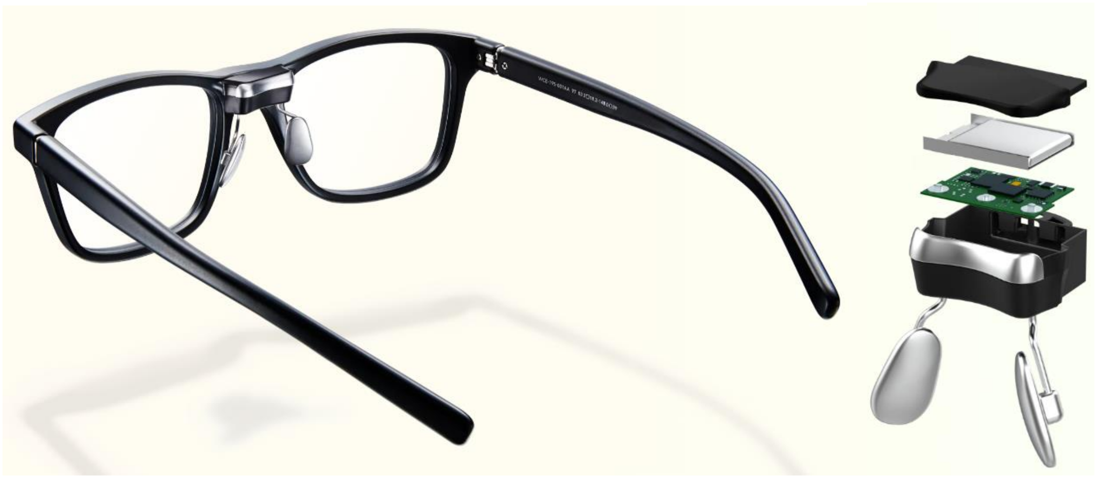
Figure 39 - Electrode position for eyeglasses assessment



Source: Author.

The inspiration for the new electrode positioning test came from the electrode placement of the commercial EOG eyeglasses (JINS MEME, JINS Inc., Japan), as shown in Figure 40. This eyewear is equipped with an EOG sensor that can detect both horizontal and vertical eye movements. The electrodes on JINS MEME are situated on the nose pad and glabellar area and are made of stainless steel (SUS316L).

Figure 40 - JIMS MEME eyeglasses



Source: Jins Inc. (2022).

4. RESULTS AND DISCUSSION

This section presents and discusses the test results for our blink EOG communication system. The results are divided into those of subjects with no disabilities and those of participants from the IRLM. The results are then compared with those of previous studies, and the limitations of our research are discussed. Lastly, we present the results of the preliminary evaluation of a new EOG acquisition system.

4.1 Tests with subjects without disabilities

Table 2 summarizes the results of Test I (accuracy test). The average blink detection accuracy was 89.38 ± 9.5%, which was based on the number of blink detections of 20 button flashes.

Table 2 - Test I (accuracy test) Results

Subject	Number of Flashes	Number of Blink Detections	Accuracy (%)
S1	20	20	100
S2	20	18	90
S3	20	15	75
S4	20	15	75
S5	20	19	95
S6	20	19	95
S7	20	20	100
S8	20	17	85
Average ± Standard Deviation	20	17.87 ± 1.89	89.38 ± 9.5

Source: Author.

Table 3 - Test II Results (use of virtual keyboard)

Subject	Time (s)	Operations	Command Ratio (%)	Accuracy (%)	RT (s)
S1	120	23	109.52	100	5.22
S2	252	25	119.05	84.62	10.08
S3	330	26	123.81	100	12.69
S4	360	41	195.24	78.57	8.78
S5	169	25	119.05	91.67	6.76
S6	180	30	142.86	84.62	6.00
Average ± Standard Deviation	235.17 ± 87.12	28.33 ± 6.05	134.92 ± 28.79	89.91 ± 8.07	8.26 ± 2.58

Source: Author.

Table 3 presents the results of Test II. All the participants completed the spelling task successfully. The average time for the subjects to spell the name was 235.17 ± 87.12 s, whereas the average RT to issue a command was 8.26 ± 2.58 s. The average number of operations required to spell the name was 28.33 ± 6.05, which resulted in an average command ratio of 134.92 ± 28.79% and an average accuracy of 89.91 ± 8.07%. The results of Test I demonstrated that the blink detection algorithm successfully detected user blinking, which enabled the subjects to accomplish the spelling task of Test II with high accuracy. In addition, the accuracy achieved was similar to that obtained in previous studies (HE; LI, 2017; LEE et al., 2017; XIAO; QU; LI, 2019).

Previous studies (HE et al., 2020; HE; LI, 2017; LEE et al., 2017; XIAO; QU; LI, 2019) reported that blinking in synchrony with only one button flash may be challenging due to the extremely short stimulus onset asynchrony (SOA). For example, (HE; LI, 2017) set the SOA to 30 ms and the flash duration to 100 ms, which resulted in an overlap between the two adjacent flashes. In our study, the SOA was approximately eight times greater (250 ms), with no overlap between flashes. This significantly alleviated the time pressure without compromising the RT.

Many EOG systems are synchronous and may not be suitable for practical applications. We used an asynchronous interface, which does not impose a fixed time frame for selecting a target button. Instead, a button was chosen when the user generated a control signal, enabling the user to rest while performing the spelling task.

Table 4 - Workload Assessment Results

Subject	MD	PD	TD	OP	FR	EF	NASA-TLX
S1	10	20	30	10	30	20	19.33
S2	80	70	100	20	20	60	63.33
S3	10	10	30	0	20	10	10.66
S4	0	0	30	30	30	0	24.00
S5	60	20	50	10	20	30	40.33
S6	10	10	30	30	20	10	22.66
Average	28.33	21.67	45	16.67	23.33	21.67	30.05
±	±	±	±	±	±	±	±
Standard Deviation	30.23	22.67	25.66	11.05	4.71	19.51	17.30

Source: Author.

Table 4 shows the scores for the six factors and the final index. On average, the scores remained less than 50, which is considered acceptable according to (EITRHEIM; FERNANDES, 2016). Similar to (HUANG et al., 2018), the TD scores remained higher than the other scores. This may be related to the time pressure to blink synchronously with the flash of the desired button. Furthermore, the low values obtained for the OP may suggest that the users were satisfied with their performance, as reported by (HART; STA, 1988).

4.2 Tests with IRLM participants

Table 5 summarizes the results of Test I for the IRLM participants. The average blink detection accuracy was 85.625 ± 10.17 %, which was based on the number of blink detections of 20 button flashes.

Table 5 - Test I for the IRLM participants

Subject	Number of Flashes	Number of Blink Detections	Accuracy (%)
S1	20	16.25	81.25
S2	20	14.25	71.25
S3	20	18.5	92.5
S4	20	19.5	97.5
Average ± Standard Deviation	20	17.12 ± 2.03	85.625 ± 10.17

Source: Author.

Table 6 presents the results of Test II. Since the IRLM participants were unable to spell the same text, it was not possible to calculate the mean time and the mean number of operations. However, for the command ratio, accuracy, and RT, the mean was calculated because it was not linked to the text size. Therefore, the average command ratio was 116.19 ± 9.32 %, the average accuracy was 91.97 ± 5.69 %, and the average RT was 9.79 ± 1.94 s.

Table 6 - Test II for the IRLM participants

Subject	Time (s)	Operations	Command Ratio (%)	Accuracy (%)	RT (s)
S1 - A	300	35	106,06	90,00%	8,57
S2 - P	1200	95	110,53	91,23%	12,63
S3 - E	1108	117	123,16	86,67%	9,47
S4 - C	68	8	125,00	100,00%	8,50
Average ± Standard Deviation			116.19 ± 9.32	91.97 ± 5.69	9.79 ± 1.94

Source: Author.

Although there was a discrepancy in the spelling task, the results in Table 6 indicate that the IRLM participants showed similar accuracy in blink detection, speller use, and response time. For blink detection, the participants' accuracy was calculated as the average of five tests, which further supported the high accuracy of the blink detection algorithm.

Table 7 - Workload Assessment Results for the IRLM Participants

Subject	MD	PD	TD	OP	FR	EF	NASA-TLX
S1 – Ana	80	20	50	60	20	30	49,33
S2 – Patricia	0	80	0	90	0	50	46,00
S3 - Elisson	20	40	30	50	20	40	13,33
Average ± Standard Deviation	33.33 ± 33.99	46.67 ± 24.94	26.67 ± 20.55	66.67 ± 17.00	13.33 ± 9.42	40 ± 8.16	36.22 ± 16.24

Source: Author.

Table 7 shows the results of the NASA-TLX protocol for the IRLM participants. The average and deviation of the NASA-TLX were comparable to the values obtained for subjects without disabilities; thus, it is also considered acceptable. However, the TD scores were lower, and the OP scores were higher than those obtained for subjects without disabilities. This may suggest that the IRLM participants felt less time pressure to write, but at the same time they felt less satisfied with their performance.

4.3 Comparison with results reported in the literature

For further evaluation, we compared the results of the present study with those of previous studies on speller systems, as shown in Table 4. The information transfer rate (ITR) of our system was determined based on the average accuracy (P), the average time to select a character (T), and the number of possible user commands (N) (Equation 6) (HE; LI, 2017).

$$ITR = 60 \times \frac{\log_2 N + P\log_2 P + (1-P)\log_2\left(\frac{1-P}{N-1}\right)}{T} \quad (6)$$

From Table 8, we may conclude that the system operation is robust with lower processing complexity and power requirements, which are adequate for real-life use. For example, (LÓPEZ et al., 2017) required a more complex processing unit and four different types of eye movements to interact with the interface, yet their performance was similar to our system. In (HE; LI, 2017), the study combined the SVM technique and a waveform detection method, but the results were similar to those of our study. The studies that presented a considerable difference in performance (DURAISAMY; REDDY, 2021; SARAVANAKUMAR; RAMASUBBA REDDY, 2020) required an additional control signal (SSVEP) that may not be convenient for real-life use.

Table 8 - Comparison of the Proposed System with Other Studies

Reference	Control Signal	Number of Possible Commands	RT (s)	Accuracy (%)	ITR (bits/min)
(LÓPEZ et al., 2017)	EOG	64	5.55	98	20
(HE; LI, 2017)	EOG	40	6.07	93.02	45.83
(POSTELNICU; TALABA, 2013)	EOG-P300	72	12.7	90.62	18.28
(SARAVANAKUMAR; RAMASUBBA REDDY, 2020)	SSVEP-EOG	23	3.70	96.73	76.02
(DURAISAMY; REDDY, 2021)	SSVEP-EOG	40	2.70	99.38	116.58
Our system (subjects without disabilities)	**EOG**	**29**	**8.26**	**89.91**	**28.34**
Our system (IRLM Participants)	**EOG**	**29**	**9.79**	**91.97**	**24.94**

Source: Author.

4.4 Limitations of the EOG system developed and its application

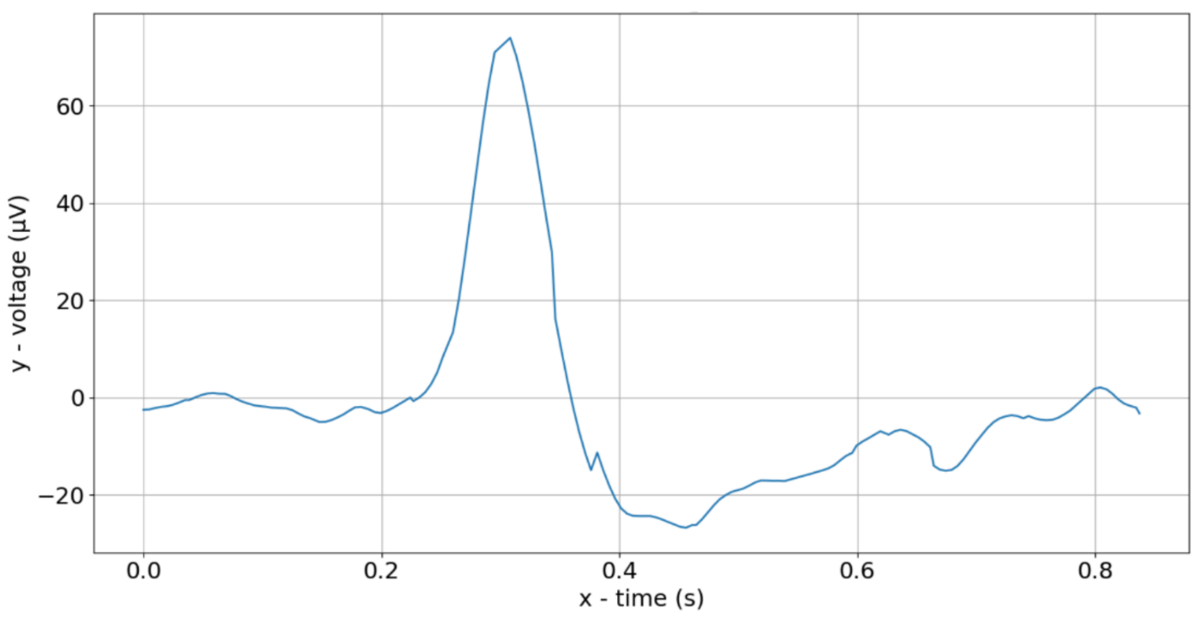
The system described in this study has some limitations. Although it is an asynchronous interface, the time between the flashes limits the speed of interaction. The time between the flashes was set to 250 ms, and if the user wanted to change the pace, the programmer needed to change it for him/her. For example, during the experiments with the Lucy Montoro participants, the time between flashes was increased for adaptation. The proposed system also requires training and time for the users to adapt. The time and extent of adaptation varied significantly among the users. Some participants began writing with high accuracy during the first test session, whereas others took several sessions. Lastly, the size of the electrodes and the discomfort caused by the wires could be improved.

The tests also had limitations. Only four IRLM participants were selected to test the proposed system, and the time spent with them was mostly restricted to a maximum of 40 min. This is a short time for the required tasks: calibration, accuracy test, adaptation, and writing test. One participant could spend more time on the tests during the sessions, whereas the other could participate in 12 sessions. However, the remaining two participants had only five sessions of interaction with the system.

4.5 Results for tests of the acquisition system

The new EOG positioning tends to produce a waveform similar to that of the first electrode positioning, as illustrated in Figure 41. However, the new positioning showed a greater amplitude for the same blink intensity. Although dry electrodes were not used, as in the JINS MEME, these findings suggest the potential miniaturization and portability.

Figure 41 - EOG segment for the new electrode position



Source: Author.

The development of a device such as JINS MEME would enable real-life use. In Brazil, a similar device has been recently launched (Colibri, TiX Tecnologia Assistiva, Brazil). It utilizes an infrared light sensor and an accelerometer to detect eye blinks and head movements, respectively. This device can function as a head computer mouse and is often used with spelling interfaces. Moreover, Colibri (TIX TECNOLOGIA ASSISTIVA INC., 2021) has addressed a crucial issue for this type of technology: maintenance and technical support. They offer a price for customers to purchase the glasses, but also provide a significantly lower price (25 times less) for users to rent the product and receive all necessary support, including replacements, if required. An EOG eyeglasses device could benefit from the same business model.

4.6 Future works

Future works may include the study of evaluation of user performance with a larger number of tests in individuals with disabilities. This would result in a more robust conclusion about the effectiveness of the interface. The next step may also involve integrating a light and flexible wireless EOG acquisition system to enhance comfort and mobility in real-life situations.

5. CONCLUSIONS

This dissertation describes the development and evaluation of an asynchronous single-channel EOG interface for people with mobility disabilities who cannot communicate by speaking, writing, or gesturing. The study initially explored the use of EOG signals as a modality for communication and verified the feasibility of an EOG-based communication device. This opened discussions with the IRLM Team, which led us to investigate the use of eye blinks in synchrony selection as a method of communication. As a result, we developed an EOG blink communication system with a GUI and evaluated it with both subjects without disabilities and with disabilities at the IRLM.

The evaluation results showed high accuracy for both the blink detection algorithm and the proposed speller. Compared to previous research, we achieved lower processing complexity and power requirements while maintaining robust performance. However, the study had some limitations, including a limited maximum speed of interaction, the need for training and adaptation that varied between users, and the small number of participants from the IRLM.

This study is expected to contribute to an assistive technology for people with disabilities by providing a practical and comfortable design for everyday use. Future work may include the study of change in user performance with the number of tests, as well as improvements to the design to enhance comfort and mobility in real-life situations. This may involve incorporating a light and flexible wireless EOG acquisition system.

REFERENCES[1]

AMERI, Shideh Kabiri; KIM, Myungsoo; KUANG, Irene Agnes; PERERA, Withanage K.; ALSHIEKH, Mohammed; JEONG, Hyoyoung; TOPCU, Ufuk; AKINWANDE, Deji; LU, Nanshu. Imperceptible electrooculography graphene sensor system for human–robot interface. **npj 2D Materials and Applications**, *[S. l.]*, v. 2, n. 1, 2018. DOI: 10.1038/s41699-018-0064-4.

BAREA, Rafael; BOQUETE, Luciano; MAZO, Manuel; LÓPEZ, Elena. System for assisted mobility using eye movements based on electrooculography. **IEEE Transactions on Neural Systems and Rehabilitation Engineering**, *[S. l.]*, v. 10, n. 4, p. 209–218, 2002. DOI: 10.1109/TNSRE.2002.806829.

