UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Máster en Ingeniería de Telecomunicación

TRABAJO DE FIN DE MÁSTER

CONTROL ADAPTADO AL SUJETO DE UNA INTERFAZ CEREBRO-MÁQUINA CON SEÑALIZACIÓN MIXTA

Ana Sotomayor Romillo
Tutor: Pablo Varona Martínez

JUNIO 2016

CONTROL ADAPTADO AL SUJETO DE UNA INTERFAZ CEREBRO-MÁQUINA CON SEÑALIZACIÓN MIXTA

AUTOR: Ana Sotomayor Romillo

TUTOR: Pablo Varona Martínez

Grupo de Neurocomputación Biológica

Dpto. de Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Junio de 2016

Agradecimientos

Y ahora sí, esto se acaba.

En primer lugar, me gustaría darte las gracias a ti, que estás leyendo este trabajo. Gracias por tu tiempo y tu atención. Me gustaría también que supieras quienes son los culpables de que hoy esto esté acabado.

Detrás de este trabajo está Pablo, mi tutor, que me dio la oportunidad de comenzar este trabajo y la confianza para hacerlo a mi manera. Está por supuesto Víctor, compañero de laboratorio, de viernes, sábados y algún que otro domingo. Gracias también a Paco, David, Víctor y a todo el equipo del GNB por su ayuda siempre que ha hecho falta.

Detrás de estos seis años están todos y cada uno de mis compañeros, mis amigos, mis Telecos. Las lecciones más importantes las he aprendido de vosotros. Qué suerte haber compartido cada clase, examen, entrega, comida, cena, excursión, partido, fiesta, viaje y tontería con gente tan extraordinaria. Gracias a Sara, Patri y Marta, mi equipo de remo. Y gracias Lenny por tus absurdeces y tu incondicionalidad.

Detrás de este año están Joan, Jontxu, Jero, Myriam, mi compi de ZCM Alex, José, Celia y su arte en la maquetación, Mery, Marta, Silvia, Jaimuchi, Elena, Cris, Dani, Cris, Maria Jesús, mi tutor Manolo, Luis, Eva, Alberto, Javi, Peter, José, Edu, Mickey, Maite, Alberto, Rangel y mi compañera, mentora y amiga Nerea. No se me ocurre mejor compañía para dar mis primeros pasos en el mundo laboral. Si tuviera la oportunidad de volver un año atrás, os volvería a elegir a vosotros.

Desde que tengo uso de razón han estado a mi lado seis personas que hacen que sea quien soy. Ana, Isa, Lidia, Ana, Bea, Paula. Mis amigas, las de siempre. Gracias por ser eso que sé que nunca va a cambiar.

Detrás de cada bache y en cada momento ha estado siempre mi incondicional, extraordinaria, única y divertidísima familia. Qué gran ejemplo, y qué gran lección de vida. Gracias por las comidas de los sábados, el Romilleo, los karaokes, las rumbas, la fiesta, los gritos, el desorden y por encima de eso el apoyo siempre, en todo momento.

Pero por encima de todo, detrás de este trabajo has estado Tú que, sin estar, estás. Tú que has creído en mí cuando yo no he sido capaz. Tú que no me has puesto nunca el camino fácil, pero lo recorres a mi lado. Esto, como todo, es para ti. Gracias.

Y ahora sí, esto se acaba.

Resumen

Una Interfaz Cerebro – Máquina o Brain Computer Interface (BCI) es un sistema de comunicaciones que permite controlar un ordenador o dispositivo externo a partir de la actividad cerebral. A lo largo de este trabajo se presenta el desarrollo un BCI en tiempo real en la que el control se realiza mediante una señalización mixta para generar potenciales visuales evocados (SSVEP). Esto quiere decir que, de cara a maximizar los resultados que ésta interfaz pueda proporcionar, se realizará una combinación entre un circuito hardware que controla los LEDs de estimulación visual y una aplicación software que informa en pantalla sobre la codificación del estímulo, ambos funcionando de manera independiente pero estando interconectados entre sí.

El objetivo de este trabajo será la realización de una Interfaz Gráfica de Usuario, en adelante GUI, que permita la integración de todos los elementos que componen este BCI, de manera que se construya un dispositivo compacto e integrado, siendo al mismo tiempo un diseño flexible, en el sentido de que sea posible la sustitución de un elemento por otro con la misma funcionalidad sin perder las prestaciones de la interfaz completa. La interfaz utiliza una señalización mixta que permite simultáneamente mostrar información en la pantalla de un ordenador y de forma coordinada realizar una estimulación visual mediante LEDs para evocar los SSVEPs que se utilizan en el control. Además, permite realizar una adaptación de las mejores frecuencias de estimulación para cada sujeto, resultando en un mejor control de la misma.

Para lograr estos objetivos, se comenzará realizando un estudio del estado del arte, de cara a entender qué es una Interfaz Cerebro – Máquina y cuáles son sus componentes, las características de los mismos y el grado de avance en la actualidad de estas interfaces.

Una vez entendido este concepto, planteará el diseño del BCI que se pretende construir, analizando ventajas e inconvenientes de cada línea de desarrollo y las características de los elementos que se van a seleccionar. Concluido el planteamiento definitivo, se procederá a explicar cómo se ha desarrollado el circuito hardware y su comunicación con el resto de elementos del BCI, así como los pasos que se han seguido para la construcción de la GUI y su integración con los programas encargados de recibir, procesar, analizar y proporcionar los resultados obtenidos con el registro de electroencefalograma.

Construidos y desarrollados todos sus elementos, el resultado será una Interfaz Cerebro – Máquina basada en SSVEPs que será necesario validar. Para ello, se realizarán pruebas tanto funcionales como con voluntarios que determinen si la misma cumple con los requisitos fijados al comienzo de este trabajo y si es posible su utilización como medio de comunicación.