CAMPBELL, Neil A.; REECE, Jane B. **Biology**. 8th. ed. San Francisco, CA: Benjamin Cummings, 2008.

CHANG, Won-Du. Electrooculograms for Human – Computer Interaction : A Review. **Sensors**, *[S. l.]*, v. 19, p. 2690, 2019. DOI: 10.3390/s19122690.

CHEN, Yingxi; NEWMAN, Wyatt S. A human-robot interface based on electrooculography. *In*: PROCEEDINGS - IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION 2004, New Orleans. **Anais** [...]. New Orleans: IEEE, 2004. p. 243–248. DOI: 10.1109/robot.2004.1307158.

COSTA, P. T. L.; JUNIOR, N. B. Rosa; TSUKIMOTO, D. R.; OSHIRO, M. S.; CESTARI, I. A. Proof of concept for the manufacturing of a low-cost electrooculography device. *In*: XXVII BRAZILIAN CONGRESS IN BIOMEDICAL ENGINEERING 2020, Vitória. **Anais** [...]. Vitória p. 594–595.

COSTA, P. T. L.; TSUKIMOTO, D. R.; TAMASHIRO, D. S. U.; BATTISTELLA, L. R.; CESTARI, I. A. A Single-Channel Asynchronous Electrooculography-Based Interface for People with Motor Disabilities. *In*: IX LATIN AMERICAN CONGRESS ON BIOMEDICAL ENGINEERING AND THE XXVIII BRAZILIAN CONGRESS ON BIOMEDICAL

[1] According to Brazilian Association of Technical Standards (ABNT). NBR 6023.

ENGINEERING 2022, Florianópolis. **Anais** [...]. Florianópolis

DEBBARMA, Shibam; BHADRA, Sharmistha. A Lightweight Flexible Wireless Electrooculogram Monitoring System with Printed Gold Electrodes. **IEEE Sensors Journal**, *[S. l.]*, v. 21, n. 18, p. 20931–20942, 2021. DOI: 10.1109/JSEN.2021.3095423.

DEBBARMA, Shibam; BHADRA, Sharmistha. A Flexible Wearable Electrooculogram System With Motion Artifacts Sensing and Reduction. **IEEE Transactions on Biomedical Circuits and Systems**, *[S. l.]*, v. 16, n. 2, p. 324–335, 2022. DOI: 10.1109/TBCAS.2022.3168236.

DENNEY, D.; DENNEY, C. The eye blink electro-oculogram. **British Journal of Ophthalmology**, *[S. l.]*, v. 68, n. 4, p. 225–228, 1984. DOI: 10.1136/bjo.68.4.225.

DOANE, Marshall G. Interaction of eyelids and tears in corneal wetting and the dynamics of the normal human eyeblink. **American Journal of Ophthalmology**, *[S. l.]*, v. 89, n. 4, p. 507–516, 1980. DOI: 10.1016/0002-9394(80)90058-6.

DOYLE, T. E.; KUCEROVSKY, Z.; GREASON, W. D. Design of an electroocular computing interface. *In*: CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING 2006, Ottawa. **Anais** [...]. Ottawa: IEEE, 2006. p. 1458–1461. DOI: 10.1109/CCECE.2006.277758.

DREWES, Heiko (2010). **Eye Gaze Tracking for Human Computer Interaction**. 2010. *[S. l.]*, 2010.

DURAISAMY, Saravanakumar; REDDY, Ramasubba M. Stimulus Paradigm for an Asynchronous Concurrent SSVEP and EOG Based BCI Speller System. **IEEE Access**, *[S. l.]*, v. 9, p. 127484–127495, 2021. DOI: 10.1109/ACCESS.2021.3112257.

EITRHEIM, M. H. R.; FERNANDES, A. The NASA Task Load Index for rating workload acceptability. *In*: HUMAN FACTORS AND ERGONOMICS SOCIETY EUROPE 2016, Prage. **Anais** [...]. Prage

FANG, Fuming; SHINOZAKI, Takahiro. Electrooculography-based continuous eye-writing recognition system for efficient assistive communication systems. **PLOS ONE**, *[S. l.]*, v. 13,

n. 2, p. e0192684, 2018. DOI: 10.1371/journal.pone.0192684.

GUYTON, Arthur C.; HALL, John E. **Guyton and Hall Textbook of Medical Physiology**. 12th. ed. Philadelphia, PA: Saunders Elsevier, 2011.

HANSEN, Dan Witzner; JI, Qiang. In the Eye of the Beholder: A Survey of Models for Eyes and Gaze. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, *[S. l.]*, v. 32, n. 3, p. 478–500, 2010. DOI: 10.1109/TPAMI.2009.30.

HART, Sandra G.; STA, Lowell E. Development of NASA-TLX (task load index): Results of empirical and theoretical research. **Advances in Psychology**, *[S. l.]*, v. 52, n. Human Mental Workload, p. 139–183, 1988. DOI: 10.1016/S0166-4115(08)62386-9.

HE, Shenghong et al. EEG- And EOG-Based Asynchronous Hybrid BCI: A System Integrating a Speller, a Web Browser, an E-Mail Client, and a File Explorer. **IEEE Transactions on Neural Systems and Rehabilitation Engineering**, *[S. l.]*, v. 28, n. 2, p. 519–530, 2020. DOI: 10.1109/TNSRE.2019.2961309.

HE, Shenghong; LI, Yuanqing. A single-channel EOG-based speller. **IEEE Transactions on Neural Systems and Rehabilitation Engineering**, *[S. l.]*, v. 25, n. 11, p. 1978–1987, 2017. DOI: 10.1109/TNSRE.2017.2716109.

HÖMKE, Paul; HOLLER, Judith; LEVINSON, Stephen C. Eye blinks are perceived as communicative signals in human face-to-face interaction. **PLoS ONE**, *[S. l.]*, v. 13, n. 12, p. 1–13, 2018. DOI: 10.1371/journal.pone.0208030.

HORI, Junichi; SAKANO, Koji; SAITOH, Yoshiaki. Development of a communication support device controlled by eye movements and voluntary eye blink. **IEICE Transactions on Information and Systems**, *[S. l.]*, v. E89-D, n. 6, p. 1790–1797, 2006. DOI: 10.1093/ietisy/e89-d.6.1790.

HUANG, Qiyun; HE, Shenghong; WANG, Qihong; GU, Zhenghui; PENG, Nengneng; LI, Kai; ZHANG, Yuandong; SHAO, Ming; LI, Yuanqing. An EOG-based human-machine interface for wheelchair control. **IEEE Transactions on Biomedical Engineering**, *[S. l.]*, v. 65, n. 9, p. 2023–2032, 2018. DOI: 10.1109/TBME.2017.2732479.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA - IBGE. **Características Gerais da População, Religião e Pessoas Com DeficiênciaCenso Demográfico 2010**. Rio de Janeiro.

IRSCH, Kristina; GUYTON, David L. Anatomy of Eyes. *In*: **Encyclopedia of Biometrics**. 1. ed. New York: Springer, 2009. p. 10–16. DOI: 10.1007/978-0-387-73003-5.

IWASAKI, Masaki; KELLINGHAUS, Christoph; ALEXOPOULOS, Andreas V.; BURGESS, Richard C.; KUMAR, Arun N.; HAN, Yanning H.; LÜDERS, Hans O.; LEIGH, R. John. Effects of eyelid closure, blinks, and eye movements on the electroencephalogram. **Clinical Neurophysiology**, *[S. l.]*, v. 116, n. 4, p. 878–885, 2005. DOI: 10.1016/j.clinph.2004.11.001.

JINS INC. **JINS MEME**. 2022. Disponível em: https://jinsmeme.com/. Acesso em: 20 out. 2023.

KANEKO, Kenichi; SAKAMOTO, Kazuyoshi. Evaluation of three types of blinks with the use of electro-oculogram and electromyogram. **Perceptual and Motor Skills**, *[S. l.]*, v. 88, n. 3, p. 1037–1052, 1999. DOI: 10.2466/pms.1999.88.3.1037.

KOSMYNA, Nataliya; MORRIS, Caitlin; SARAWGI, Utkarsh; NGUYEN, Thanh; MAES, Pattie. AttentivU: A Wearable Pair of EEG and EOG glasses for real-time physiological processing. *In*: 2019 IEEE 16TH INTERNATIONAL CONFERENCE ON WEARABLE AND IMPLANTABLE BODY SENSOR NETWORKS, BSN 2019 2019, **Anais** [...]. [s.l: s.n.] p. 2–5. DOI: 10.1109/BSN.2019.8771080.

LEE, Kwang Ryeol; CHANG, Won Du; KIM, Sungkean; IM, Chang Hwan. Real-time eye-writing recognition using electrooculogram. **IEEE Transactions on Neural Systems and Rehabilitation Engineering**, *[S. l.]*, v. 25, n. 1, p. 37–48, 2017. DOI: 10.1109/TNSRE.2016.2542524.

LÓPEZ, Alberto; FERRERO, Francisco; VILLAR, José Ramón; POSTOLACHE, Octavian. High-performance analog front-end (AFE) for EOG systems. **Electronics (Switzerland)**, *[S. l.]*, v. 9, n. 6, p. 1–15, 2020. DOI: 10.3390/electronics9060970.

LÓPEZ, Alberto; FERRERO, Francisco; YANGÜELA, David; ÁLVAREZ, Constantina;

POSTOLACHE, Octavian. Development of a computer writing system based on EOG. **Sensors (Switzerland)**, *[S. l.]*, v. 17, n. 7, p. 1–20, 2017. DOI: 10.3390/s17071505.

LUKANDER, Kristian. **Mobile usability - Measuring gaze point on handheld devices**. 2003. HELSINKI UNIVERSITY OF TECHNOLOGY, *[S. l.]*, 2003.

MALMIVUO, Jaakko; PLONSEY, Robert. **Bioelectromagnetism Principles and Applications of Bioelectric**. New York: Oxford University Press, 1995.

MATSUO, Fumisuke; PETERS, Jon F.; REILLY, Edward L. Electrical phenomena associated with movements of the eyelid. **Electroencephalography and Clinical Neurophysiology**, *[S. l.]*, n. 38, p. 507–11, 1975. DOI: 10.1016/0013-4694(75)90191-1.

PERUZZO, Pedro Pulzatto; FLORES, Enrique Pace Lima. The repercussion of the Convention on the Rights of Persons with Disabilities in Brazilian courts. **Revista Direito e Práxis**, *[S. l.]*, v. 12, n. 4, p. 2601–2627, 2021. DOI: 10.1590/2179-8966/2020/47403.

POSTELNICU, C. C.; TALABA, D. P300-based brain-neuronal computer interaction for spelling applications. **IEEE Transactions on Biomedical Engineering**, *[S. l.]*, v. 60, n. 2, p. 534–543, 2013. DOI: 10.1109/TBME.2012.2228645.

PURVES, Dale et Al. **Neuroscience**. 3th. ed. Sunderland MA: Sinauer Associates, Inc., 2004.

SARAVANAKUMAR, D.; RAMASUBBA REDDY, M. **A Brain Computer Interface based Communication System using SSVEP and EOG**. **Procedia Computer Science**, 2020. DOI: 10.1016/j.procs.2020.03.241.

SINGH, Hari; SINGH, Jaswinder. Human Eye Tracking and Related Issues: A Review. **International Journal of Scientific and Research Publications**, *[S. l.]*, v. 2, n. 1, 2012. Disponível em: www.ijsrp.org.

TIX TECNOLOGIA ASSISTIVA INC. **Colibri**. 2021. Disponível em: https://colibrimouse.com/. Acesso em: 2 abr. 2024.

USAKLI, Ali Bulent; GURKAN, Serkan. Design of a novel efficient humancomputer interface: An electrooculagram based virtual keyboard. **IEEE Transactions on Instrumentation and**

Measurement, *[S. l.]*, v. 59, n. 8, p. 2099–2108, 2010. DOI: 10.1109/TIM.2009.2030923.

USAKLI, Ali Bulent; GURKAN, Serkan; GURKAN, Guray; KAYA, Alper. A novel EOG-based wireless rapid communication device for people with motor neuron diseases. **Journal of Medical Engineering and Technology**, *[S. l.]*, v. 42, n. 6, p. 420–425, 2018. DOI: 10.1080/03091902.2018.1531947.

WEBSTER, John G. **Medical Instrumentation**. 4th. ed. Hoboken, NJ: John Wiley & Sons, 2010.

WORLD HEALTH ORGANISATION. World report on disability. **The Lancet**, *[S. l.]*, v. 377, n. 9782, p. 1977, 2011. DOI: 10.1016/S0140-6736(11)60844-1.

XIAO, Jing; QU, Jun; LI, Yuanqing. An Electrooculogram-Based Interaction Method and Its Music-on-Demand Application in a Virtual Reality Environment. **IEEE Access**, *[S. l.]*, v. 7, p. 22059–22070, 2019. DOI: 10.1109/ACCESS.2019.2898324.

YOUNG, Laurence R.; SHEENA, David. Survey of eye movement recording methods. **Behavior Research Methods & Instrumentation**, *[S. l.]*, v. 7, n. 5, p. 397–429, 1975. DOI: 10.3758/BF03201553.

YU, Yang; LIU, Yadong; YIN, Erwei; JIANG, Jun; ZHOU, Zongtan; HU, Dewen. An Asynchronous Hybrid Spelling Approach Based on EEG-EOG Signals for Chinese Character Input. **IEEE Transactions on Neural Systems and Rehabilitation Engineering**, *[S. l.]*, v. 27, n. 6, p. 1292–1302, 2019. DOI: 10.1109/TNSRE.2019.2914916.

APPENDIX A – PYTHON CODE FOR LSL

```
import time
from pylsl import StreamInlet, resolve_stream
from time import sleep

# first resolve an EOG stream on the lab network
print("looking for an EOG stream...")
streams = resolve_stream('type', 'EEG')

# create a new inlet to read from the stream
inlet = StreamInlet(streams[0])
duration = 2

sleep(1)

def testLSLSamplingRate():
    start = time.time()
    totalNumSamples = 0
    validSamples = 0
    numChunks = 0

    while time.time() <= start + duration:
        # get chunks of samples
        samples, timestamp = inlet.pull_chunk()
        if samples:
            numChunks += 1
            print( len(samples) )
            totalNumSamples += len(samples)
            # print(samples);
            for sample in samples:
                print(sample)
                validSamples += 1

    print( "Number of Chunks and Samples == {} , {}".format(numChunks, 
totalNumSamples) )
    print( "Valid Samples and Duration == {} / {}".format(validSamples, 
duration) )
    print( "Avg Sampling Rate == {}".format(validSamples / duration) )


testLSLSamplingRate()
```