Por último, se realizará un análisis a posteriori que determine si se han cumplido los requisitos establecidos y qué conclusiones se pueden extraer de este desarrollo, finalizando con las líneas de estudio futuro que se van a seguir para continuar avanzando en esta línea de investigación.

Abstract

A Brain Computer Interface is a means of communication based on neural activity with the aim of controlling a computer or external device. Throughout this document the development of a real-time Brain Computer Interface or BCI is provided, whose control is performed by an SSVEP based mixed signaling. This means that, in order to achieve the best results that this interface can achieve, a combination of a hardware circuit for the LED stimulation control and a software application to provide additional information will be implemented, both of them working in a standalone mode but capable of communicating between each other.

The aim of the project is to build a Graphic User Interface, hereinafter GUI, allowing the integration of all the BCI elements, developing a compact and integrated device that, at the same time, is built in a flexible manner, meaning that any change of the interface elements by one with the same features will not affect the BCI properties. The BCI combines the delivery of information to the user regarding the stimulus encoding and the stimulation through hardware controlled LEDs. Moreover, an adaptation to each user will be carried out, which will result in a better control of the interface.

To achieve these goals, the first step will be a study of the current situation, leading to a better understanding of what a BCI is, which are its components and features and the progress of these interfaces.

Once this concept has been engaged, the BCI design will be detailed, determining its benefits and drawbacks of each development and the features of each selected element. Once the final approach has been determined, an explanation of the hardware design will be provided, determining at the same time the means of communication between this circuit and the other elements. Along that chapter, the steps for building the GUI will be as well provided, together with the integration with the programs that are in charge of receiving, processing, analyzing and bringing up the results recorded by the electroencephalography.

With all the elements built and developed, the final result will be the mixed signaled Brain – Computer Interface that needs to be upheld. Functional and user acceptance tests will be carried out, determining whether the requirements are met and if it is possible to use this BCI as a means of communication.

Finally, a posteriori analysis will determinate if the goals of this project have been accomplished and which conclusions can be extracted of this development, concluding with the next steps that will be taken in this research line.

Lista de Palabras Clave

Interfaz cerebro-máquina (BCI), electroencefalograma (EEG), Steady-State Visual Evoked Potentials (SSVEP), artefacto, BCI Illiteracy, arduino, LED, Emotiv EPOC, Qt, Interfaz Gráfica, Señalización mixta, adaptación al sujeto

Key Words

Brain Computer Interface (BCI), electroencephalogram (EEG), Steady-State Visual Evoked Potentials (SSVEP), Graphic User Interface, Arduino, LED, Emotiv EPOC, Qt, Graphical interface, hybrid signaling, user adaptation

Glosario

Interfaz Cerebro Máquina (BCI)

Un BCI es un sistema de comunicaciones hardware y/o software que permite controlar un ordenador o dispositivo externo únicamente mediante la actividad cerebral, proporcionando un medio de comunicación a personas con algún tipo de discapacidad que, de otra manera, no tendrían manera de expresar sus intenciones [1] [2]

Electroencefalograma (EEG)

El electroencefalograma mide la actividad eléctrica en el cerebro causada por el flujo de corrientes eléctricas durante las excitaciones sinápticas de las dendritas en las neuronas [3]

Steady – State Visually Evoked Potentials (SSVEP)

Los "Visual Evoked Potentials" (VEP) son modulaciones en la actividad del cerebro que suceden en el córtex visual después de recibir un estímulo visual. Estas modulaciones son relativamente fáciles de detectar, ya que la amplitud de los VEPs se incrementa considerablemente cuando el estímulo se acerca al centro del campo de visión [4]. Los "Steady-State Visually Evoked Potentials" (SSVEP) son VEPS en respuesta a estimulación en frecuencias específicas que generan actividad a la misma frecuencia que tiene el estímulo [5]

Artefacto

Esta palabra, utilizada en el ámbito del trazado de un aparato registrador, se refiere a toda variación no originada por el órgano cuya actividad se desea registrar [6]

BCI Illiteracy

Fenómeno que denomina el hecho de que una interfaz cerebro –máquina no funcione para un porcentaje de sujetos no despreciable, de alrededor de entre un 15% y un 30% [7]

Arduino

Arduino es una plataforma de hardware libre, de bajo coste y altas prestaciones, basada en una placa con un microcontrolador y un entorno de desarrollo sencillo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares [8]

LED

De las siglas Light-Emiting diode, 'diodo emisor de luz', se trata de un dispositivo de señalización utilizado en electrónica, ordenadores, paneles numéricos, etc. En este proyecto se utilizan LEDs para evocar SSVEPs y poder reconocer la intención de una persona en cuanto al manejo de un BCI [9]

Emotiv EPOC

Emotiv EPOC es un casco comercial de registro de EEG que actúa como una interfaz personal para que las personas interactúen con ordenadores. Realiza un registro de electroencefalografía de alta resolución multicanal, utilizando sensores que transforman la actividad cerebral en señales eléctricas [10]

Interfaz Gráfica de Usuario

Una Interfaz Gráfica de Usuario, referida en este documento como GUI, es un conjunto de imágenes y objetos gráficos que se utilizan para representar la información y acciones disponibles en una interfaz. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador, y en este caso habilitará la comunicación en la Interfaz Cerebro – Máquina [11]

Qt

Qt es una amplia plataforma de desarrollo que incluye clases, librerías y herramientas para la producción de aplicaciones de interfaz gráfica en C++ que pueden operar en varias plataformas [12]

Señalización Mixta

Cuando en este documento se habla de señalización mixta, se hace referencia a que, de cara a lograr detectar estímulos en el registro de actividad cerebral, se diseñará por un lado un sistema de generación de estímulos mediante un circuito hardware que controla los LEDs de estimulación, que funcionará de manera independiente al ordenador. Por otro lado, se implementará una GUI mediante software que señalice al usuario las opciones que ha escogido y hacia qué estímulo debe mirar. Estos dos sistemas funcionarán de manera independiente pero estarán intercomunicados y coordinados entre sí

Adaptación al sujeto

Las señales registradas en la región occipital varían en función del usuario que esté utilizando el BCI basado en SSVEPs. Se ha demostrado en estudios previos [13] que para cada sujeto, existen una serie de frecuencias que se detectan con una mayor amplitud y precisión que otras. Realizando una adaptación de frecuencias previa a la utilización de un BCI se logra maximizar los resultados obtenidos, en lugar de utilizar la opción alternativa que sería establecer unas frecuencias prefijadas para todos los sujetos

Índice

Índice de Tablas

Índice de Figuras

CAPÍTULO 1. INTRODUCCIÓN

A lo largo de este documento se va a proceder a explicar el trabajo realizado, en el que se ha desarrollado un control adaptado a cada sujeto de una Interfaz Cerebro-Máquina basada en SSVEPs utilizando una señalización mixta.

En este capítulo se comenzará exponiendo la **motivación** que ha llevado al planteamiento, diseño y posterior desarrollo de este trabajo. Entendida la problemática, se abordarán los **objetivos** que en un primer momento se han fijado y que se pretenden alcanzar durante la elaboración de este trabajo. Por último, para facilitar el seguimiento de este documento se describirá la **estructura** del mismo.

1.1 Motivación

A lo largo de los últimos años, la tecnología ha dado un giro radical con respecto a las líneas de evolución que se habían seguido durante las décadas anteriores. Sólo durante los últimos 2 años, la cantidad de información que se ha generado supone el 90% de toda la información histórica del ser humano [14], y se prevé que esta evolución seguirá creciendo de esta manera exponencial.

Hace tan solo unos años, tecnologías como el reconocimiento de voz, de patrones o de seguimiento ocular estaban muy distantes de ser perfeccionadas, y a día de hoy podemos verlas integradas con nuestro día a día. Se hace patente entonces que cada vez el medio de comunicación entre el ser humano y los ordenadores adquiere alternativas diferentes a los tradicionales ratón y teclado, que han sido la interfaz humano-ordenador durante los últimos 40 años, sencillos pero poco eficientes y distantes de los métodos naturales e intuitivos que el ser humano dispone para comunicarse.

Así pues, cada vez más nos acostumbramos a medios de comunicación basados en pantallas táctiles, dispositivos de reconocimiento de voz o de seguimiento ocular, rompiendo las barreras que separaban las máquinas de los seres humanos e integrándolos con cada acción que llevamos a cabo cada día.

Estas nuevas interfaces, sin embargo, resultan inútiles para personas discapacitadas o con alguna minusvalía que limite su desarrollo motriz. Sin forma de comunicarse verbalmente y sin capacidad de realizar movimientos físicos voluntariamente, este sector de personas queda excluido de la utilización de dichas interfaces y consecuentemente de hacer uso de la tecnología, entre otras muchas actividades.

En esta línea de investigación, se han desarrollado nuevos dispositivos que actúen como interfaces entre el ser humano y la tecnología, permitiendo recibir y procesar nuevos parámetros de entrada además de los que se han mencionado anteriormente. Otro dispositivo que surge a raíz de este problema es la Interfaz Cerebro-Máquina (BCI), que puede tomar como punto de partida para la adquisición de datos un electroencefalograma (EEG). Registrando la actividad cerebral de un sujeto,

podemos traducir esas señales en instrucciones y ser capaces de controlar un ordenador o dispositivo. Estas interfaces cerebro-máquina tienen como objetivo principal proporcionar un mecanismo de comunicación a personas con discapacidades que, de otra forma, no tendrían manera de expresar sus intenciones [2].

No obstante, dependiendo de cada persona y de cada set de instrucciones, el sistema BCI a utilizar será efectivo en mayor o menor medida. Algunos BCI no serán válidos para algunas personas, mientras que otros sí que lo son [7], [15].

El propósito de este trabajo es desarrollar una interfaz gráfica con señalización híbrida a partir de un dispositivo de estimulación SSVEP mixto software-hardware que funcione en tiempo real, de manera que se pueda establecer un medio de comunicación con personas que de otra forma no tendrían forma de hacerse entender.

1.2 Objetivos

Durante el Trabajo de Fin de Grado "Diseño de interfaces cerebro-máquina y hombre-máquina controlados por señalización biológica"[2] realizado en el año 2014, el objetivo fue el diseño de dispositivo de bajo consumo y bajo coste que, integrándose en un BCI, garantizase la correcta configuración y conectividad con unos LEDs a través de un controlador que permitiese además una comunicación inalámbrica, llegando como resultado a un dispositivo compacto de pequeñas dimensiones.

Los objetivos han cambiado considerablemente para este Trabajo de Fin de Máster, buscando una gran evolución con respecto a trabajos anteriores y persiguiendo los siguientes hitos:

- El objetivo de este Trabajo de Fin de Máster es la creación de una Interfaz Cerebro – Máquina **en tiempo real** que sea universal y **flexible,** tanto con respecto a sus aplicaciones como sistemas de estimulación y métodos de adquisición.

- Se implementará para ello una **interfaz gráfica de usuario** que permita realizar una **señalización mixta,** es decir, que habilite la comunicación a través de una señalización por pantalla que informa sobre la codificación del estímulo y otra simultánea mediante un hardware externo de control de los LEDs, siendo ambas independientes pero capaces de **intercomunicarse** entre sí.

- Se crearán nuevas funcionalidades que el usuario podrá seleccionar, que permitirán utilizar este BCI como **sistema de comunicación flexible.**

- Por último, se busca que esta interfaz realice una **selección de estímulos adaptada a cada sujeto,** por lo que se trabajará en la búsqueda de aquellas frecuencias que maximicen las respuestas de los usuarios.