APPENDIX B – PYTHON CODE FOR THE PROPOSED SYSTEM

```
# -*- coding: cp1252 -*-
import tkinter
from pylsl import StreamInlet, resolve_stream
import time
import matplotlib.pyplot as plt
from time import sleep
import statistics
import tkinter.font as tkFont

deltaArr = []
samples = []
timeSamples = []
timeStamps = []

timeDiff = []
duration = 0.8
timeFlashes = 60

AmaxVec = []
tpAMaxVec = []
tpSMaxVec = []
SmaxVec = []
SminVec = []
dTSVec = []
avgDeltaArr = []

rodada = 1
piscadas = 0
piscadaErrada = 0

class EOGInterface:
    def __init__(self):
        self.main_window = tkinter.Tk()

        self.main_window.geometry('1300x700')

        self.construcMainMenu()

    def construcMainMenu(self):

        self.main_window.title('Menu')

        self.button = tkinter.Button( self.main_window, text = 'Calibração',
                                      font = ("Arial Bold", 30),)
        self.button.grid(column=0, row=0, sticky ="WENS", pady = 10, padx = 10)

        self.button1 = tkinter.Button( self.main_window, text = 'Alimentação',
                                      font = ("Arial Bold",30),
command=self.sleep)
        self.button1.grid(column=1, row=0, sticky ="WENS", pady = 10, padx = 10)

        self.button2 = tkinter.Button( self.main_window, text = 'Banheiro',
                                      font = ("Arial Bold", 30),
command=self.bath)
```

```
        self.button2.grid(column=2, row=0, sticky ="WENS", pady = 10, padx = 10)

        self.button3 = tkinter.Button( self.main_window, text = 'Emergência',
                                       font = ("Arial Bold", 30),
command=self.emergencyHelp)
        self.button3.grid(column=0, row=1, sticky ="WENS", pady = 10, padx = 10)

        self.button4 = tkinter.Button( self.main_window, text = 'Água',
                                       font = ("Arial Bold", 30),
command=self.emergencyHelp)
        self.button4.grid(column=1, row=1, sticky ="WENS", pady = 10, padx = 10)

        self.button5 = tkinter.Button( self.main_window, text = 'Teclado Virtual',
                                       font = ("Arial Bold", 30),
command=self.open_keyboard)
        self.button5.grid(column=2, row=1, sticky ="WENS", pady = 10, padx = 10)

        tkinter.mainloop()

    def emergencyHelp(self):
            print('Emergency Help')

    def medicalAssistance(self):
            print('Medical Assistance')

    def sleep(self):
        print('Sleep')

    def bath(self):
        print('Bath')

    def flashBlocksSelection(self):

        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration,
timeFlashes
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dTSVec,
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global tpAmaxMean, tpSMaxMean, AmaxThreshold, SmaxThreshold,
dTSLThreshold, dTSHThreshold, tpSMaxLThreshold, tpSMaxHThreshold
        global textOfTextBox

        sleep(1)
        self.flashLetters('Start')
        sleep(0.1)
        self.unFlashLetters('Start')
        sleep(1)

        # first resolve an EOG stream on the lab network
        print("\nlooking for an EOG stream...")
        streams = resolve_stream('type', 'EEG')
        # create a new inlet to read from the stream
        inlet = StreamInlet(streams[0])
        sample, timestamp = inlet.pull_sample()
        while len(sample) == 0:
            samples, timestamp = inlet.pull_sample()
        x1 = sample[0]
        initialTimeStamp = timestamp
        timeStamps.append(0)
        samples.append(x1)
        deltaArr.append(0)
        oldTimestamp = timestamp

        self.flashBlocks(1)

        start = time.time()

        while time.time() <= (start + (timeFlashes*0.07+1)):
            sample, timestamp = inlet.pull_sample()
            if timestamp > oldTimestamp:
                timeStamps.append(timestamp - initialTimeStamp )
                timeSample = time.time() - start
                timeSamples.append(timeSample)
                x2 = sample[0]
                samples.append(x2)
                deltaX = x2 - x1
                deltaArr.append(deltaX)
                x1 = x2
                oldTimestamp = timestamp

                tempoA =  time.time() - start
                if (tempoA >= 0.1 and self.labelABCDE.cget("bg") == "darkgreen"):
                    self.unflashBlocks(1)
                elif ( tempoA >= 0.01*timeFlashes and tempoA < (0.01*timeFlashes +
0.1)  and self.labelFGHIJ.cget("bg")!= "darkgreen"):
                    self.flashBlocks(2)
                elif ( tempoA >= (0.01*timeFlashes + 0.1)  and
self.labelFGHIJ.cget("bg")== "darkgreen" ):
                    self.unflashBlocks(2)
                elif ( tempoA >= 0.02*timeFlashes and tempoA < (0.02*timeFlashes +
0.1) and self.labelKLMNO.cget("bg")!= "darkgreen"):
                    self.flashBlocks(3)
                elif ( tempoA >= (0.02*timeFlashes + 0.1) and
self.labelKLMNO.cget("bg")== "darkgreen" ):
                    self.unflashBlocks(3)
                elif ( tempoA >= 0.03*timeFlashes and tempoA < (0.03*timeFlashes +
0.1) and self.labelPQRST.cget("bg")!= "darkgreen"):
                    self.flashBlocks(4)
                elif ( tempoA >= (0.03*timeFlashes + 0.1) and
self.labelPQRST.cget("bg")== "darkgreen" ):
                    self.unflashBlocks(4)
                elif ( tempoA >= 0.04*timeFlashes and tempoA < (0.04*timeFlashes +
0.1) and self.labelUVWXYZ.cget("bg")!= "darkgreen"):
                    self.flashBlocks(5)
                elif ( tempoA >= (0.04*timeFlashes + 0.1) and
self.labelUVWXYZ.cget("bg")== "darkgreen" ):
                    self.unflashBlocks(5)
                elif ( tempoA >= 0.05*timeFlashes  and tempoA < (0.05*timeFlashes
+ 0.1)and self.backspace.cget("bg")!= "darkgreen"):
                    self.flashLetters('Backspace')
                elif ( tempoA >= (0.05*timeFlashes + 0.1) and
self.backspace.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('Backspace')
                elif ( tempoA >= 0.06*timeFlashes  and tempoA < (0.06*timeFlashes
+ 0.1)and self.space.cget("bg")!= "darkgreen"):
                    self.flashLetters('Space')
                elif ( tempoA >= (0.06*timeFlashes + 0.1) and
self.space.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('Space')
                elif ( tempoA >= 0.07*timeFlashes and tempoA < (0.07*timeFlashes +
0.1) and self.returnMenu.cget("bg")!= "darkgreen"):
                    self.flashLetters('Return')
                elif ( tempoA >= (0.07*timeFlashes + 0.1) and
self.returnMenu.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('Return')

        avgDeltaArr.insert(0,deltaArr[0])
        avgDeltaArr.insert(1,deltaArr[1])
        for i in range(2,len(deltaArr),1):
            a = i-1
            aa = i -2
            value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
            avgDeltaArr.insert(i , value )

        buttons = {}

        for i in range(1,9,1):
            timeStampsIndex = [idx for idx, val in enumerate(timeStamps) if (val >
0.1+(i-1)*0.01*timeFlashes and val < 0.8+(i-1)*0.01*timeFlashes)]
            timeStampsA = [timeStamps[index] for index in timeStampsIndex]
            samplesA = [samples[index] for index in timeStampsIndex]
            deltaArrA = [avgDeltaArr[index] for index in timeStampsIndex]
            delayT = 0.01*timeFlashes*(i-1)
            timeStampsA[:] = [timeStampA - delayT for timeStampA in timeStampsA]

            if(samplesA):
                AmaxA = max(samplesA)
                tpAMaxIndexA = samplesA.index(AmaxA)
                SmaxA = max(deltaArrA)
                tpSMaxIndexA = deltaArrA.index(SmaxA)
                tpSMaxA = timeStampsA[tpSMaxIndexA]
                SminA = min(deltaArrA[tpSMaxIndexA:(tpSMaxIndexA+50)])

                tpMinIndexA = deltaArrA.index(SminA)
                dTSA = timeStampsA[tpMinIndexA] - timeStampsA[tpSMaxIndexA]

            if( (SmaxA > SmaxThreshold) and (dTSA > dTSLThreshold) and (dTSA <
dTSHThreshold) and (tpSMaxA > tpSMaxLThreshold) and (tpSMaxA < tpSMaxHThreshold)):

                if i == 1:
                    buttons["1"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 2:
                    buttons["2"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 3:
                    buttons["3"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 4:
                    buttons["4"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 5:
                    buttons["5"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 6:
                    buttons["Backspace"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 7:
                    buttons["Space"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 8:
```

```
                buttons["Return"] =  abs(tpSMaxA - tpSMaxMean)

            print('\Linha '+ str(i) + ' Pré-Selecionada')
            print('Amax = ' + str(AmaxA))
            print('tpAmax = ' + str(timeStampsA[tpAMaxIndexA]))
            print('tpSmax = ' + str(timeStampsA[tpSMaxIndexA]))
            print('Smax = ' + str(SmaxA))
            print('tpSmin = ' + str(timeStampsA[tpMinIndexA]))
            print('Smin = ' + str(SminA))
            print('dTS = ' + str(dTSA))
            print('Samples = ' + str(len(samplesA)))
            print('deltaArr = ' + str(len(deltaArrA)))

        deltaArrA = []
        samplesA = []
        timeStampsA = []

    selected = ''
    if buttons:

        selected = min(buttons, key=buttons.get)
        print('\n Final Selection ' + selected)
        if selected == 'Return':
            textOfTextBox = self.textBox.get('1.0', 'end')
            deltaArr = []
            samples = []
            avgDeltaArr = []
            timeSamples = []
            timeStamps = []
            timeDiff = []
            self.rebuild_Menu()
            return
        elif selected == "Backspace":
            self.deleteTextChar()
        elif selected == 'Space':
            self.textBox.insert(tkinter.END, '_')
        else:
            self.main_window.update()
            deltaArr = []
            samples = []
            avgDeltaArr = []
            timeSamples = []
            timeStamps = []
            timeDiff = []
            if selected == '1':
                self.open_keyboardABCDE()
            elif selected =='2':
                self.open_keyboardFGHIJ()
            elif selected =='3':
                self.open_keyboarKLMNO()
            elif selected =='4':
                self.open_keyboarPQRST()
            elif selected =='5':
                self.open_keyboarUVWXYZ()
            return
    print('length buttons ' + str(buttons))

    deltaArr = []
```

```
        samples = []
        avgDeltaArr = []
        timeSamples = []
        timeStamps = []
        timeDiff = []

        self.flashBlocksSelection()

    def flashLettersSelectionABCDE(self):

        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration, 
timeFlashes
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dTSVec, 
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global tpAmaxMean, tpSMaxMean, AmaxThreshold, SmaxThreshold, 
dTSLThreshold, dTSHThreshold, tpSMaxLThreshold, tpSMaxHThreshold
        global textOfTextBox

        sleep(1)
        self.flashLetters('Start')
        sleep(0.1)
        self.unFlashLetters('Start')
        sleep(1)

        # first resolve an EOG stream on the lab network
        print("\nlooking for an EOG stream...")
        streams = resolve_stream('type', 'EEG')
        # create a new inlet to read from the stream
        inlet = StreamInlet(streams[0])
        sample, timestamp = inlet.pull_sample()
        while len(sample) == 0:
            samples, timestamp = inlet.pull_sample()
        x1 = sample[0]
        initialTimeStamp = timestamp
        timeStamps.append(0)
        samples.append(x1)
        deltaArr.append(0)
        oldTimestamp = timestamp

        self.flashLetters('A')

        start = time.time()

        while time.time() <= (start + (timeFlashes*0.04+1)):
            sample, timestamp = inlet.pull_sample()
            if timestamp > oldTimestamp:
                timeStamps.append(timestamp - initialTimeStamp )
                timeSample = time.time() - start
                timeSamples.append(timeSample)
                x2 = sample[0]
                samples.append(x2)
                deltaX = x2 - x1
                deltaArr.append(deltaX)
                x1 = x2
                oldTimestamp = timestamp

                tempoA =  time.time() - start
```

```
                    if (tempoA >= 0.1 and self.labelA.cget("bg") == "darkgreen"):
                        self.unFlashLetters('A')
                    elif ( tempoA >= 0.01*timeFlashes and tempoA < (0.01*timeFlashes +
0.1)  and self.labelB.cget("bg")!= "darkgreen"):
                        self.flashLetters('B')
                    elif ( tempoA >= (0.01*timeFlashes + 0.1)  and
self.labelB.cget("bg")== "darkgreen" ):
                        self.unFlashLetters('B')
                    elif ( tempoA >= 0.02*timeFlashes and tempoA < (0.02*timeFlashes +
0.1) and self.labelC.cget("bg")!= "darkgreen"):
                        self.flashLetters('C')
                    elif ( tempoA >= (0.02*timeFlashes + 0.1) and
self.labelC.cget("bg")== "darkgreen" ):
                        self.unFlashLetters('C')
                    elif ( tempoA >= 0.03*timeFlashes and tempoA < (0.03*timeFlashes +
0.1) and self.labelD.cget("bg")!= "darkgreen"):
                        self.flashLetters('D')
                    elif ( tempoA >= (0.03*timeFlashes + 0.1) and
self.labelD.cget("bg")== "darkgreen" ):
                        self.unFlashLetters('D')
                    elif ( tempoA >= 0.04*timeFlashes and tempoA < (0.04*timeFlashes +
0.1) and self.labelE.cget("bg")!= "darkgreen"):
                        self.flashLetters('E')
                    elif ( tempoA >= (0.04*timeFlashes + 0.1) and
self.labelE.cget("bg")== "darkgreen" ):
                        self.unFlashLetters('E')


        avgDeltaArr.insert(0,deltaArr[0])
        avgDeltaArr.insert(1,deltaArr[1])
        for i in range(2,len(deltaArr),1):
            a = i-1
            aa = i -2
            value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
            avgDeltaArr.insert(i , value )

        buttons = {}

        for i in range(1,6,1):
            timeStampsIndex = [idx for idx, val in enumerate(timeStamps) if (val >
0.1+(i-1)*0.01*timeFlashes and val < 0.8+(i-1)*0.01*timeFlashes)]
            timeStampsA = [timeStamps[index] for index in timeStampsIndex]
            samplesA = [samples[index] for index in timeStampsIndex]
            deltaArrA = [avgDeltaArr[index] for index in timeStampsIndex]
            delayT = 0.01*timeFlashes*(i-1)
            timeStampsA[:] = [timeStampA - delayT for timeStampA in timeStampsA]

            if(samplesA):
                AmaxA = max(samplesA)
                tpAMaxIndexA = samplesA.index(AmaxA)
                SmaxA = max(deltaArrA)
                tpSMaxIndexA = deltaArrA.index(SmaxA)
                tpSMaxA = timeStampsA[tpSMaxIndexA]
                SminA = min(deltaArrA[tpSMaxIndexA:(tpSMaxIndexA+50)])

                tpMinIndexA = deltaArrA.index(SminA)
                dTSA = timeStampsA[tpMinIndexA] - timeStampsA[tpSMaxIndexA]
            if( (SmaxA > SmaxThreshold) and (dTSA > dTSLThreshold) and (dTSA <
dTSHThreshold) and (tpSMaxA > tpSMaxLThreshold) and (tpSMaxA < tpSMaxHThreshold)):

                if i == 1:
                    buttons["A"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 2:
                    buttons["B"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 3:
                    buttons["C"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 4:
                    buttons["D"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 5:
                    buttons["E"] =  abs(tpSMaxA - tpSMaxMean)

                print('\nBotao '+ str(i) + 'Pré-Selecionado')
                print('Amax = ' + str(AmaxA))
                print('tpAmax = ' + str(timeStampsA[tpAMaxIndexA]))
                print('tpSmax = ' + str(timeStampsA[tpSMaxIndexA]))
                print('Smax = ' + str(SmaxA))
                print('tpSmin = ' + str(timeStampsA[tpMinIndexA]))
                print('Smin = ' + str(SminA))
                print('dTS = ' + str(dTSA))
                print('Samples = ' + str(len(samplesA)))
                print('deltaArr = ' + str(len(deltaArrA)))

            deltaArrA = []
            samplesA = []
            timeStampsA = []

        selected = ''
        if buttons:

            selected = min(buttons, key=buttons.get)
            print('\n Final Selection ' + selected)
            self.textBox.insert(tkinter.END, selected)
            self.main_window.update()

        print('length buttons ' + str(buttons))

        deltaArr = []
        samples = []
        avgDeltaArr = []
        timeSamples = []
        timeStamps = []
        timeDiff = []

        self.labelA.grid_remove()
        self.labelB.grid_remove()
        self.labelC.grid_remove()
        self.labelD.grid_remove()
        self.labelE.grid_remove()
        self.labelABCDE.grid()
        self.labelFGHIJ.grid()
        self.labelKLMNO.grid()
        self.labelPQRST.grid()
        self.labelUVWXYZ.grid()
        self.flashBlocksSelection()
```

```
    def flashLettersSelectionFGHIJ(self):

        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration,
timeFlashes
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dTSVec,
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global tpAmaxMean, tpSMaxMean, AmaxThreshold, SmaxThreshold,
dTSLThreshold, dTSHThreshold, tpSMaxLThreshold, tpSMaxHThreshold
        global textOfTextBox

        sleep(1)
        self.flashLetters('Start')
        sleep(0.1)
        self.unFlashLetters('Start')
        sleep(1)

        # first resolve an EOG stream on the lab network
        print("\nlooking for an EOG stream...")
        streams = resolve_stream('type', 'EEG')
        # create a new inlet to read from the stream
        inlet = StreamInlet(streams[0])
        sample, timestamp = inlet.pull_sample()
        while len(sample) == 0:
            samples, timestamp = inlet.pull_sample()
        x1 = sample[0]
        initialTimeStamp = timestamp
        timeStamps.append(0)
        samples.append(x1)
        deltaArr.append(0)
        oldTimestamp = timestamp

        self.flashLetters('F')

        start = time.time()

        while time.time() <= (start + (timeFlashes*0.04+1)):
            sample, timestamp = inlet.pull_sample()
            if timestamp > oldTimestamp:
                timeStamps.append(timestamp - initialTimeStamp )
                timeSample = time.time() - start
                timeSamples.append(timeSample)
                x2 = sample[0]
                samples.append(x2)
                deltaX = x2 - x1
                deltaArr.append(deltaX)
                x1 = x2
                oldTimestamp = timestamp

                tempoA =  time.time() - start
                if (tempoA >= 0.1 and self.labelF.cget("bg") == "darkgreen"):
                    self.unFlashLetters('F')
                elif ( tempoA >= 0.01*timeFlashes and tempoA < (0.01*timeFlashes +
0.1)  and self.labelG.cget("bg")!= "darkgreen"):
                    self.flashLetters('G')
                elif ( tempoA >= (0.01*timeFlashes + 0.1)  and
self.labelG.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('G')
                elif ( tempoA >= 0.02*timeFlashes and tempoA < (0.02*timeFlashes + 
0.1) and self.labelH.cget("bg")!= "darkgreen"):
                    self.flashLetters('H')
                elif ( tempoA >= (0.02*timeFlashes + 0.1) and 
self.labelH.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('H')
                elif ( tempoA >= 0.03*timeFlashes and tempoA < (0.03*timeFlashes + 
0.1) and self.labelI.cget("bg")!= "darkgreen"):
                    self.flashLetters('I')
                elif ( tempoA >= (0.03*timeFlashes + 0.1) and 
self.labelI.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('I')
                elif ( tempoA >= 0.04*timeFlashes and tempoA < (0.04*timeFlashes + 
0.1) and self.labelJ.cget("bg")!= "darkgreen"):
                    self.flashLetters('J')
                elif ( tempoA >= (0.04*timeFlashes + 0.1) and 
self.labelJ.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('J')


        avgDeltaArr.insert(0,deltaArr[0])
        avgDeltaArr.insert(1,deltaArr[1])
        for i in range(2,len(deltaArr),1):
            a = i-1
            aa = i -2
            value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
            avgDeltaArr.insert(i , value )

        buttons = {}

        for i in range(1,6,1):
            timeStampsIndex = [idx for idx, val in enumerate(timeStamps) if (val > 
0.1+(i-1)*0.01*timeFlashes and val < 0.8+(i-1)*0.01*timeFlashes)]
            timeStampsA = [timeStamps[index] for index in timeStampsIndex]
            samplesA = [samples[index] for index in timeStampsIndex]
            deltaArrA = [avgDeltaArr[index] for index in timeStampsIndex]
            delayT = 0.01*timeFlashes*(i-1)
            timeStampsA[:] = [timeStampA - delayT for timeStampA in timeStampsA]

            if(samplesA):
                AmaxA = max(samplesA)
                tpAMaxIndexA = samplesA.index(AmaxA)
                SmaxA = max(deltaArrA)
                tpSMaxIndexA = deltaArrA.index(SmaxA)
                tpSMaxA = timeStampsA[tpSMaxIndexA]
                SminA = min(deltaArrA[tpSMaxIndexA:(tpSMaxIndexA+50)])

                tpMinIndexA = deltaArrA.index(SminA)
                dTSA = timeStampsA[tpMinIndexA] - timeStampsA[tpSMaxIndexA]

            if( (SmaxA > SmaxThreshold) and (dTSA > dTSLThreshold) and (dTSA < 
dTSHThreshold) and (tpSMaxA > tpSMaxLThreshold) and (tpSMaxA < tpSMaxHThreshold)):

                if i == 1:
                    buttons["F"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 2:
                    buttons["G"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 3:
```

```
                buttons["H"] =  abs(tpSMaxA - tpSMaxMean)
            elif i == 4:
                buttons["I"] =  abs(tpSMaxA - tpSMaxMean)
            elif i == 5:
                buttons["J"] =  abs(tpSMaxA - tpSMaxMean)

            print('\nBotao '+ str(i) + 'Pré-Selecionado')
            print('Amax = ' + str(AmaxA))
            print('tpAmax = ' + str(timeStampsA[tpAMaxIndexA]))
            print('tpSmax = ' + str(timeStampsA[tpSMaxIndexA]))
            print('Smax = ' + str(SmaxA))
            print('tpSmin = ' + str(timeStampsA[tpMinIndexA]))
            print('Smin = ' + str(SminA))
            print('dTS = ' + str(dTSA))
            print('Samples = ' + str(len(samplesA)))
            print('deltaArr = ' + str(len(deltaArrA)))

        deltaArrA = []
        samplesA = []
        timeStampsA = []

    selected = ''
    if buttons:

        selected = min(buttons, key=buttons.get)
        print('\n Final Selection ' + selected)
        if selected == 'Return':
            textOfTextBox = self.textBox.get('1.0', 'end')
            self.rebuild_Menu()
            deltaArr = []
            samples = []
            avgDeltaArr = []
            timeSamples = []
            timeStamps = []
            timeDiff = []
            return
        elif selected == "Backspace":
            self.deleteTextChar()
        else:
            self.textBox.insert(tkinter.END, selected)
            self.main_window.update()

    print('Length buttons ' + str(buttons))

    deltaArr = []
    samples = []
    avgDeltaArr = []
    timeSamples = []
    timeStamps = []
    timeDiff = []

    self.labelF.grid_remove()
    self.labelG.grid_remove()
    self.labelH.grid_remove()
    self.labelI.grid_remove()
    self.labelJ.grid_remove()
    self.labelABCDE.grid()
    self.labelFGHIJ.grid()
```

```
        self.labelKLMNO.grid()
        self.labelPQRST.grid()
        self.labelUVWXYZ.grid()
        self.flashBlocksSelection()

    def flashLettersSelectionKLMNO(self):

        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration,
timeFlashes
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dTSVec,
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global tpAmaxMean, tpSMaxMean, AmaxThreshold, SmaxThreshold,
dTSLThreshold, dTSHThreshold, tpSMaxLThreshold, tpSMaxHThreshold
        global textOfTextBox

        sleep(1)
        self.flashLetters('Start')
        sleep(0.1)
        self.unFlashLetters('Start')
        sleep(1)

        # first resolve an EOG stream on the lab network
        print("\nlooking for an EOG stream...")
        streams = resolve_stream('type', 'EEG')
        # create a new inlet to read from the stream
        inlet = StreamInlet(streams[0])
        sample, timestamp = inlet.pull_sample()
        while len(sample) == 0:
            samples, timestamp = inlet.pull_sample()
        x1 = sample[0]
        initialTimeStamp = timestamp
        timeStamps.append(0)
        samples.append(x1)
        deltaArr.append(0)
        oldTimestamp = timestamp

        self.flashLetters('K')

        start = time.time()

        while time.time() <= (start + (timeFlashes*0.04+1)):
            sample, timestamp = inlet.pull_sample()
            if timestamp > oldTimestamp:
                timeStamps.append(timestamp - initialTimeStamp )
                timeSample = time.time() - start
                timeSamples.append(timeSample)
                x2 = sample[0]
                samples.append(x2)
                deltaX = x2 - x1
                deltaArr.append(deltaX)
                x1 = x2
                oldTimestamp = timestamp

                tempoA =  time.time() - start
                if (tempoA >= 0.1 and self.labelK.cget("bg") == "darkgreen"):
                    self.unFlashLetters('K')
                elif ( tempoA >= 0.01*timeFlashes and tempoA < (0.01*timeFlashes +
0.1)  and self.labelL.cget("bg")!= "darkgreen"):
                    self.flashLetters('L')
                elif ( tempoA >= (0.01*timeFlashes + 0.1)  and
self.labelL.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('L')
                elif ( tempoA >= 0.02*timeFlashes and tempoA < (0.02*timeFlashes +
0.1) and self.labelM.cget("bg")!= "darkgreen"):
                    self.flashLetters('M')
                elif ( tempoA >= (0.02*timeFlashes + 0.1) and
self.labelM.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('M')
                elif ( tempoA >= 0.03*timeFlashes and tempoA < (0.03*timeFlashes +
0.1) and self.labelN.cget("bg")!= "darkgreen"):
                    self.flashLetters('N')
                elif ( tempoA >= (0.03*timeFlashes + 0.1) and
self.labelN.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('N')
                elif ( tempoA >= 0.04*timeFlashes and tempoA < (0.04*timeFlashes +
0.1) and self.labelO.cget("bg")!= "darkgreen"):
                    self.flashLetters('O')
                elif ( tempoA >= (0.04*timeFlashes + 0.1) and
self.labelO.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('O')


        avgDeltaArr.insert(0,deltaArr[0])
        avgDeltaArr.insert(1,deltaArr[1])
        for i in range(2,len(deltaArr),1):
            a = i-1
            aa = i -2
            value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
            avgDeltaArr.insert(i , value )

        buttons = {}

        for i in range(1,6,1):
            timeStampsIndex = [idx for idx, val in enumerate(timeStamps) if (val >
0.1+(i-1)*0.01*timeFlashes and val < 0.8+(i-1)*0.01*timeFlashes)]
            timeStampsA = [timeStamps[index] for index in timeStampsIndex]
            samplesA = [samples[index] for index in timeStampsIndex]
            deltaArrA = [avgDeltaArr[index] for index in timeStampsIndex]
            delayT = 0.01*timeFlashes*(i-1)
            timeStampsA[:] = [timeStampA - delayT for timeStampA in timeStampsA]

            if(samplesA):
                AmaxA = max(samplesA)
                tpAMaxIndexA = samplesA.index(AmaxA)
                SmaxA = max(deltaArrA)
                tpSMaxIndexA = deltaArrA.index(SmaxA)
                tpSMaxA = timeStampsA[tpSMaxIndexA]
                SminA = min(deltaArrA[tpSMaxIndexA:(tpSMaxIndexA+50)])

                tpMinIndexA = deltaArrA.index(SminA)
                dTSA = timeStampsA[tpMinIndexA] - timeStampsA[tpSMaxIndexA]

            if( (SmaxA > SmaxThreshold) and (dTSA > dTSLThreshold) and (dTSA <
dTSHThreshold) and (tpSMaxA > tpSMaxLThreshold) and (tpSMaxA < tpSMaxHThreshold)):
```

```
            if i == 1:
                buttons["K"] =  abs(tpSMaxA - tpSMaxMean)
            elif i == 2:
                buttons["L"] =  abs(tpSMaxA - tpSMaxMean)
            elif i == 3:
                buttons["M"] =  abs(tpSMaxA - tpSMaxMean)
            elif i == 4:
                buttons["N"] =  abs(tpSMaxA - tpSMaxMean)
            elif i == 5:
                buttons["O"] =  abs(tpSMaxA - tpSMaxMean)

            print('\nBotao '+ str(i) + 'Pré-Selecionado')
            print('Amax = ' + str(AmaxA))
            print('tpAmax = ' + str(timeStampsA[tpAMaxIndexA]))
            print('tpSmax = ' + str(timeStampsA[tpSMaxIndexA]))
            print('Smax = ' + str(SmaxA))
            print('tpSmin = ' + str(timeStampsA[tpMinIndexA]))
            print('Smin = ' + str(SminA))
            print('dTS = ' + str(dTSA))
            print('Samples = ' + str(len(samplesA)))
            print('deltaArr = ' + str(len(deltaArrA)))

        deltaArrA = []
        samplesA = []
        timeStampsA = []

    selected = ''
    if buttons:

        selected = min(buttons, key=buttons.get)
        print('\n Final Selection ' + selected)
        self.textBox.insert(tkinter.END, selected)
        self.main_window.update()

    print('Length buttons ' + str(buttons))

    deltaArr = []
    samples = []
    avgDeltaArr = []
    timeSamples = []
    timeStamps = []
    timeDiff = []

    self.labelK.grid_remove()
    self.labelL.grid_remove()
    self.labelM.grid_remove()
    self.labelN.grid_remove()
    self.labelO.grid_remove()
    self.labelABCDE.grid()
    self.labelFGHIJ.grid()
    self.labelKLMNO.grid()
    self.labelPQRST.grid()
    self.labelUVWXYZ.grid()
    self.flashBlocksSelection()

def flashLettersSelectionPQRST(self):
        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration, 
timeFlashes
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dTSVec, 
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global tpAmaxMean, tpSMaxMean, AmaxThreshold, SmaxThreshold, 
dTSLThreshold, dTSHThreshold, tpSMaxLThreshold, tpSMaxHThreshold
        global textOfTextBox

        sleep(1)
        self.flashLetters('Start')
        sleep(0.1)
        self.unFlashLetters('Start')
        sleep(1)

        # first resolve an EOG stream on the lab network
        print("\nlooking for an EOG stream...")
        streams = resolve_stream('type', 'EEG')
        # create a new inlet to read from the stream
        inlet = StreamInlet(streams[0])
        sample, timestamp = inlet.pull_sample()
        while len(sample) == 0:
            samples, timestamp = inlet.pull_sample()
        x1 = sample[0]
        initialTimeStamp = timestamp
        timeStamps.append(0)
        samples.append(x1)
        deltaArr.append(0)
        oldTimestamp = timestamp

        self.flashLetters('P')

        start = time.time()

        while time.time() <= (start + (timeFlashes*0.04+1)):
            sample, timestamp = inlet.pull_sample()
            if timestamp > oldTimestamp:
                timeStamps.append(timestamp - initialTimeStamp )
                timeSample = time.time() - start
                timeSamples.append(timeSample)
                x2 = sample[0]
                samples.append(x2)
                deltaX = x2 - x1
                deltaArr.append(deltaX)
                x1 = x2
                oldTimestamp = timestamp

                tempoA =  time.time() - start
                if (tempoA >= 0.1 and self.labelP.cget("bg") == "darkgreen"):
                    self.unFlashLetters('P')
                elif ( tempoA >= 0.01*timeFlashes and tempoA < (0.01*timeFlashes + 
0.1)  and self.labelQ.cget("bg")!= "darkgreen"):
                    self.flashLetters('Q')
                elif ( tempoA >= (0.01*timeFlashes + 0.1)  and 
self.labelQ.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('Q')
                elif ( tempoA >= 0.02*timeFlashes and tempoA < (0.02*timeFlashes + 
0.1) and self.labelR.cget("bg")!= "darkgreen"):
```

```
                    self.flashLetters('R')
                elif ( tempoA >= (0.02*timeFlashes + 0.1) and
self.labelR.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('R')
                elif ( tempoA >= 0.03*timeFlashes and tempoA < (0.03*timeFlashes +
0.1) and self.labelS.cget("bg")!= "darkgreen"):
                    self.flashLetters('S')
                elif ( tempoA >= (0.03*timeFlashes + 0.1) and
self.labelS.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('S')
                elif ( tempoA >= 0.04*timeFlashes and tempoA < (0.04*timeFlashes +
0.1) and self.labelT.cget("bg")!= "darkgreen"):
                    self.flashLetters('T')
                elif ( tempoA >= (0.04*timeFlashes + 0.1) and
self.labelT.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('T')


        avgDeltaArr.insert(0,deltaArr[0])
        avgDeltaArr.insert(1,deltaArr[1])
        for i in range(2,len(deltaArr),1):
            a = i-1
            aa = i -2
            value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
            avgDeltaArr.insert(i , value )

        buttons = {}

        for i in range(1,6,1):
            timeStampsIndex = [idx for idx, val in enumerate(timeStamps) if (val >
0.1+(i-1)*0.01*timeFlashes and val < 0.8+(i-1)*0.01*timeFlashes)]
            timeStampsA = [timeStamps[index] for index in timeStampsIndex]
            samplesA = [samples[index] for index in timeStampsIndex]
            deltaArrA = [avgDeltaArr[index] for index in timeStampsIndex]
            delayT = 0.01*timeFlashes*(i-1)
            timeStampsA[:] = [timeStampA - delayT for timeStampA in timeStampsA]

            if(samplesA):
                AmaxA = max(samplesA)
                tpAMaxIndexA = samplesA.index(AmaxA)
                SmaxA = max(deltaArrA)
                tpSMaxIndexA = deltaArrA.index(SmaxA)
                tpSMaxA = timeStampsA[tpSMaxIndexA]
                SminA = min(deltaArrA[tpSMaxIndexA:(tpSMaxIndexA+50)])

                tpMinIndexA = deltaArrA.index(SminA)
                dTSA = timeStampsA[tpMinIndexA] - timeStampsA[tpSMaxIndexA]


            if( (SmaxA > SmaxThreshold) and (dTSA > dTSLThreshold) and (dTSA <
dTSHThreshold) and (tpSMaxA > tpSMaxLThreshold) and (tpSMaxA < tpSMaxHThreshold)):

                if i == 1:
                    buttons["P"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 2:
                    buttons["Q"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 3:
                    buttons["R"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 4:
                    buttons["S"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 5:
                    buttons["T"] =  abs(tpSMaxA - tpSMaxMean)

                print('\nBotao '+ str(i) + 'Pré-Selecionado')
                print('Amax = ' + str(AmaxA))
                print('tpAmax = ' + str(timeStampsA[tpAMaxIndexA]))
                print('tpSmax = ' + str(timeStampsA[tpSMaxIndexA]))
                print('Smax = ' + str(SmaxA))
                print('tpSmin = ' + str(timeStampsA[tpMinIndexA]))
                print('Smin = ' + str(SminA))
                print('dTS = ' + str(dTSA))
                print('Samples = ' + str(len(samplesA)))
                print('deltaArr = ' + str(len(deltaArrA)))

            deltaArrA = []
            samplesA = []
            timeStampsA = []

        selected = ''
        if buttons:

            selected = min(buttons, key=buttons.get)
            print('\n Final Selection ' + selected)
            self.textBox.insert(tkinter.END, selected)
            self.main_window.update()

        print('Length buttons ' + str(buttons))

        deltaArr = []
        samples = []
        avgDeltaArr = []
        timeSamples = []
        timeStamps = []
        timeDiff = []


        self.labelP.grid_remove()
        self.labelQ.grid_remove()
        self.labelR.grid_remove()
        self.labelS.grid_remove()
        self.labelT.grid_remove()
        self.labelABCDE.grid()
        self.labelFGHIJ.grid()
        self.labelKLMNO.grid()
        self.labelPQRST.grid()
        self.labelUVWXYZ.grid()
        self.flashBlocksSelection()

    def flashLettersSelectionUVWXYZ(self):

        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration,
timeFlashes
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dTSVec,
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global tpAmaxMean, tpSMaxMean, AmaxThreshold, SmaxThreshold,
dTSLThreshold, dTSHThreshold, tpSMaxLThreshold, tpSMaxHThreshold
        global textOfTextBox

        sleep(1)
        self.flashLetters('Start')
        sleep(0.1)
        self.unFlashLetters('Start')
        sleep(1)

        # first resolve an EOG stream on the lab network
        print("\nlooking for an EOG stream...")
        streams = resolve_stream('type', 'EEG')
        # create a new inlet to read from the stream
        inlet = StreamInlet(streams[0])
        sample, timestamp = inlet.pull_sample()
        while len(sample) == 0:
            samples, timestamp = inlet.pull_sample()
        x1 = sample[0]
        initialTimeStamp = timestamp
        timeStamps.append(0)
        samples.append(x1)
        deltaArr.append(0)
        oldTimestamp = timestamp

        self.flashLetters('U')

        start = time.time()

        while time.time() <= (start + (timeFlashes*0.05+1)):
            sample, timestamp = inlet.pull_sample()
            if timestamp > oldTimestamp:
                timeStamps.append(timestamp - initialTimeStamp )
                timeSample = time.time() - start
                timeSamples.append(timeSample)
                x2 = sample[0]
                samples.append(x2)
                deltaX = x2 - x1
                deltaArr.append(deltaX)
                x1 = x2
                oldTimestamp = timestamp

                tempoA =  time.time() - start
                if (tempoA >= 0.1 and self.labelU.cget("bg") == "darkgreen"):
                    self.unFlashLetters('U')
                elif ( tempoA >= 0.01*timeFlashes and tempoA < (0.01*timeFlashes +
0.1)  and self.labelV.cget("bg")!= "darkgreen"):
                    self.flashLetters('V')
                elif ( tempoA >= (0.01*timeFlashes + 0.1)  and
self.labelV.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('V')
                elif ( tempoA >= 0.02*timeFlashes and tempoA < (0.02*timeFlashes +
0.1) and self.labelW.cget("bg")!= "darkgreen"):
                    self.flashLetters('W')
                elif ( tempoA >= (0.02*timeFlashes + 0.1) and
self.labelW.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('W')
                elif ( tempoA >= 0.03*timeFlashes and tempoA < (0.03*timeFlashes + 
0.1) and self.labelX.cget("bg")!= "darkgreen"):
                    self.flashLetters('X')
                elif ( tempoA >= (0.03*timeFlashes + 0.1) and 
self.labelX.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('X')
                elif ( tempoA >= 0.04*timeFlashes and tempoA < (0.04*timeFlashes + 
0.1) and self.labelY.cget("bg")!= "darkgreen"):
                    self.flashLetters('Y')
                elif ( tempoA >= (0.04*timeFlashes + 0.1) and 
self.labelY.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('Y')
                elif ( tempoA >= 0.05*timeFlashes and tempoA < (0.05*timeFlashes + 
0.1) and self.labelZ.cget("bg")!= "darkgreen"):
                    self.flashLetters('Z')
                elif ( tempoA >= (0.05*timeFlashes + 0.1) and 
self.labelZ.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('Z')


        avgDeltaArr.insert(0,deltaArr[0])
        avgDeltaArr.insert(1,deltaArr[1])
        for i in range(2,len(deltaArr),1):
            a = i-1
            aa = i -2
            value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
            avgDeltaArr.insert(i , value )

        buttons = {}

        for i in range(1,7,1):
            timeStampsIndex = [idx for idx, val in enumerate(timeStamps) if (val > 
0.1+(i-1)*0.01*timeFlashes and val < 0.8+(i-1)*0.01*timeFlashes)]
            timeStampsA = [timeStamps[index] for index in timeStampsIndex]
            samplesA = [samples[index] for index in timeStampsIndex]
            deltaArrA = [avgDeltaArr[index] for index in timeStampsIndex]
            delayT = 0.01*timeFlashes*(i-1)
            timeStampsA[:] = [timeStampA - delayT for timeStampA in timeStampsA]

            if(samplesA):
                AmaxA = max(samplesA)
                tpAMaxIndexA = samplesA.index(AmaxA)
                SmaxA = max(deltaArrA)
                tpSMaxIndexA = deltaArrA.index(SmaxA)
                tpSMaxA = timeStampsA[tpSMaxIndexA]
                SminA = min(deltaArrA[tpSMaxIndexA:(tpSMaxIndexA+50)])

                tpMinIndexA = deltaArrA.index(SminA)
                dTSA = timeStampsA[tpMinIndexA] - timeStampsA[tpSMaxIndexA]


            if( (SmaxA > SmaxThreshold) and (dTSA > dTSLThreshold) and (dTSA < 
dTSHThreshold) and (tpSMaxA > tpSMaxLThreshold) and (tpSMaxA < tpSMaxHThreshold)):

                if i == 1:
                    buttons["U"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 2:
                    buttons["V"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 3:
                    buttons["W"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 4:
                    buttons["X"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 5:
                    buttons["Y"] =  abs(tpSMaxA - tpSMaxMean)
                elif i == 5:
                    buttons["Z"] =  abs(tpSMaxA - tpSMaxMean)

                print('\nBotao '+ str(i) + 'Pré-Selecionado')
                print('Amax = ' + str(AmaxA))
                print('tpAmax = ' + str(timeStampsA[tpAMaxIndexA]))
                print('tpSmax = ' + str(timeStampsA[tpSMaxIndexA]))
                print('Smax = ' + str(SmaxA))
                print('tpSmin = ' + str(timeStampsA[tpMinIndexA]))
                print('Smin = ' + str(SminA))
                print('dTS = ' + str(dTSA))
                print('Samples = ' + str(len(samplesA)))
                print('deltaArr = ' + str(len(deltaArrA)))

            deltaArrA = []
            samplesA = []
            timeStampsA = []

        selected = ''
        if buttons:

            selected = min(buttons, key=buttons.get)
            print('\n Final Selection ' + selected)
            self.textBox.insert(tkinter.END, selected)
            self.main_window.update()

        print('length buttons ' + str(buttons))

        deltaArr = []
        samples = []
        avgDeltaArr = []
        timeSamples = []
        timeStamps = []
        timeDiff = []


        self.labelU.grid_remove()
        self.labelV.grid_remove()
        self.labelW.grid_remove()
        self.labelX.grid_remove()
        self.labelY.grid_remove()
        self.labelZ.grid_remove()
        self.labelABCDE.grid()
        self.labelFGHIJ.grid()
        self.labelKLMNO.grid()
        self.labelPQRST.grid()
        self.labelUVWXYZ.grid()
        self.flashBlocksSelection()

    def deleteTextChar(self):
        self.textBox.delete("1." + str(len(self.textBox.get('1.0', tkinter.END))-
2),tkinter.END)
```

```
    def open_keyboard(self):

        global textOfTextBox

        self.button.grid_remove()
        self.button1.grid_remove()
        self.button2.grid_remove()
        self.button3.grid_remove()
        self.button4.grid_remove()
        self.button5.grid_remove()

        self.textBox = tkinter.Text(self.main_window, height=1)
        self.textBox.grid(row = 0, column = 0, columnspan= 5)
        myFont = tkFont.Font(family = "Times New Roman", size = 25, weight
="bold")
        self.textBox.configure(font = myFont)


        if textOfTextBox:
            self.textBox.insert(tkinter.END,textOfTextBox.strip())

        self.start = tkinter.Button( self.main_window, text = 'Começar',
                                     font = ("Arial Bold", 33))
        self.start.grid(column=0, row=2, sticky ="WENS", pady = 30, padx = 30,
columnspan =1 )

        self.labelABCDE = tkinter.Label(self.main_window,
                                      text = 'A B C D E',
                                      font = ("Arial Bold", 33))
        self.labelABCDE.grid(column=2, row=2, sticky ="WENS", pady = 30, padx =
30)

        self.labelFGHIJ = tkinter.Label(self.main_window,
                                      text = 'F G H I J',
                                      font = ("Arial Bold", 33))
        self.labelFGHIJ.grid(column=4, row=3, sticky ="WENS", pady = 30, padx =
30)

        self.labelKLMNO = tkinter.Label(self.main_window,
                                      text = 'K L M N O',
                                      font = ("Arial Bold", 33))
        self.labelKLMNO.grid(column=3, row=4, sticky ="WENS", pady = 30, padx =
30)

        self.labelPQRST = tkinter.Label(self.main_window,
                                      text = 'P Q R S T',
                                      font = ("Arial Bold", 33))
        self.labelPQRST.grid(column=1, row=4, sticky ="WENS", pady = 30, padx =
30)

        self.labelUVWXYZ = tkinter.Label(self.main_window,
                                      text = 'U V W X Y Z',
                                      font = ("Arial Bold", 33))
        self.labelUVWXYZ.grid(column=0, row=3, sticky ="WENS", pady = 30, padx =
30)

        self.backspace = tkinter.Button( self.main_window, text = 'Apagar',
                                       font = ("Arial Bold", 35),
command=self.deleteTextChar)
        self.backspace.grid(column=0, row=5, sticky ="WENS", pady = 30, padx = 30,
columnspan =1 )

        self.space = tkinter.Button( self.main_window, text = 'Espaço',
                                       font = ("Arial Bold", 35),
command=self.textBox.insert(tkinter.END, ' '))
        self.space.grid(column=1, row=5, sticky ="WENS", pady = 33, padx = 30,
columnspan =1 )

        self.returnMenu = tkinter.Button( self.main_window, text = 'retornar para
o Menu',
                                       font = ("Arial Bold", 33),
command=self.rebuild_Menu)
        self.returnMenu.grid(column=2, row=5, sticky ="WENS", pady = 30, padx =
30, columnspan =3 )

        self.main_window.update()

        self.flashBlocksSelection()

    def open_keyboardABCDE(self):
        global textOfTextBox

        self.labelABCDE.grid_remove()
        self.labelFGHIJ.grid_remove()
        self.labelKLMNO.grid_remove()
        self.labelPQRST.grid_remove()
        self.labelUVWXYZ.grid_remove()

        self.labelA = tkinter.Label(self.main_window,
                                        text = '       A       ',
                                        font = ("Arial Bold", 35))
        self.labelA.grid(column=2, row=2, sticky ="WENS", pady = 30, padx = 30)

        self.labelB = tkinter.Label(self.main_window,
                                        text = '       B       ',
                                        font = ("Arial Bold", 35))
        self.labelB.grid(column=4, row=3, sticky ="WENS", pady = 30, padx = 30)

        self.labelC = tkinter.Label(self.main_window,
                                        text = '       C       ',
                                        font = ("Arial Bold", 35))
        self.labelC.grid(column=3, row=4, sticky ="WENS", pady = 30, padx = 30)

        self.labelD = tkinter.Label(self.main_window,
                                        text = '       D       ',
                                        font = ("Arial Bold", 35))
        self.labelD.grid(column=1, row=4, sticky ="WENS", pady = 30, padx = 30)

        self.labelE = tkinter.Label(self.main_window,
                                        text = '       E       ',
                                        font = ("Arial Bold", 35))
        self.labelE.grid(column=0, row=3, sticky ="WENS", pady = 30, padx = 30)

        self.main_window.update()
        self.flashLettersSelectionABCDE()

    def open_keyboardFGHIJ(self):
        global textOfTextBox

        self.labelABCDE.grid_remove()
        self.labelFGHIJ.grid_remove()
        self.labelKLMNO.grid_remove()
        self.labelPQRST.grid_remove()
        self.labelUVWXYZ.grid_remove()

        self.labelF = tkinter.Label(self.main_window,
                                    text = '        F        ',
                                    font = ("Arial Bold", 35))
        self.labelF.grid(column=2, row=2, sticky ="WENS", pady = 30, padx = 30)

        self.labelG = tkinter.Label(self.main_window,
                                    text = '        G        ',
                                    font = ("Arial Bold", 35))
        self.labelG.grid(column=4, row=3, sticky ="WENS", pady = 30, padx = 30)

        self.labelH = tkinter.Label(self.main_window,
                                    text = '        H        ',
                                    font = ("Arial Bold", 35))
        self.labelH.grid(column=3, row=4, sticky ="WENS", pady = 30, padx = 30)

        self.labelI = tkinter.Label(self.main_window,
                                    text = '        I        ',
                                    font = ("Arial Bold", 35))
        self.labelI.grid(column=1, row=4, sticky ="WENS", pady = 30, padx = 30)

        self.labelJ = tkinter.Label(self.main_window,
                                    text = '        J        ',
                                    font = ("Arial Bold", 35))
        self.labelJ.grid(column=0, row=3, sticky ="WENS", pady = 30, padx = 30)

        self.main_window.update()

        self.flashLettersSelectionFGHIJ()

    def open_keyboarKLMNO(self):
            global textOfTextBox

            self.labelABCDE.grid_remove()
            self.labelFGHIJ.grid_remove()
            self.labelKLMNO.grid_remove()
            self.labelPQRST.grid_remove()
            self.labelUVWXYZ.grid_remove()

            self.labelK = tkinter.Label(self.main_window,
                                        text = '        K        ',
                                        font = ("Arial Bold", 35))
            self.labelK.grid(column=2, row=2, sticky ="WENS", pady = 30, padx =
30)

            self.labelL = tkinter.Label(self.main_window,
                                        text = '        L        ',
                                        font = ("Arial Bold", 35))
            self.labelL.grid(column=4, row=3, sticky ="WENS", pady = 30, padx =
30)

            self.labelM = tkinter.Label(self.main_window,
                                        text = '        M        ',
                                        font = ("Arial Bold", 35))
            self.labelM.grid(column=3, row=4, sticky ="WENS", pady = 30, padx =
30)

            self.labelN = tkinter.Label(self.main_window,
                                        text = '        N        ',
                                        font = ("Arial Bold", 35))
            self.labelN.grid(column=1, row=4, sticky ="WENS", pady = 30, padx =
30)

            self.labelO = tkinter.Label(self.main_window,
                                        text = '        O        ',
                                        font = ("Arial Bold", 35))
            self.labelO.grid(column=0, row=3, sticky ="WENS", pady = 30, padx =
30)

            self.main_window.update()

            self.flashLettersSelectionKLMNO()

    def open_keyboarPQRST(self):
        global textOfTextBox

        self.labelABCDE.grid_remove()
        self.labelFGHIJ.grid_remove()
        self.labelKLMNO.grid_remove()
        self.labelPQRST.grid_remove()
        self.labelUVWXYZ.grid_remove()

        self.labelP = tkinter.Label(self.main_window,
                                    text = '        P        ',
                                    font = ("Arial Bold", 35))
        self.labelP.grid(column=2, row=2, sticky ="WENS", pady = 30, padx = 30)

        self.labelQ = tkinter.Label(self.main_window,
                                    text = '        Q        ',
                                    font = ("Arial Bold", 35))
        self.labelQ.grid(column=4, row=3, sticky ="WENS", pady = 30, padx = 30)

        self.labelR = tkinter.Label(self.main_window,
                                    text = '        R        ',
                                    font = ("Arial Bold", 35))
        self.labelR.grid(column=3, row=4, sticky ="WENS", pady = 30, padx = 30)

        self.labelS = tkinter.Label(self.main_window,
                                    text = '        S        ',
                                    font = ("Arial Bold", 35))
        self.labelS.grid(column=1, row=4, sticky ="WENS", pady = 30, padx = 30)

        self.labelT = tkinter.Label(self.main_window,
                                    text = '        T        ',
                                    font = ("Arial Bold", 35))
        self.labelT.grid(column=0, row=3, sticky ="WENS", pady = 30, padx = 30)
        self.main_window.update()

        self.flashLettersSelectionPQRST()

    def open_keyboarUVWXYZ(self):
        global textOfTextBox

        self.labelABCDE.grid_remove()
        self.labelFGHIJ.grid_remove()
        self.labelKLMNO.grid_remove()
        self.labelPQRST.grid_remove()
        self.labelUVWXYZ.grid_remove()

        self.labelU = tkinter.Label(self.main_window,
                                    text = '       U        ',
                                    font = ("Arial Bold", 35))
        self.labelU.grid(column=2, row=2, sticky ="WENS", pady = 30, padx = 30)

        self.labelV = tkinter.Label(self.main_window,
                                    text = '       V        ',
                                    font = ("Arial Bold", 35))
        self.labelV.grid(column=4, row=3, sticky ="WENS", pady = 30, padx = 30)

        self.labelW = tkinter.Label(self.main_window,
                                    text = '       W        ',
                                    font = ("Arial Bold", 35))
        self.labelW.grid(column=3, row=4, sticky ="WENS", pady = 30, padx = 30)

        self.labelX = tkinter.Label(self.main_window,
                                    text = '       X        ',
                                    font = ("Arial Bold", 35))
        self.labelX.grid(column=1, row=4, sticky ="WENS", pady = 30, padx = 30)

        self.labelY = tkinter.Label(self.main_window,
                                    text = '       Y        ',
                                    font = ("Arial Bold", 35))
        self.labelY.grid(column=0, row=3, sticky ="WENS", pady = 30, padx = 30)

        self.labelZ = tkinter.Label(self.main_window,
                                    text = '       Z        ',
                                    font = ("Arial Bold", 35))
        self.labelZ.grid(column=0, row=3, sticky ="WENS", pady = 30, padx = 30)

        self.main_window.update()

        self.flashLettersSelectionUVWXYZ()

    def flashLetters(self, letter):
        if letter == 'A':
            self.labelA.config(bg = "darkgreen")
            self.labelA.update()
        elif letter == 'B':
            self.labelB.config(bg = "darkgreen")
            self.labelB.update()
        elif letter == 'C':
            self.labelC.config(bg = "darkgreen")
            self.labelC.update()
```

```
elif letter == 'D':
    self.labelD.config(bg = "darkgreen")
    self.labelD.update()
elif letter == 'E':
    self.labelE.config(bg = "darkgreen")
    self.labelE.update()
elif letter == 'F':
    self.labelF.config(bg = "darkgreen")
    self.labelF.update()
elif letter == 'G':
    self.labelG.config(bg = "darkgreen")
    self.labelG.update()
elif letter == 'H':
    self.labelH.config(bg = "darkgreen")
    self.labelH.update()
elif letter == 'I':
    self.labelI.config(bg = "darkgreen")
    self.labelI.update()
elif letter == 'J':
    self.labelJ.config(bg = "darkgreen")
    self.labelJ.update()
elif letter == 'K':
    self.labelK.config(bg = "darkgreen")
    self.labelK.update()
elif letter == 'L':
    self.labelL.config(bg = "darkgreen")
    self.labelL.update()
elif letter == 'M':
    self.labelM.config(bg = "darkgreen")
    self.labelM.update()
elif letter == 'N':
    self.labelN.config(bg = "darkgreen")
    self.labelN.update()
elif letter == 'O':
    self.labelO.config(bg = "darkgreen")
    self.labelO.update()
elif letter == 'P':
    self.labelP.config(bg = "darkgreen")
    self.labelP.update()
elif letter == 'Q':
    self.labelQ.config(bg = "darkgreen")
    self.labelQ.update()
elif letter == 'R':
    self.labelR.config(bg = "darkgreen")
    self.labelR.update()
elif letter == 'S':
    self.labelS.config(bg = "darkgreen")
    self.labelS.update()
elif letter == 'T':
    self.labelT.config(bg = "darkgreen")
    self.labelT.update()
elif letter == 'U':
    self.labelU.config(bg = "darkgreen")
    self.labelU.update()
elif letter == 'V':
    self.labelV.config(bg = "darkgreen")
    self.labelV.update()
elif letter == 'W':
```

```
            self.labelW.config(bg = "darkgreen")
            self.labelW.update()
        elif letter == 'X':
            self.labelX.config(bg = "darkgreen")
            self.labelX.update()
        elif letter == 'Y':
            self.labelY.config(bg = "darkgreen")
            self.labelY.update()
        elif letter == 'Z':
            self.labelZ.config(bg = "darkgreen")
            self.labelZ.update()
        elif letter == 'Backspace':
            self.backspace.config(bg = "darkgreen")
            self.backspace.update()
        elif letter == 'Start':
            self.start.config(bg = "darkgreen")
            self.start.update()
        elif letter == 'Space':
            self.space.config(bg = "darkgreen")
            self.space.update()
        elif letter == 'Return':
            self.returnMenu.config(bg = "darkgreen")
            self.returnMenu.update()

    def flashBlocks(self,block):
        if block == 1:
                self.labelABCDE.config(bg = "darkgreen")
                self.main_window.update()
        elif block == 2:
                self.labelFGHIJ.config(bg = "darkgreen")
                self.main_window.update()
        elif block == 3:
                self.labelKLMNO.config(bg = "darkgreen")
                self.main_window.update()
        elif block == 4:
                self.labelPQRST.config(bg = "darkgreen")
                self.main_window.update()
        elif block == 5:
                self.labelUVWXYZ.config(bg = "darkgreen")
                self.main_window.update()

    def unflashBlocks(self,block):
        if block == 1:
                self.labelABCDE.config(bg = "#F0F0F0")
                self.main_window.update()
        elif block == 2:
                self.labelFGHIJ.config(bg = "#F0F0F0")
                self.main_window.update()
        elif block == 3:
                self.labelKLMNO.config(bg = "#F0F0F0")
                self.main_window.update()
        elif block == 4:
                self.labelPQRST.config(bg = "#F0F0F0")
                self.main_window.update()
        elif block == 5:
                self.labelUVWXYZ.config(bg = "#F0F0F0")
                self.main_window.update()
```

```
def unFlashLetters(self,letter):
    if letter == 'A':
        self.labelA.config(bg = "#F0F0F0")
        self.labelA.update()
    elif letter == 'B':
        self.labelB.config(bg = "#F0F0F0")
        self.labelB.update()
    elif letter == 'C':
        self.labelC.config(bg = "#F0F0F0")
        self.labelC.update()
    elif letter == 'D':
        self.labelD.config(bg = "#F0F0F0")
        self.labelD.update()
    elif letter == 'E':
        self.labelE.config(bg = "#F0F0F0")
        self.labelE.update()
    elif letter == 'F':
        self.labelF.config(bg = "#F0F0F0")
        self.labelF.update()
    elif letter == 'G':
        self.labelG.config(bg = "#F0F0F0")
        self.labelG.update()
    elif letter == 'H':
        self.labelH.config(bg = "#F0F0F0")
        self.labelH.update()
    elif letter == 'I':
        self.labelI.config(bg = "#F0F0F0")
        self.labelI.update()
    elif letter == 'J':
        self.labelJ.config(bg = "#F0F0F0")
        self.labelJ.update()
    elif letter == 'K':
        self.labelK.config(bg = "#F0F0F0")
        self.labelK.update()
    elif letter == 'L':
        self.labelL.config(bg = "#F0F0F0")
        self.labelL.update()
    elif letter == 'M':
        self.labelM.config(bg = "#F0F0F0")
        self.labelM.update()
    elif letter == 'N':
        self.labelN.config(bg = "#F0F0F0")
        self.labelN.update()
    elif letter == 'O':
        self.labelO.config(bg = "#F0F0F0")
        self.labelO.update()
    elif letter == 'P':
        self.labelP.config(bg = "#F0F0F0")
        self.labelP.update()
    elif letter == 'Q':
        self.labelQ.config(bg = "#F0F0F0")
        self.labelQ.update()
    elif letter == 'R':
        self.labelR.config(bg = "#F0F0F0")
        self.labelR.update()
    elif letter == 'S':
        self.labelS.config(bg = "#F0F0F0")
        self.labelS.update()
        elif letter == 'T':
            self.labelT.config(bg = "#F0F0F0")
            self.labelT.update()
        elif letter == 'U':
            self.labelU.config(bg = "#F0F0F0")
            self.labelU.update()
        elif letter == 'V':
            self.labelV.config(bg = "#F0F0F0")
            self.labelV.update()
        elif letter == 'W':
            self.labelW.config(bg = "#F0F0F0")
            self.labelW.update()
        elif letter == 'X':
            self.labelX.config(bg = "#F0F0F0")
            self.labelX.update()
        elif letter == 'Y':
            self.labelY.config(bg = "#F0F0F0")
            self.labelY.update()
        elif letter == 'Z':
            self.labelZ.config(bg = "#F0F0F0")
            self.labelZ.update()
        elif letter == 'Backspace':
            self.backspace.config(bg = "#F0F0F0")
            self.backspace.update()
        elif letter == 'Start':
            self.start.config(bg = "#F0F0F0")
            self.start.update()
        elif letter == 'Space':
            self.space.config(bg = "#F0F0F0")
            self.space.update()
        elif letter == 'Return':
            self.returnMenu.config(bg = "#F0F0F0")
            self.returnMenu.update()

    def rebuild_Menu(self):
        for items in self.main_window.grid_slaves():
            items.grid_remove()
        self.button.grid()
        self.button1.grid()
        self.button2.grid()
        self.button3.grid()
        self.button4.grid()
        self.button5.grid()

window = EOGInterface()
```