1.3 Estructura de la Memoria

Este documento se ha dividido en **seis capítulos** que, en su conjunto, dan sentido y permiten entender la problemática, el desarrollo y la validación del dispositivo que se ha creado. A continuación se resume cada uno de ellos:

- Comenzando por el **"CAPÍTULO 1 Introducción "**se realiza una descripción inicial de las razones que han llevado a realizar este trabajo y los objetivos que se pretenden alcanzar.

- A lo largo del **"CAPÍTULO 2 Estado del Arte"** se recopila la información clave para entender la situación actual de la investigación en torno a las Interfaces Cerebro – Máquina, las diferentes maneras que existen para su implementación y las etapas y componentes necesarios.

- Tras dicho análisis, se procederá a explicar el trabajo realizado, comenzando con un boceto inicial de la interfaz que se pretende construir y siguiendo con un análisis de las diferentes posibilidades de implementación. Tras seleccionar los elementos, se describirá el desarrollo de cada una de las partes involucradas, así como la comunicación entre las mismas. Todo esto se describirá en el **"CAPÍTULO 3 Diseño y Desarrollo".**

- Será necesario validar el trabajo realizado, y por esta razón se incluyen en el **"CAPÍTULO 4 Experimentos y Resultados"** las pruebas realizadas en cada etapa de diseño para verificar que el dispositivo cumple con los requisitos esperados.

- Por último, en el **"CAPÍTULO 5 Conclusiones y Estudio Futuro"** se enumeran las conclusiones observadas tras la elaboración de este trabajo, así como las líneas de trabajo futuro que se van a seguir y algunas alternativas de mejora.

CAPÍTULO 2. ESTADO DEL ARTE

El objetivo de este capítulo es adquirir un entendimiento global sobre el funcionamiento y características de los sistemas BCI, así como las diferentes técnicas que se utilizan en la actualidad.

Se quiere comenzar el estado del arte haciendo un análisis de la situación actual de las interfaces Cerebro – Máquina, de manera que permita comprender la motivación y logros que se pueden alcanzar con el desarrollo de este proyecto.

Para ello, en una primera aproximación se definirán y explicarán los principios de los BCIs, continuando con el sistema de adquisición elegido para este caso: el electroencefalograma (EEG). Se estudiarán los principios de un EEG, así como los diferentes electrodos y artefactos asociados a los mismos y cómo puede utilizarse para determinar la intención de una persona en base a su actividad cerebral.

Se estudiará también en este capítulo qué son y por qué se utilizan los Steady-State Visual Evoked Potentials (SSVEP) para el diseño de BCIs, qué tipos existen y qué resultados se pueden obtener en función del método de implementación escogido.

2.1 BCI: Interfaz Cerebro – Máquina

Para entender la motivación y objetivos de este trabajo es necesario comenzar estudiando en profundidad **qué es un BCI**. Una interfaz cerebro-máquina (BCI) es un sistema de comunicaciones que permite controlar un ordenador o dispositivo externo únicamente mediante la actividad cerebral. El objetivo de la investigación de estas interfaces es principalmente poder proporcionar capacidades de comunicación a personas con serias discapacidades que han resultado en una parálisis total o un bloqueo causado por un trastorno neuromuscular y neurológico, tales como la esclerosis lateral amiotrófica o una lesión de la médula espinal [16], pero cuyas capacidades cognitivas y de razonamiento no hayan sido alteradas.

La interfaz cerebro-máquina permite al usuario interactuar con su entorno sin tener que hacer uso de nervios periféricos y músculos, sino únicamente a partir de las señales de control generadas por la actividad encefalográfica. De esta forma, se crea un nuevo canal, en el que no intervienen los músculos, para transmitir las intenciones de una persona a dispositivos externos como ordenadores, sintetizadores de voz, dispositivos de asistencia y prótesis neurales. Estas características permitirán a aquellas personas que sufren discapacidades motoras mejorar su calidad de vida incrementando su capacidad de comunicación, y al mismo tiempo reducir el coste de los cuidados intensivos [17].

2.1.1 Etapas de un BCI

Una Interfaz Cerebro-Máquina es un sistema de inteligencia artificial que es capaz de reconocer unos determinados patrones en las señales obtenidas del cerebro, siguiendo cinco etapas consecutivas: primero se adquiere la señal a través de distintos sistemas de medición. Después, la señal obtenida se preprocesa para mejorar la señal y se extraen sus características, que se interpretan y clasifican para obtener la orden que el usuario está mandando, y por último se manda dicha orden a la interfaz de control.

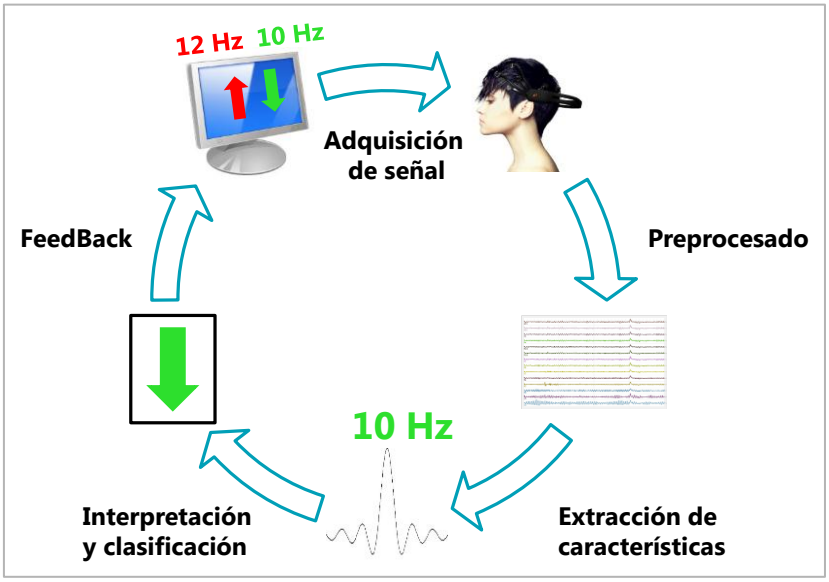

Figura 1. Etapas de funcionamiento de una Interfaz Cerebro – Máquina (adaptada de [2])