APPENDIX C – PYTHON CODE OF THE TEST INTERFACE

```
import tkinter
from pylsl import StreamInlet, resolve_stream
import time
import matplotlib.pyplot as plt
from time import sleep
import statistics
import xlsxwriter

deltaArr = []
samples = []
timeSamples = []
timeStamps = []

timeDiff = []
duration = 0.8

AmaxVec = []
tpAMaxVec = []
tpSMaxVec = []
SmaxVec = []
SminVec = []
dpnVec = []
avgDeltaArr = []

AmaxVec = []
tpAMaxVec = []
tpSMaxVec = []
SmaxVec = []
SminVec = []
dpnVec = []

rodada = 1
piscadas = 0
piscadaErrada = 0

tpAMaxMean = 0.2989
tpSMaxMean = 0.2848
AmaxThreshold = 50
SmaxThreshold = 10
dpnLThreshold = 0.01719546178285966
dpnHThreshold = 0.09648825820838099
tpSMaxLThreshold  = 0.17852545621477722
tpSMaxHThreshold = 0.391202563781769
dpnCalibration = []
tpSMaxCalibration = []

class FlashingInterface:
    def __init__(self):

        self.main_window = tkinter.Tk()

        self.main_window.geometry('770x250')

        self.button = tkinter.Button( self.main_window, text = 'Segmento',
                                        font = ("Arial Bold",30),
command=self.extractSegment)
        self.button.grid(column=0, row=0)

        self.button2 = tkinter.Button( self.main_window, text = 'Calibrar',
                                        font = ("Arial Bold", 30),
command=self.calibrar)
        self.button2.grid(column=1, row=0)

        self.button2 = tkinter.Button( self.main_window, text = 'Acurácia',
                                        font = ("Arial Bold", 30),
command=self.acuracia)
        self.button2.grid(column=2, row=0)

        self.button2 = tkinter.Button( self.main_window, text = 'Iniciar',
                                        font = ("Arial Bold", 30),
command=self.iniciar)
        self.button2.grid(column=3, row=0)

        self.label1 = tkinter.Label(self.main_window,
                                         text = 'Pisque',
                                         font = ("Arial Bold", 30))
        self.label1.grid(column=1, row=1)

        self.labelA = tkinter.Label(self.main_window,
                                         text = 'A',
                                         font = ("Arial Bold", 30))
        self.labelA.grid(column=0, row=2)

        self.labelB = tkinter.Label(self.main_window,
                                         text = 'B',
                                         font = ("Arial Bold", 30))
        self.labelB.grid(column=1, row=2)

        self.labelC = tkinter.Label(self.main_window,
                                         text = 'C',
                                         font = ("Arial Bold", 30))
        self.labelC.grid(column=2, row=2)

        tkinter.mainloop()

    def extractSegment(self):
        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dpnVec,
avgDeltaArr
        global rodada, piscadas, piscadaErrada

        # first resolve an EOG stream on the lab network
        print("\nlooking for an EOG stream...")
        streams = resolve_stream('type', 'EEG')
        # create a new inlet to read from the stream
        inlet = StreamInlet(streams[0])
        sample, timestamp = inlet.pull_sample()
        while len(sample) == 0:
            sample, timestamp = inlet.pull_sample()
        x1 = sample[0]
        initialTimeStamp = timestamp
        timeStamps.append(0)
```

```
samples.append(x1)
deltaArr.append(0)
oldTimestamp = timestamp

self.flash()

start = time.time()

while time.time() <= start + duration:
    sample, timestamp = inlet.pull_sample()
    if timestamp > oldTimestamp:
        timeStamps.append(timestamp - initialTimeStamp )
        timeSample = time.time() - start
        timeSamples.append(timeSample)
        x2 = sample[0]
        samples.append(x2)
        deltaX = x2 - x1
        deltaArr.append(deltaX)
        x1 = x2
        timeDiff.append(timestamp - oldTimestamp)
        oldTimestamp = timestamp
        tempo = time.time()
        lim = start + 0.1
        if (tempo > lim ):
            self.unFlash()

avgDeltaArr.insert(0,deltaArr[0])
avgDeltaArr.insert(1,deltaArr[1])
for i in range(2,len(deltaArr),1):
    a = i-1
    aa = i -2
    value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
    avgDeltaArr.insert(i , value)

# plotting the points
plt.rcParams.update({'font.size': 18})
plt.plot(timeStamps, samples)
plt.plot(timeStamps, deltaArr)
plt.plot(timeStamps, avgDeltaArr)
# naming the x axis
plt.xlabel('x - time (s)')
plt.xlabel
# naming the y axis
plt.ylabel('y - voltage (\u03BCV)')
# giving a title to my graph
plt.title('EOG Blink Segment')
# function to show the plot
plt.grid()


samples = samples[20:]
deltaArr = avgDeltaArr[20:]
timeStamps = timeStamps[20:]

Amax = max(samples)
tpAMaxIndex = samples.index(Amax)
Smax = max(deltaArr)
tpSMaxIndex = deltaArr.index(Smax)
```

```
        Smin = min(deltaArr[tpSMaxIndex:(tpSMaxIndex+25)])
        tpMinIndex = deltaArr.index(Smin)
        dpn = timeStamps[tpMinIndex] - timeStamps[tpSMaxIndex]

        print('Amax = ' + str(Amax))
        print('tpAmax = ' + str(timeStamps[tpAMaxIndex]))
        print('tpSmax = ' + str(timeStamps[tpSMaxIndex]))
        print('Smax = ' + str(Smax))
        print('tpSmin = ' + str(timeStamps[tpMinIndex]))
        print('Smin = ' + str(Smin))
        print('dpn = ' + str(dpn))
        print('Samples Len= ' + str(len(samples)))
        print('deltaArr Len= ' + str(len(deltaArr)))

        plt.show()

        timeSamples = []
        deltaArr = []
        samples = []
        timeStamps = []
        timeDiff = []
        avgDeltaArr = []

    def calibrar(self):

        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dpnVec,
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global AmaxMean, tpAMaxMean, tpSMaxMean
        global AmaxThreshold, SmaxThreshold, dpnLThreshold, dpnHThreshold,
tpSMaxLThreshold, tpSMaxHThreshold
        global dpnCalibration, tpSMaxCalibration, SMaxMeanCalibration


        while len(SmaxVec) < 10:


            self.label2 = tkinter.Label(self.main_window,
                                        text = '3',
                                        font = ("Arial Bold", 30))
            self.label2.grid(column=0, row=1)
            self.label2.update()
            sleep(0.75)
            self.label2.config(text = "2")
            self.label2.update()
            sleep(0.75)
            self.label2.config(text = "1")
            self.label2.update()
            sleep(0.75)
            self.label2.config(text = "0")
            self.label2.update()
            self.label2.destroy()

            # first resolve an EOG stream on the lab network
            print("\nlooking for an EOG stream...")
            streams = resolve_stream('type', 'EEG')
            # create a new inlet to read from the stream
```

```
inlet = StreamInlet(streams[0])
sample, timestamp = inlet.pull_sample()
while len(sample) == 0:
    sample, timestamp = inlet.pull_sample()
x1 = sample[0]
initialTimeStamp = timestamp
timeStamps.append(0)
samples.append(x1)
deltaArr.append(0)
oldTimestamp = timestamp

self.flash()
start = time.time()

while time.time() <= start + duration:
    sample, timestamp = inlet.pull_sample()
    if timestamp > oldTimestamp:
        timeStamps.append(timestamp - initialTimeStamp )
        timeSample = time.time() - start
        timeSamples.append(timeSample)
        x2 = sample[0]
        samples.append(x2)
        deltaX = x2 - x1
        deltaArr.append(deltaX)
        x1 = x2
        timeDiff.append(timestamp - oldTimestamp)
        oldTimestamp = timestamp
        tempo = time.time()
        lim = start + 0.1
        if (tempo > lim ):
            self.unFlash()

avgDeltaArr.insert(0,deltaArr[0])
avgDeltaArr.insert(1,deltaArr[1])
for i in range(2,len(deltaArr),1):
    a = i-1
    aa = i -2
    value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
    avgDeltaArr.insert(i , value)

samples = samples[20:]
deltaArr = avgDeltaArr[20:]
timeStamps = timeStamps[20:]

Amax = max(samples)
tpAMaxIndex = samples.index(Amax)
Smax = max(deltaArr)
tpSMaxIndex = deltaArr.index(Smax)
Smin = min(deltaArr[tpAMaxIndex:(tpAMaxIndex+25)])
tpMinIndex = deltaArr.index(Smin)
dpn = timeStamps[tpMinIndex] - timeStamps[tpSMaxIndex]

print('Amax = ' + str(Amax))
print('tpAmax = ' + str(timeStamps[tpAMaxIndex]))
print('tpSmax = ' + str(timeStamps[tpSMaxIndex]))
print('Smax = ' + str(Smax))
print('tpSmin = ' + str(timeStamps[tpMinIndex]))
print('Smin = ' + str(Smin))
```

```
                print('dpn = ' + str(dpn))
                print('Samples Len= ' + str(len(samples)))
                print('deltaArr Len= ' + str(len(deltaArr)))

                AmaxVec.append(Amax)
                tpAMaxVec.append(timeStamps[tpAMaxIndex])
                tpSMaxVec.append(timeStamps[tpSMaxIndex])
                SmaxVec.append(Smax)
                SminVec.append(Smin)
                dpnVec.append(dpn)

                timeSamples = []
                deltaArr = []
                samples = []
                timeStamps = []
                timeDiff = []
                avgDeltaArr = []

        self.buttonContinuar = tkinter.Button( self.main_window, text = 
'Continuar',
                                              font = ("Arial Bold", 30), 
command=self.calibrar2)
        self.buttonContinuar.grid(column = 2, row=3)

    def calibrar2(self):

        self.buttonContinuar.destroy()
        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dpnVec, 
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global AmaxMean, tpAMaxMean, tpSMaxMean
        global AmaxThreshold, SmaxThreshold, dpnLThreshold, dpnHThreshold, 
tpSMaxLThreshold, tpSMaxHThreshold
        global dpnCalibration, tpSMaxCalibration, SMaxMeanCalibration


        while len(SmaxVec) < 20:


            self.label2 = tkinter.Label(self.main_window,
                                        text = '3',
                                        font = ("Arial Bold", 30))
            self.label2.grid(column=0, row=1)
            self.label2.update()
            sleep(0.75)
            self.label2.config(text = "2")
            self.label2.update()
            sleep(0.75)
            self.label2.config(text = "1")
            self.label2.update()
            sleep(0.75)
            self.label2.config(text = "0")
            self.label2.update()
            self.label2.destroy()

            # first resolve an EOG stream on the lab network
            print("\nLooking for an EOG stream...")
```

```
streams = resolve_stream('type', 'EEG')
# create a new inlet to read from the stream
inlet = StreamInlet(streams[0])
sample, timestamp = inlet.pull_sample()
while len(sample) == 0:
    sample, timestamp = inlet.pull_sample()
x1 = sample[0]
initialTimeStamp = timestamp
timeStamps.append(0)
samples.append(x1)
deltaArr.append(0)
oldTimestamp = timestamp

self.flash()
start = time.time()

while time.time() <= start + duration:
    sample, timestamp = inlet.pull_sample()
    if timestamp > oldTimestamp:
        timeStamps.append(timestamp - initialTimeStamp )
        timeSample = time.time() - start
        timeSamples.append(timeSample)
        x2 = sample[0]
        samples.append(x2)
        deltaX = x2 - x1
        deltaArr.append(deltaX)
        x1 = x2
        timeDiff.append(timestamp - oldTimestamp)
        oldTimestamp = timestamp
        tempo = time.time()
        lim = start + 0.1
        if (tempo > lim ):
            self.unFlash()

avgDeltaArr.insert(0,deltaArr[0])
avgDeltaArr.insert(1,deltaArr[1])
for i in range(2,len(deltaArr),1):
    a = i-1
    aa = i -2
    value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
    avgDeltaArr.insert(i , value)

samples = samples[20:]
deltaArr = avgDeltaArr[20:]
timeStamps = timeStamps[20:]

Amax = max(samples)
tpAMaxIndex = samples.index(Amax)
Smax = max(deltaArr)
tpSMaxIndex = deltaArr.index(Smax)
Smin = min(deltaArr[tpAMaxIndex:(tpAMaxIndex+25)])
tpMinIndex = deltaArr.index(Smin)
dpn = timeStamps[tpMinIndex] - timeStamps[tpSMaxIndex]

print('Amax = ' + str(Amax))
print('tpAmax = ' + str(timeStamps[tpAMaxIndex]))
print('tpSmax = ' + str(timeStamps[tpSMaxIndex]))
print('Smax = ' + str(Smax))
```

```
            print('tpSmin = ' + str(timeStamps[tpMinIndex]))
            print('Smin = ' + str(Smin))
            print('dpn = ' + str(dpn))
            print('Samples Len= ' + str(len(samples)))
            print('deltaArr Len= ' + str(len(deltaArr)))

            AmaxVec.append(Amax)
            tpAMaxVec.append(timeStamps[tpAMaxIndex])
            tpSMaxVec.append(timeStamps[tpSMaxIndex])
            SmaxVec.append(Smax)
            SminVec.append(Smin)
            dpnVec.append(dpn)

            timeSamples = []
            deltaArr = []
            samples = []
            timeStamps = []
            timeDiff = []
            avgDeltaArr = []

        AmaxMean = sum(AmaxVec) / len(AmaxVec)
        tpAMaxMean = sum(tpAMaxVec) / len(tpAMaxVec)
        tpSMaxMean = sum(tpSMaxVec) / len(tpSMaxVec)
        SMaxMean  = sum(SmaxVec) / len(SmaxVec)
        SMinMean = sum(SminVec) / len(SminVec)
        dpnMean = sum(dpnVec) / len(dpnVec)

        print('\nAmaxMean = ' + str(AmaxMean))
        print('TpAMaxMean = ' + str(tpAMaxMean))
        print('TpSMaxMean = ' + str(tpSMaxMean))
        print('SmaxMean = ' + str(SMaxMean))
        print('SminMean = ' + str(SMinMean))
        print('dpnMean = ' + str(dpnMean))

        for i in dpnVec:
            if (i > (dpnMean - 2*statistics.stdev(dpnVec))) or (i < (dpnMean + 2*
statistics.stdev(dpnVec))):
                if (i > 0):
                    dpnCalibration.append(i)

        for i in tpSMaxVec:
            if (i > (tpSMaxMean - 2*statistics.stdev(tpSMaxVec))) or (i <
(tpSMaxMean + 2* statistics.stdev(tpSMaxVec))):
                tpSMaxCalibration.append(i)

        for i in SmaxVec:
            if (i > (SMaxMean - 2*statistics.stdev(SmaxVec))) or (i < (SMaxMean +
2* statistics.stdev(SmaxVec))):
                SMaxMeanCalibration.append(i)

        print('dpnMean after remove outliers= ' +
str(sum(dpnCalibration)/len(dpnCalibration)))
        print('TpSMaxMean after remove outliers= ' +
str(sum(tpSMaxCalibration)/len(tpSMaxCalibration)))
        print('SMaxMean after remove outliers= ' +
str(sum(SMaxMeanCalibration)/len(SMaxMeanCalibration)))
        tpSDeviation = statistics.stdev(tpSMaxCalibration)
        dpnDeviation = statistics.stdev(dpnCalibration)
        SMaxMeanDeviation = statistics.stdev(SMaxMeanCalibration)
        print('tpSDeviation = ' + str(tpSDeviation))
        print('dpnDeviation = ' + str(dpnDeviation))
        print('SMaxMeanDeviation = ' + str(SMaxMeanDeviation))

        dpnMean =  sum(dpnCalibration) / len(dpnVec)
        dpnLThreshold = dpnMean - 2*statistics.stdev(dpnCalibration)
        dpnHThreshold = dpnMean + 2*statistics.stdev(dpnCalibration)
        tpSMaxLThreshold = tpSMaxMean - 2*statistics.stdev(tpSMaxCalibration)
        tpSMaxHThreshold = tpSMaxMean + 2*statistics.stdev(tpSMaxCalibration)
        SmaxThreshold = SMaxMean - 2*statistics.stdev(SMaxMeanCalibration)

        print('\nNew Thresholds')
        print('Smax Threshold = ' + str(SmaxThreshold))
        print('dpnLTh = ' + str(dpnLThreshold))
        print('dpnHTh = ' + str(dpnHThreshold))
        print('tpSMaxLThreshold = ' + str(tpSMaxLThreshold))
        print('tpSMaxHThreshold = ' + str(tpSMaxHThreshold))

        AmaxVec = []
        tpAMaxVec = []
        tpSMaxVec = []
        SmaxVec = []
        SminVec = []
        dpnVec = []

    def acuracia(self):

        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dpnVec,
avgDeltaArr
        global rodada, piscadas, piscadaErrada

        while rodada < 21:

            sleep(2)
            # first resolve an EOG stream on the lab network
            print("\nlooking for an EOG stream...")
            streams = resolve_stream('type', 'EEG')
            # create a new inlet to read from the stream
            inlet = StreamInlet(streams[0])
            sample, timestamp = inlet.pull_sample()
            while len(sample) == 0:
                samples, timestamp = inlet.pull_sample()
            x1 = sample[0]
            initialTimeStamp = timestamp
            timeStamps.append(0)
            samples.append(x1)
            deltaArr.append(0)
            oldTimestamp = timestamp

            self.flash()
            start = time.time()

            while time.time() <= start + duration:
                sample, timestamp = inlet.pull_sample()
                if timestamp > oldTimestamp:
                    timeStamps.append(timestamp - initialTimeStamp )
                    timeSample = time.time() - start
                    timeSamples.append(timeSample)
                    x2 = sample[0]
                    samples.append(x2)
                    deltaX = x2 - x1
                    deltaArr.append(deltaX)
                    x1 = x2
                    timeDiff.append(timestamp - oldTimestamp)
                    oldTimestamp = timestamp
                    tempo = time.time()
                    lim = start + 0.1
                    if (tempo > lim and (self.label1.cget("bg")== "darkgreen")):
                        self.unFlash()

            avgDeltaArr.insert(0,deltaArr[0])
            avgDeltaArr.insert(1,deltaArr[1])
            for i in range(2,len(deltaArr),1):
                a = i-1
                aa = i -2
                value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
                avgDeltaArr.insert(i , value )

            samples = samples[20:]
            deltaArr = avgDeltaArr[20:]
            timeStamps = timeStamps[20:]

            Amax = max(samples)
            tpAMaxIndex = samples.index(Amax)
            Smax = max(deltaArr)
            tpSMaxIndex = deltaArr.index(Smax)
            tpSMax = timeStamps[tpSMaxIndex]
            Smin = min(deltaArr[tpAMaxIndex:(tpAMaxIndex+25)])
            tpMinIndex = deltaArr.index(Smin)
            dpn = timeStamps[tpMinIndex] - timeStamps[tpSMaxIndex]

            print('Amax = ' + str(Amax))
            print('tpAmax = ' + str(timeStamps[tpAMaxIndex]))
            print('tpSmax = ' + str(timeStamps[tpSMaxIndex]))
            print('Smax = ' + str(Smax))
            print('tpSmin = ' + str(timeStamps[tpMinIndex]))
            print('Smin = ' + str(Smin))
            print('dpn = ' + str(dpn))
            print('Samples = ' + str(len(samples)))
            print('deltaArr = ' + str(len(deltaArr)))


            #Count number of blinks
            if( (Amax > 50) and (Smax >10) and (dpn > 0.021) and (dpn <0.118) and
(tpSMax > 0.33) and (tpSMax <0.63)):
                print('\n Piscou')
                print('rodada = ' + str(rodada))
                piscadas += 1

                AmaxVec.append(Amax)
                tpAMaxVec.append(timeStamps[tpAMaxIndex])
                tpSMaxVec.append(timeStamps[tpSMaxIndex])
                SmaxVec.append(Smax)
                SminVec.append(Smin)
                dpnVec.append(dpn)
                if (rodada > 10):
                    piscadaErrada += 1

            timeSamples = []
            deltaArr = []
            samples = []
            timeStamps = []
            timeDiff = []
            avgDeltaArr = []
            rodada = rodada + 1

    def iniciar(self):

        global deltaArr, samples, timeSamples, timeStamps, timeDiff, duration
        global AmaxVec, tpAMaxVec, tpSMaxVec, SmaxVec, SminVec, dpnVec,
avgDeltaArr
        global rodada, piscadas, piscadaErrada
        global tpAmaxMean, tpSMaxMean, AmaxThreshold, SmaxThreshold,
dpnLThreshold, dpnHThreshold, tpSMaxLThreshold, tpSMaxHThreshold

        sleep(1)
        # first resolve an EOG stream on the lab network
        print("\nlooking for an EOG stream...")
        streams = resolve_stream('type', 'EEG')
        # create a new inlet to read from the stream
        inlet = StreamInlet(streams[0])
        sample, timestamp = inlet.pull_sample()
        while len(sample) == 0:
            samples, timestamp = inlet.pull_sample()
        x1 = sample[0]
        initialTimeStamp = timestamp
        timeStamps.append(0)
        samples.append(x1)
        deltaArr.append(0)
        oldTimestamp = timestamp

        self.flashLetters('A')
        start = time.time()

        while time.time() <= (start + 1.1):
            sample, timestamp = inlet.pull_sample()
            if timestamp > oldTimestamp:
                timeStamps.append(timestamp - initialTimeStamp )
                timeSample = time.time() - start
                timeSamples.append(timeSample)
                x2 = sample[0]
                samples.append(x2)
                deltaX = x2 - x1
                deltaArr.append(deltaX)
                x1 = x2
                oldTimestamp = timestamp

                tempoA =  time.time() - start
                if ( tempoA >= 0.1 and self.labelA.cget("bg") == "darkgreen"):
                    self.unFlashLetters('A')
                elif ( tempoA >= 0.15 and tempoA < 0.25 and
self.labelB.cget("bg")!= "darkgreen"):
```

```
                    self.flashLetters('B')
                elif ( tempoA >= 0.25 and self.labelB.cget("bg")== "darkgreen" ):
                    self.unFlashLetters('B')
                elif ( tempoA >= 0.3 and tempoA < 0.4 and self.labelC.cget("bg")!=
"darkgreen"):
                    self.flashLetters('C')
                elif ( tempoA >= 0.4 and self.labelC.cget("bg") == "darkgreen"):
                    self.unFlashLetters('C')

        avgDeltaArr.insert(0,deltaArr[0])
        avgDeltaArr.insert(1,deltaArr[1])
        for i in range(2,len(deltaArr),1):
            a = i-1
            aa = i -2
            value = (deltaArr[i] + deltaArr[a] + deltaArr[aa])/ 3
            avgDeltaArr.insert(i , value )

        samplesA = samples[20:160]
        deltaArrA = avgDeltaArr[20:160]
        timeStampsA = timeStamps[20:160]

        AmaxA = max(samplesA)
        tpAMaxIndexA = samplesA.index(AmaxA)
        SmaxA = max(deltaArrA)
        tpSMaxIndexA = deltaArrA.index(SmaxA)
        tpSMaxA = timeStampsA[tpSMaxIndexA]
        if (tpAMaxIndexA < 115):
            SminA = min(deltaArrA[tpAMaxIndexA:(tpAMaxIndexA+25)])
        else:
            SminA = min(deltaArrA[tpAMaxIndexA:])

        tpMinIndexA = deltaArrA.index(SminA)
        dpnA = timeStampsA[tpMinIndexA] - timeStampsA[tpSMaxIndexA]

        print('\nBotao A')
        print('Amax = ' + str(AmaxA))
        print('tpAmax = ' + str(timeStampsA[tpAMaxIndexA]))
        print('tpSmax = ' + str(timeStampsA[tpSMaxIndexA]))
        print('Smax = ' + str(SmaxA))
        print('tpSmin = ' + str(timeStampsA[tpMinIndexA]))
        print('Smin = ' + str(SminA))
        print('dpn = ' + str(dpnA))
        print('Samples = ' + str(len(samplesA)))
        print('deltaArr = ' + str(len(deltaArrA)))

        # plotting the points
        plt.plot(timeStampsA, samplesA)
        plt.plot(timeStampsA, deltaArrA)
        # naming the x axis
        plt.xlabel('x - time(s)')
        # naming the y axis
        plt.ylabel('y - voltage(uV)')
        # giving a title to my graph
        plt.title('EOG Blink Segment')
        # function to show the plot
        plt.grid()
        plt.show()
```

```
deltaArrA = []
samplesA = []
timeStampsA = []

samplesB = samples[30:190]
deltaArrB = avgDeltaArr[30:190]
timeStampsB = timeStamps[30:190]
timeStampsB[:] = [timeStampB - 0.15 for timeStampB in timeStampsB]

AmaxB = max(samplesB)
tpAMaxIndexB = samplesB.index(AmaxB)
SmaxB = max(deltaArrB)
tpSMaxIndexB = deltaArrB.index(SmaxB)
tpSMaxB = timeStampsB[tpSMaxIndexB]
if (tpAMaxIndexB < 165):
    SminB = min(deltaArrB[tpAMaxIndexB:(tpAMaxIndexB+25)])
else:
    SminB = min(deltaArrB[tpAMaxIndexB:])

tpMinIndexB = deltaArrB.index(SminB)
dpnB = timeStampsB[tpMinIndexB] - timeStampsB[tpSMaxIndexB]

print('\nBotao B')
print('Amax = ' + str(AmaxB))
print('tpAmax = ' + str(timeStampsB[tpAMaxIndexB]))
print('tpSmax = ' + str(timeStampsB[tpSMaxIndexB]))
print('Smax = ' + str(SmaxB))
print('tpSmin = ' + str(timeStampsB[tpMinIndexB]))
print('Smin = ' + str(SminB))
print('dpn = ' + str(dpnB))
print('Samples = ' + str(len(samplesB)))
print('deltaArr = ' + str(len(deltaArrB)))

# plotting the points
plt.plot(timeStampsB, samplesB)
plt.plot(timeStampsB, deltaArrB)
# naming the x axis
plt.xlabel('x - time(s)')
# naming the y axis
plt.ylabel('y - voltage(uV)')
# giving a title to my graph
plt.title('EOG Blink Segment')
# function to show the plot
plt.grid()
plt.show()

deltaArrB = []
samplesB = []
timeStampsB = []

samplesC = samples[60:220]
deltaArrC = avgDeltaArr[60:220]
timeStampsC = timeStamps[60:220]
timeStampsC[:] = [timeStampC - 0.3 for timeStampC in timeStampsC]

AmaxC = max(samplesC)
tpAMaxIndexC = samplesC.index(AmaxC)
SmaxC = max(deltaArrC)
```