La primera etapa es la de **adquisición de señal**. En el estado del arte de la tecnología de BCIs, existen dos tipos principales de actividad cerebral que pueden ser monitorizadas por distintos sistemas de obtención de señal:

- La primera es la **actividad electrofisiológica**, que se genera cuando dos neuronas intercambian información mediante transmisores electroquímicos. Estas neuronas generan corrientes iónicas que fluyen dentro y a través del resto de neuronas, y estas corrientes pueden ser registradas mediante sistemas como el electroencefalograma (EEG), electrocorticografía, magnetoencefalografía o un sistema de adquisición de señales eléctricas en neuronas simples [18].

- El segundo tipo de actividad cerebral que se puede monitorizar es la **respuesta hemodinámica** [19]. La respuesta hemodinámica es el proceso mediante el cual la sangre libera más glucosa a las neuronas activas que a las inactivas. La glucosa y oxígeno que se entrega a través del flujo de sangre en las zonas activas del cerebro se pueden medir con métodos de neuroimagen como por ejemplo una resonancia magnética o una espectroscopia infrarroja.

Para la mayor parte de los sistemas BCI se elige la actividad electrofisiológica como señal a procesar, ya que se considera un sistema directamente relacionado con la actividad neuronal, al contrario que la respuesta hemodinámica que se considera indirecta. Así pues, el siguiente paso es elegir **qué sistema se va a utilizar** para monitorizar dicha actividad neuronal.

La mayoría de los BCI utilizan comúnmente el **electroencefalograma**, que tiene una alta resolución temporal, bajo coste, portabilidad alta y poco riesgo para el usuario. Existen otros métodos de obtención de la señal, como el magnetoencefalograma, electrocorticograma o sistemas de adquisición de señales eléctricas en neuronas individuales [16], [20], [21].

- El **magnetoencefalograma** registra la actividad magnética en lugar de la eléctrica a través de una inducción magnética. Tiene menos distorsiones provocadas por el cráneo y el cuero cabelludo, pero no son sistemas portátiles y son muy costosos, resultando imposibles para el uso diario [22].

- El **electrocorticograma** tiene la ventaja de tener una alta resolución tanto espacial como temporal, así como mayores amplitudes y menor vulnerabilidad a artefactos, pero se trata de un sistema invasivo en el que los electrodos se colocan directamente en la superficie del cerebro [23]. Los registros invasivos se utilizan únicamente en pacientes con patologías severas.

- Por último, el sistema de adquisición de **señales eléctricas en neuronas individuales** registra la actividad cerebral dentro de la materia gris del cerebro. Se trata también de un sistema invasivo, teniendo una resolución tanto temporal como espacial alta, y las señales obtenidas son más fáciles de procesar que las de un EEG. Sin embargo, la calidad de la señal se puede ver afectada por la reacción del tejido cerebral, y el microelectrodo se daña con el tiempo [24].

A continuación se expone una tabla resumiendo las propiedades de cada sistema de adquisición de señal, con sus ventajas e inconvenientes:

Método	Actividad registrada	Resolución temporal	Resolución espacial	Riesgo	Portabilidad
EEG	Eléctrica	~0.05s	~10 mm	No invasivo	Portátil
MEG	Magnética	~0.05s	~5 mm	No invasivo	No Portátil
ECoG	Eléctrica	~0.003s	~1 mm	Invasivo	Portátil
Adquisición en neuronas individuales	Eléctrica	~0.003s	~0.5mm(LFP) ~0.1mm(MUA) ~0.05mm(SUA)	Invasivo	Portátil

Tabla 1. Métodos de adquisición de señal en un BCI (adaptada de [16])

Con el electroencefalograma realizamos la primera etapa (adquisición de señal), pero para las siguientes etapas de preprocesamiento y extracción de características es necesario determinar qué señal va a controlar las intenciones e instrucciones del usuario.

Las señales que emite el cerebro incluyen todas tareas que se realizan en el mismo en cada momento, la mayoría de ellas aún desconocidas e incomprensibles para el ser humano. Sin embargo, el origen fisiológico de algunas señales sí que han sido decodificadas de forma que las personas puedan aprender a modularlas según su voluntad, permitiendo al sistema BCI interpretar las órdenes que el usuario esté mandando, y son éstas señales las que son objeto de estudio y posibles señales de control. Según su origen, las señales se pueden clasificar como señales endógenas o exógenas.

- Son señales **exógenas** aquellas en las lo que genera la instrucción es un estímulo externo, como por ejemplo un LED parpadeante. Estos sistemas tienen la ventaja de que no requieren de un entrenamiento excesivo y resulta fácil y rápido, aunque necesitan que se preste una atención permanente al estímulo externo, lo que provoca cansancio y molestias en algunos sujetos. Los sistemas que generan estímulos exógenos son los VEP y los P300.

  - Los **VEPs** son modulaciones que ocurren en el córtex al recibir un estímulo visual [25], como puede ser un LED oscilando a una determinada frecuencia o una señal a través del monitor. Estas modulaciones se pueden detectar con facilidad y permiten un elevado número de elecciones a la hora de asignar estímulos a instrucciones.

  - Por otro lado, el **P300** es también un estímulo exógeno en el que se muestran al usuario varias opciones que se van resaltando una cada vez. Cuando la instrucción que el usuario quiere escoger es la que está resaltada, aparece un pico en el electroencefalograma, y el sistema detecta que ese es el estímulo escogido [26].

- Las señales **endógenas** surgen del propio sujeto, que mediante un entrenamiento y práctica aprende a generar patrones cerebrales específicos que se pueden registrar en el BCI. Este método tiene la ventaja de que una vez aprendido es independiente de cualquier estimulación, se puede ejecutar con libertad y es muy útil para personas con órganos sensoriales afectados, pero el inconveniente de que se puede tardar meses o años en aprender a manejarlo, y no todos los usuarios pueden aprender a controlarlo. Además, se necesitan EEG multicanales para un buen rendimiento, y la tasa de bits es más baja que en los exógenos. Los sistemas que utilizan señales endógenas son los SCPs y los ritmos sensoriomotores.

  - Los **SCPs (Slow Cortical Potentials)** son un tipo de señal endógena consistentes en en una ondulación lenta en el EEG que dura entre un segundo y varios segundos. Son señales que proceden de la actividad cortical, y pueden ser negativas o positivas, siendo esa la diferencia entre las dos órdenes que un SCP puede detectar [27].

  - Los **ritmos sensoriomotores** son un tipo de señal endógena consistente en modulaciones relacionadas con actividades motoras, es decir, para generar una

orden basta con que el sujeto piense en realizar un movimiento. Aunque dicho movimiento no llegue a realizarse, al concentrarse en ello genera actividad que el EEG puede registrar y traducir en la orden [24].

El sistema que la mayoría de BCIs utiliza para generar la señal que posteriormente se extraerá con el electroencefalograma será el **SSVEP**, debido al elevado número de instrucciones que se pueden implementar, su facilidad a la hora de aprender y su elevada tasa de transmisión de información.

Señal	Tipo	Número de elecciones	Entrenamiento	Tasa de transmisión de información
VEP	Exógena	Altas	No	60-100 bits/min
P300	Exógena	Bajas (2 o 4)	No	20-25 bits/min
SCP	Endógena	Altas	Sí	5-12 bits/min
Ritmos Sensoriomotores	Endógena	Bajas (2-5)	Sí	3-35 bits/min

Tabla 2. Comparativa de procedimientos de estimulación (adaptado de [16])

Después de determinar qué características son las que se están buscando, los últimos pasos son **extraer dichas características** y **clasificar** la señal que será enviada a la interfaz de control.

Debido a que en el cerebro en cada instante se realizan una variedad de tareas que incluyen pensamientos, órdenes motoras voluntarias e involuntarias, coordinación de las actividades del resto del organismo entre otras tareas, se hace complicado distinguir una característica determinada en una señal ya que está solapada con el resto de tareas mencionadas anteriormente. Por esta razón, a la hora de extraer las características, es necesario algo más que un método simple como podría ser un filtro paso banda. La información se recibe a través de varios canales, no todos ellos de interés, así que se utilizan técnicas de reducción como el análisis de componentes principales o análisis de componentes independientes, que reducen las dimensiones de los datos originales eliminando y reduciendo la información redundante [28].

Para clasificar la información ya analizada y reducida, se pueden utilizar tanto algoritmos de **regresión** como de **clasificación**, siendo los algoritmos de clasificación los más utilizados [29]. Los algoritmos de regresión se basan en las características extraídas para predecir la intención del sujeto, mientras que los de clasificación separan esas características, definiendo una frontera entre las instrucciones.

En la figura 2, se puede observar un ejemplo de los algoritmos de clasificación y de regresión. En la gráfica de la izquierda, aparecen los datos que se quieren clasificar. Mientras que un algoritmo de clasificación agruparía los datos en bloques o clusters, el de regresión calcularía la recta donde es más probable que caigan los nuevos datos que puedan aparecer.

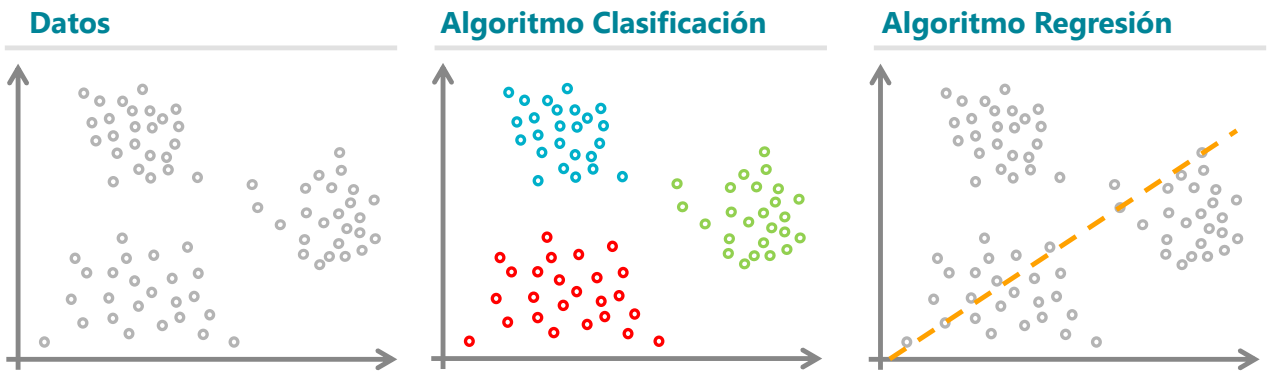

Figura 2. Algoritmo de Clasificación VS Algoritmo de Regresión Lineal

Para un caso en el que hay que determinar una instrucción entre dos posibles, tanto clasificación como regresión precisan de una sola función. Sin embargo, para cuatro instrucciones posibles, el algoritmo de clasificación necesitaría tres funciones, una por cada frontera, mientras que para regresión seguiría bastando con una. Por este motivo suele utilizarse regresión para casos en los que hay varias instrucciones, y clasificación para instrucciones de dos casos. Además, la aproximación por regresión resulta útil en casos en los que haya un feedback continuo, como el control de un cursor [30].