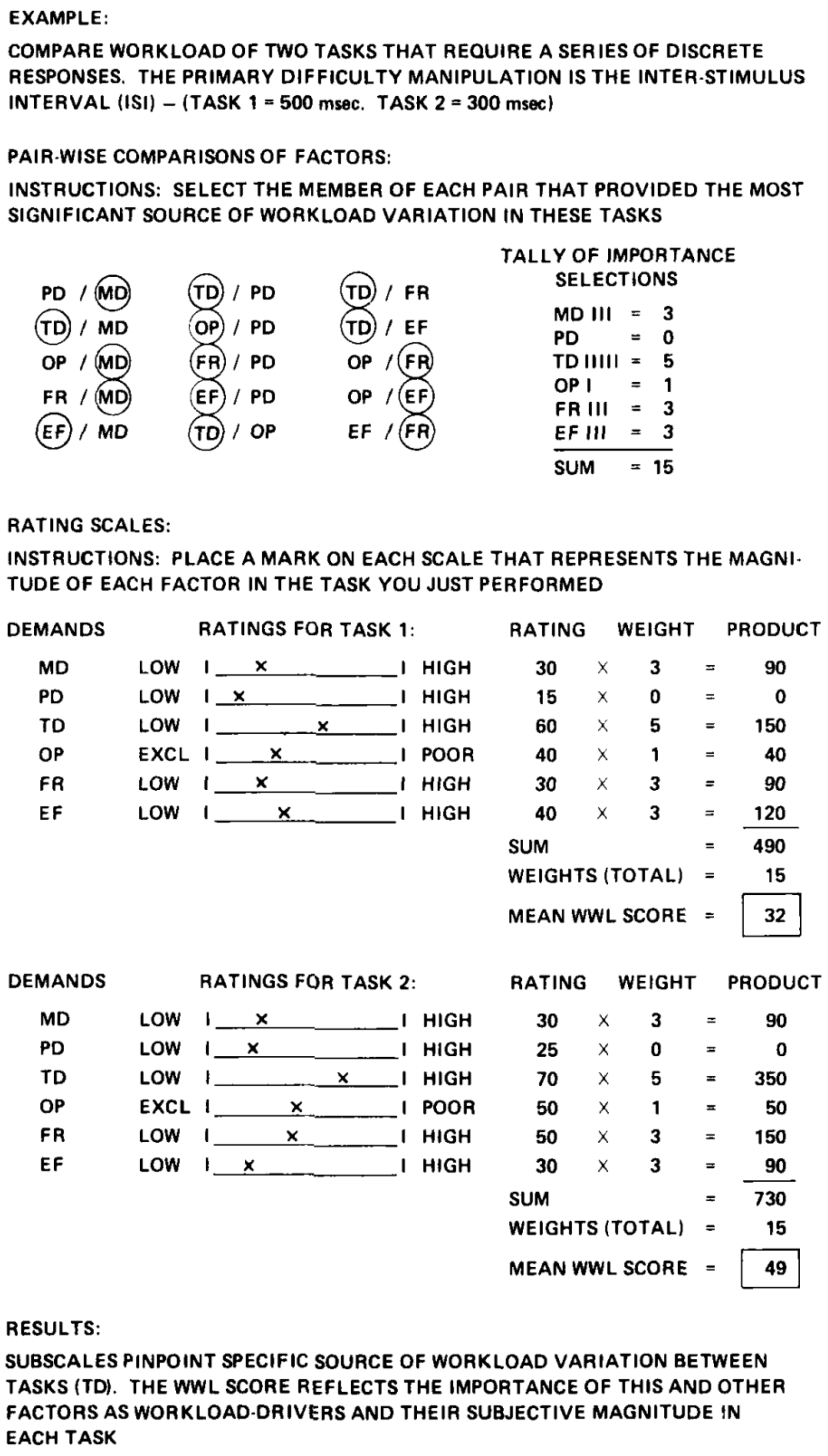
```
tpSMaxIndexC = deltaArrC.index(SmaxC)
tpSMaxC = timeStampsC[tpSMaxIndexC]
if (tpAMaxIndexC < 195):
    SminC = min(deltaArrC[tpAMaxIndexC:(tpAMaxIndexC+25)])
else:
    SminC = min(deltaArrC[tpAMaxIndexC:])

tpMinIndexC = deltaArrC.index(SminC)
dpnC = timeStampsC[tpMinIndexC] - timeStampsC[tpSMaxIndexC]

print('\nBotao C')
print('Amax = ' + str(AmaxC))
print('tpAmax = ' + str(timeStampsC[tpAMaxIndexC]))
print('tpSmax = ' + str(timeStampsC[tpSMaxIndexC]))
print('Smax = ' + str(SmaxC))
print('tpSmin = ' + str(timeStampsC[tpMinIndexC]))
print('Smin = ' + str(SminC))
print('dpn = ' + str(dpnC))
print('Samples = ' + str(len(samplesC)))
print('deltaArr = ' + str(len(deltaArrC)))

# plotting the points
plt.plot(timeStampsC, samplesC)
plt.plot(timeStampsC, deltaArrC)
# naming the x axis
plt.xlabel('x - time(s)')
# naming the y axis
plt.ylabel('y - voltage(uV)')
# giving a title to my graph
plt.title('EOG Blink Segment')
# function to show the plot
plt.grid()
plt.show()

deltaArrC = []
samplesC = []
timeStampsC = []

# plotting the points
plt.plot(timeStamps, samples)
plt.plot(timeStamps, deltaArr)
plt.plot(timeStamps, avgDeltaArr)
# naming the x axis
plt.xlabel('x - time(s)')
# naming the y axis
plt.ylabel('y - voltage(uV)')
# giving a title to my graph
plt.title('EOG Blink Segment')
# function to show the plot
plt.grid()
plt.show()


deltaArr = []
samples = []
avgDeltaArr = []
timeSamples = []
timeStamps = []
```

```
        timeDiff = []

        buttons = {}

        if( (AmaxA > AmaxThreshold) and (SmaxA > SmaxThreshold) and (dpnA >
dpnLThreshold) and (dpnA < dpnHThreshold) and (tpSMaxA > tpSMaxLThreshold) and
(tpSMaxA < tpSMaxHThreshold)):
            print('\n A Selecionado')
            buttons["A"] =  abs(tpSMaxA - tpSMaxMean)

        if( (AmaxB > AmaxThreshold) and (SmaxB > SmaxThreshold) and (dpnB >
dpnLThreshold) and (dpnB < dpnHThreshold) and (tpSMaxB > tpSMaxLThreshold) and
(tpSMaxB < tpSMaxHThreshold)):
            print('\n B Selecionado')
            buttons["B"] = abs(tpSMaxB - tpSMaxMean)

        if( (AmaxC > AmaxThreshold) and (SmaxC > SmaxThreshold) and (dpnC >
dpnLThreshold) and (dpnC < dpnHThreshold) and (tpSMaxC > tpSMaxLThreshold) and
(tpSMaxC < tpSMaxHThreshold)):
            print('\n C Selecionado')
            buttons["C"] =  abs(tpSMaxC - tpSMaxMean)

        if buttons:
            print('\n Final Selection ' + str( min(buttons, key=buttons.get)))

    def flash(self):
        self.label1.config(bg = "darkgreen")
        self.label1.update()

    def flashLetters(self, letter):
        if letter == 'A':
            self.labelA.config(bg = "darkgreen")
            self.labelA.update()
        elif letter == 'B':
            self.labelB.config(bg = "darkgreen")
            self.labelB.update()
        elif letter == 'C':
            self.labelC.config(bg = "darkgreen")
            self.labelC.update()

    def unFlashLetters(self,letter):

        if letter == 'A':
            self.labelA.config(bg = "#F0F0F0")
            self.labelA.update()
        elif letter == 'B':
            self.labelB.config(bg = "#F0F0F0")
            self.labelB.update()
        elif letter == 'C':
            self.labelC.config(bg = "#F0F0F0")
            self.labelC.update()

    def unFlash(self):
        self.label1.config(bg = '#F0F0F0')
        self.label1.update()

window = FlashingInterface()
```

```
AmaxMean = sum(AmaxVec) / len(AmaxVec)
tpAMaxMean = sum(tpAMaxVec) / len(tpAMaxVec)
tpSMaxMean = sum(tpSMaxVec) / len(tpSMaxVec)
SMaxMean  = sum(SmaxVec) / len(SmaxVec)
SMinMean = sum(SminVec) / len(SminVec)
dpnMean = sum(dpnVec) / len(dpnVec)

print('\nAmaxMean = ' + str(AmaxMean))
print('TpAMaxMean = ' + str(tpAMaxMean))
print('TpSMaxMean = ' + str(tpSMaxMean))
print('SmaxMean = ' + str(SMaxMean))
print('SminMean = ' + str(SMinMean))
print('dpnMean = ' + str(dpnMean))

for i in dpnVec:
    if (i > (dpnMean - 2*statistics.stdev(dpnVec))) or (i < (dpnMean + 2*
statistics.stdev(dpnVec))):
        dpnCalibration.append(i)

for i in tpSMaxVec:
    if (i > (tpSMaxMean - 2*statistics.stdev(tpSMaxVec))) or (i < (tpSMaxMean + 2*
statistics.stdev(tpSMaxVec))):
        tpSMaxCalibration.append(i)

dpnMean =  sum(dpnCalibration) / len(dpnVec)
dpnLThreshold = dpnMean - 2*statistics.stdev(dpnCalibration)
dpnHThreshold = dpnMean + 2*statistics.stdev(dpnCalibration)
tpSMaxLThreshold = tpSMaxMean - 2*statistics.stdev(tpSMaxCalibration)
tpSMaxHThreshold = tpSMaxMean + 2*statistics.stdev(tpSMaxCalibration)

print('\nNew Thresholds')
print('dpnLTh = ' + str(dpnLThreshold))
print('dpnHTh = ' + str(dpnHThreshold))
print('tpSMaxLThreshold = ' + str(tpSMaxLThreshold))
print('tpSMaxHThreshold = ' + str(tpSMaxHThreshold))
```