2.1.2 Aplicaciones

Una vez concluidas las etapas que conforman el funcionamiento de un BCI, cabe mencionar cuál es su función y en qué aplicaciones resulta beneficioso. Dependiendo del sujeto que lo va a utilizar, se pueden distinguir dos tipos de BCIs: el grupo principal está enfocado hacia aquellos que serán utilizados por personas con discapacidad hasta un grado de parálisis casi total, pero con los suficientes recursos para controlar el sistema. Existe también una línea de investigación hacia BCIs más rápidos que se utilizarían para personas en plena posesión de sus capacidades. En este caso las aplicaciones BCI son limitadas, ya que en este contexto son más prácticos y fáciles de utilizar los tradicionales ratón y teclado, así que estos BCIs se utilizan en su mayoría con fines de entretenimiento, por ejemplo en videojuegos.

Las principales aplicaciones de los BCIs para las personas con discapacidad son aquellas relacionadas con la comunicación, ya que es una actividad esencial de los seres humanos. En este ámbito, se han creado teclados virtuales en los que el usuario puede seleccionar una letra del alfabeto a través del BCI, por ejemplo a través de un P300 [15], [31]. También se pueden controlar cursores mediante SCPs que se muevan sobre el teclado y seleccionen la letra [32].

Otro campo de aplicaciones BCI es el de la restauración de la actividad motora. Se pueden crear dispositivos que, mediante instrucciones ordenadas por el usuario, puedan por ejemplo agarrar y transportar objetos, así como manejar sillas de ruedas [33]. También se puede controlar el medio en el que viven, permitiendo manejar mandos de televisión, encender y apagar luces de la casa o la temperatura [16].

2.2 EEG: Obtención de la Señal

La actividad cerebral se registra para determinar las intenciones del usuario. Para esto, el BCI necesita de un sistema que monitorice esta actividad cerebral, y posteriormente traduzca esta información a señales eléctricas manejables. Se han mencionado diferentes métodos para conseguir estas señales, y el más utilizado de ellos es el electroencefalograma (EEG).

Un elecroencefalograma mide la actividad eléctrica en el cerebro causada por el flujo de corrientes eléctricas durante las excitaciones sinápticas de las dendritas en las neuronas, y es extremadamente sensible a los efectos de corrientes secundarias [3]. Las señales de un EEG se registran fácilmente de forma no invasiva a través de unos electrodos colocados en el cuero cabelludo. El hecho de que la señal tenga que cruzar el cuero cabelludo, el cráneo y demás capas hace que la señal tenga una calidad baja. También afecta a la calidad de la señal el ruido de fondo generado tanto en el interior del cerebro como exteriormente en el cuero cabelludo.

Las neuronas están polarizadas eléctricamente por su membrana de transporte, que se encarga de bombear iones. Estas células están constantemente intercambiando iones con el medio extracelular y produciendo corrientes eléctricas. Mediante un casco de EEG se puede medir la diferencia de potencial a lo largo del tiempo entre el electrodo activo y el electrodo de referencia. Está técnica de medida se conoce como registro bipolar [3].

También existe un tercer electrodo, conocido como electrodo de tierra, que se utiliza para medir la diferencia de potencial entre el punto activo y el de referencia. Estos tres electrodos componen la configuración mínima para registrar una señal EEG correctamente, y se conocen como registro unipolar [16]. En aplicaciones clínicas, se utilizan típicamente 19 electrodos colocados alrededor del cráneo para realizar las mediciones, aunque si es necesario aumentar la resolución espacial en un área particular del cerebro se puede aumentar el número de electrodos, llegando hasta 256 electrodos colocados de forma relativamente uniforme en el cuero cabelludo.

Un sistema EEG está compuesto de electrodos, amplificadores, conversores A/D y un dispositivo de grabación. Los electrodos adquieren la señal del cuero cabelludo, y los amplificadores procesan la señal analógica para aumentar la amplitud de la señal EEG para que el conversor A/D pueda digitalizar la señal con mayor precisión. Finalmente, el dispositivo de grabación, que puede ser un ordenador personal, almacena y procesa estas señales.

2.2.1 Clasificación de los electrodos de un EEG

Existen diferentes tipos de electrodos que pueden conformar un EEG, que se clasifican en función del **material** con el que están hechos. Los más comunes son los de plata y oro, existiendo también electrodos hechos de estaño, más baratos pero menos usados hoy en día.

Los electrodos también pueden clasificarse en **activos o pasivos**. Los electrodos activos tienen en su interior un amplificador, que mejora la calidad de la señal y evita tener que poner geles conductores en el cuero cabelludo. Los pasivos son más simples y tienen un menor coste, pero requiere que cada electrodo sea acondicionado manualmente para captar la actividad cerebral, y precisa de geles conductores.

2.2.2 Artefactos

Al analizar las señales registradas por el EEG hay que tener en cuenta los distintos artefactos que pueden surgir en las señales. Se denomina artefacto a toda variación no originada por el órgano cuya actividad se desea registrar. Los datos registrados en un EEG casi siempre están contaminados por estos elementos, y su amplitud puede llegar a ser relativamente amplia en comparación con las señales que resultan de interés. Estos artefactos se pueden clasificar en dos categorías: artefactos y técnicos artefactos fisiológicos [16].

- Los **artefactos técnicos** se atribuyen en su mayoría a cambios en la impedancia de los electrodos y a ruidos en la potencia de la línea, que normalmente se pueden evitar mediante un correcto filtrado y protección. Debido a que tienen una solución más fácil, los estudios se suelen centrar en los artefactos fisiológicos, que suponen un reto mayor al distorsionar más la señal obtenida.