ATTACHMENT A – SAMPLE APPLICATION OF THE NASA-TLX

Source: (HART; STA, 1988)



EXAMPLE:

COMPARE WORKLOAD OF TWO TASKS THAT REQUIRE A SERIES OF DISCRETE RESPONSES. THE PRIMARY DIFFICULTY MANIPULATION IS THE INTER-STIMULUS INTERVAL (ISI) – (TASK 1 = 500 msec. TASK 2 = 300 msec)

PAIR-WISE COMPARISONS OF FACTORS:

INSTRUCTIONS: SELECT THE MEMBER OF EACH PAIR THAT PROVIDED THE MOST SIGNIFICANT SOURCE OF WORKLOAD VARIATION IN THESE TASKS

PD / [✓ MD]	[✓ TD] / PD	[✓ TD] / FR
[✓ TD] / MD	[✓ OP] / PD	[✓ TD] / EF
OP / [✓ MD]	[✓ FR] / PD	OP / [✓ FR]
FR / [✓ MD]	[✓ EF] / PD	OP / [✓ EF]
[✓ EF] / MD	[✓ TD] / OP	EF / [✓ FR]

TALLY OF IMPORTANCE SELECTIONS

MD III	=	3
PD	=	0
TD IIIII	=	5
OP I	=	1
FR III	=	3
EF III	=	3
SUM	=	15

RATING SCALES:

INSTRUCTIONS: PLACE A MARK ON EACH SCALE THAT REPRESENTS THE MAGNITUDE OF EACH FACTOR IN THE TASK YOU JUST PERFORMED

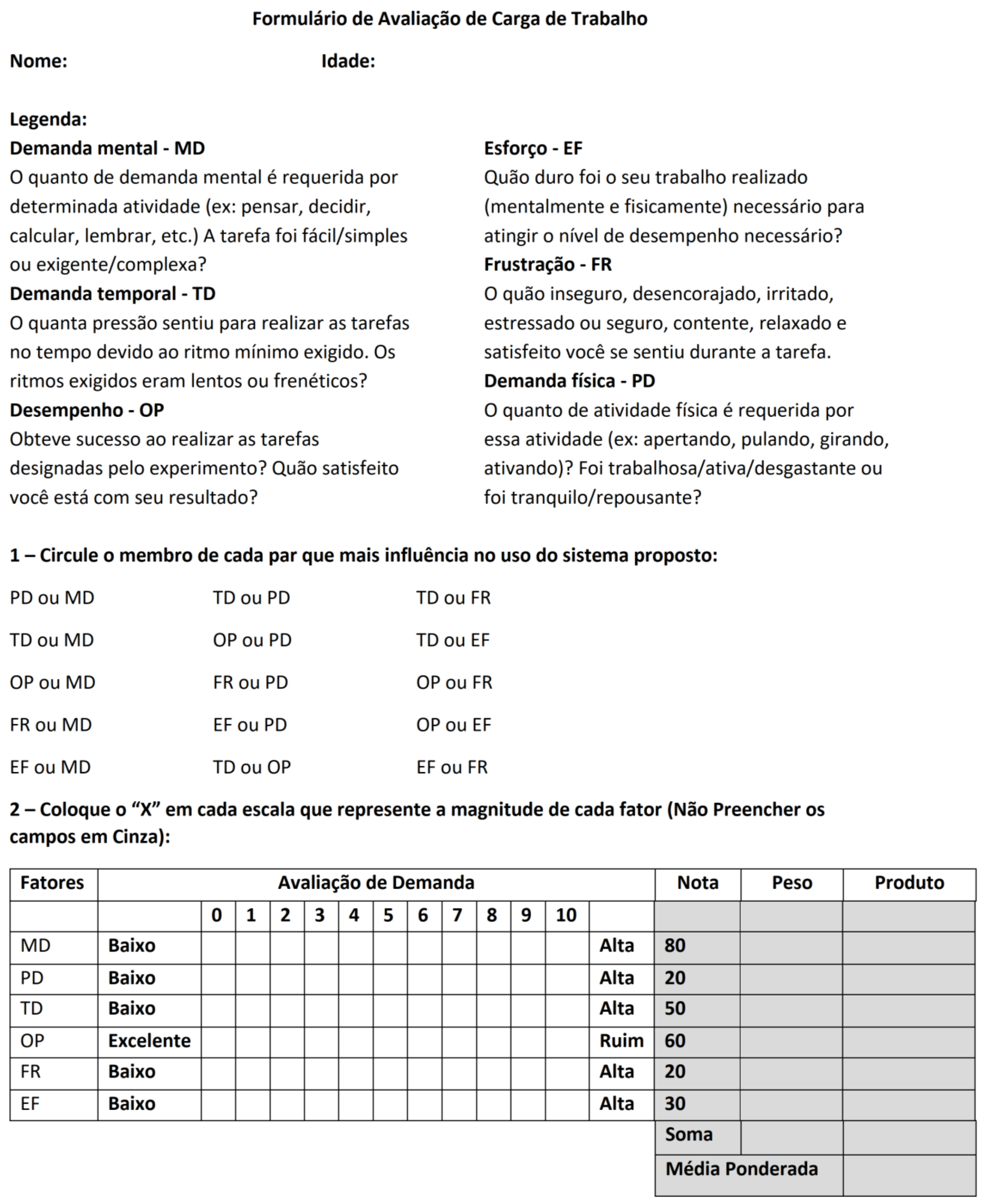
DEMANDS	RATINGS FOR TASK 1:			RATING		WEIGHT		PRODUCT
MD	LOW	I ___x_________________ I	HIGH	30	X	3	=	90
PD	LOW	I _x___________________ I	HIGH	15	X	0	=	0
TD	LOW	I ____________x________ I	HIGH	60	X	5	=	150
OP	EXCL	I _______x_____________ I	POOR	40	X	1	=	40
FR	LOW	I ____x________________ I	HIGH	30	X	3	=	90
EF	LOW	I _______x_____________ I	HIGH	40	X	3	=	120
				SUM			=	490
				WEIGHTS (TOTAL)			=	15
				MEAN WWL SCORE			=	$\boxed{32}$

DEMANDS	RATINGS FOR TASK 2:			RATING		WEIGHT		PRODUCT
MD	LOW	I ____x________________ I	HIGH	30	X	3	=	90
PD	LOW	I ___x_________________ I	HIGH	25	X	0	=	0
TD	LOW	I _______________x_____ I	HIGH	70	X	5	=	350
OP	EXCL	I ________x____________ I	POOR	50	X	1	=	50
FR	LOW	I ________x____________ I	HIGH	50	X	3	=	150
EF	LOW	I ___x_________________ I	HIGH	30	X	3	=	90
				SUM			=	730
				WEIGHTS (TOTAL)			=	15
				MEAN WWL SCORE			=	$\boxed{49}$

RESULTS:

SUBSCALES PINPOINT SPECIFIC SOURCE OF WORKLOAD VARIATION BETWEEN TASKS (TD). THE WWL SCORE REFLECTS THE IMPORTANCE OF THIS AND OTHER FACTORS AS WORKLOAD-DRIVERS AND THEIR SUBJECTIVE MAGNITUDE IN EACH TASK

ATTACHMENT B – ADAPTED FREE TRANSLATION VERSION OF THE NASA-TLX PROTOCOL



Formulário de Avaliação de Carga de Trabalho

Nome: **Idade:**

Legenda:

Demanda mental - MD
O quanto de demanda mental é requerida por determinada atividade (ex: pensar, decidir, calcular, lembrar, etc.) A tarefa foi fácil/simples ou exigente/complexa?

Demanda temporal - TD
O quanta pressão sentiu para realizar as tarefas no tempo devido ao ritmo mínimo exigido. Os ritmos exigidos eram lentos ou frenéticos?

Desempenho - OP
Obteve sucesso ao realizar as tarefas designadas pelo experimento? Quão satisfeito você está com seu resultado?

Esforço - EF
Quão duro foi o seu trabalho realizado (mentalmente e fisicamente) necessário para atingir o nível de desempenho necessário?

Frustração - FR
O quão inseguro, desencorajado, irritado, estressado ou seguro, contente, relaxado e satisfeito você se sentiu durante a tarefa.

Demanda física - PD
O quanto de atividade física é requerida por essa atividade (ex: apertando, pulando, girando, ativando)? Foi trabalhosa/ativa/desgastante ou foi tranquilo/repousante?

1 – Circule o membro de cada par que mais influência no uso do sistema proposto:

PD ou MD	TD ou PD	TD ou FR
TD ou MD	OP ou PD	TD ou EF
OP ou MD	FR ou PD	OP ou FR
FR ou MD	EF ou PD	OP ou EF
EF ou MD	TD ou OP	EF ou FR

2 – Coloque o "X" em cada escala que represente a magnitude de cada fator (Não Preencher os campos em Cinza):

Fatores	**Avaliação de Demanda**													**Nota**	**Peso**	**Produto**
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**				
MD	**Baixo**												**Alta**	**80**		
PD	**Baixo**												**Alta**	**20**		
TD	**Baixo**												**Alta**	**50**		
OP	**Excelente**												**Ruim**	**60**		
FR	**Baixo**												**Alta**	**20**		
EF	**Baixo**												**Alta**	**30**		
														Soma		
														Média Ponderada		