- Los **artefactos fisiológicos** se deben a actividades musculares, oculares o cardiacas, como por ejemplo el movimiento de los ojos, parpadeo, movimientos musculares involuntarios extraoculares, el latido del corazón, contracción de la mandíbula… Cuando el movimiento es voluntario, estos artefactos se pueden corregir simplemente pidiendo al sujeto que evite dicho movimiento, como por ejemplo parpadear lo menos posible. Sin embargo, artefactos como el pulso cardiaco son involuntarios y no se pueden corregir de esta manera. En este caso, se pueden monitorizar las señales de los artefactos de forma separada, y sustraer esta señal de la señal registrada por el EEG.

2.3 SSVEP: Steady – State Visually Evoked Potentials

A la hora de registrar correctamente la orden que se registra en el EEG, a menudo se utilizan técnicas exógenas (mencionadas en el apartado "2.1 BCI: Interfaz Cerebro – Máquina") que provocan en el cerebro la respuesta buscada.

Para generar este estímulo, una técnica utilizada son los "evoked potentials", en los que se registra la actividad cerebral mientras se presenta un estímulo, ya sea visual, auditivo o somatosensorial. Como se ha definido anteriormente, los "Visual Evoked Potentials" (VEP) son modulaciones en la actividad del cerebro que suceden en el córtex visual después de recibir un estímulo visual [25]. Estas modulaciones son relativamente fáciles de detectar, ya que la amplitud de los VEPs se incrementa considerablemente cuando el estímulo se acerca al centro del campo de visión.

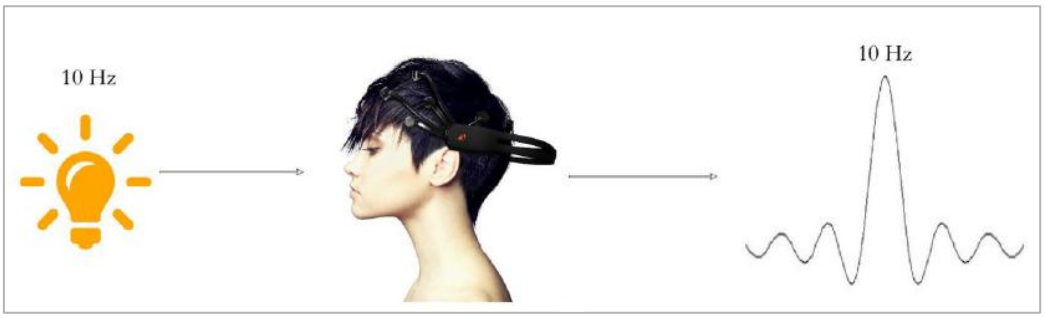

Figura 3. Generación de un estímulo basado en SSVEP [10]

En la figura 3 se muestra un esquema de la generación de un estímulo SSVEP. Cuando un sujeto mira una luz parpadeante a una frecuencia específica, en su corteza visual se produce una oscilación a esa misma frecuencia que se puede detectar en un registro de EEG. Este fenómeno se puede utilizar para determinar la intención del sujeto cuando mira a una fuente de luz específica que codifica un comando.

Los VEP se pueden clasificar en VEP transitorios (TVEPs) y de estado estacionario, "steady-state" (SSVEPs) [34]. Se considera TVEP cuando la frecuencia de estimulación visual es menor de 6 Hz, mientras que para un rango mayor de frecuencias produce un SSVEP. Los SSVEPs corresponden a una actividad neuronal oscilatoria que tiene la misma frecuencia del estímulo. Los sistemas BCI basados en SSVEP permiten al usuario seleccionar un objetivo utilizando la mirada. La estimulación visual consiste en varias fuentes de luz parpadeantes a distintas frecuencias, cada una de las cuales representa un comando. Sólo con fijar la mirada en el objetivo el BCI puede identificar mediante un análisis de características espectrales del SSVEP, y esto lo convierte en una de las modalidades más prometedoras para un sistema BCI no invasivo. Hay que tener en cuenta que en un BCI basado en SSVEP se utilizan varios estímulos que parpadean simultáneamente codificando las opciones de la interfaz. Aquél estímulo en el que el sujeto fije su atención visual será el que genere un SSVEP más reconocible. Este fenómeno se puede utilizar para decodificar el comando que el sujeto quiere realizar.

A la hora de seleccionar las frecuencias de estimulación en un BCI basado en SSVEPs, hay que tener en cuenta que la amplitud será mayor cuanto menor sea la frecuencia [5]. Por tanto en la zona de bajas frecuencias del SSVEP, que comprende las frecuencias entre 4 y 12 Hz, será más fácil detectar la instrucción. Este rango de frecuencias, por otra parte, tiene el inconveniente de que puede

causar molestias y fatiga en el sujeto. Frecuencias más altas reducen la sensación de fatiga pero son más difíciles de identificar.

A la hora de diseñar este tipo de dispositivos, es importante tener en cuenta que la respuesta de cada sujeto al mismo estímulo es diferente y difícil de predecir [35], [36]. Aunque en la mayor parte de los BCIs basados en SSVEP se utilizan frecuencias fijas, el Grupo de Neurocomputación Biológica de la EPS UAM ha demostrado que puede ser mucho más eficaz seleccionar frecuencias específicas para cada sujeto [37]. Es por esto que resultaría útil diseñar un dispositivo de estimulación que sea capaz de admitir una configuración de frecuencias variables y adaptables a cada individuo que utiliza el BCI.

Para la elaboración de este Trabajo, se utilizará la técnica de generación de estímulos SSVEP a partir de un hardware externo, que ha demostrado generar mejores resultados que una estimulación por pantalla [2]. Para ello, se hará uso de un total de seis LEDs que tendrán diferentes instrucciones asociadas. Estas instrucciones se configurarán a través de una interfaz gráfica y permitirán la realización de diferentes actividades en un sistema de comunicación asistido. En el siguiente apartado se explicará el sistema seleccionado para la configuración de estos LEDs para obtener una respuesta que pueda ser interpretada, y el sistema de comunicación con un software que permita que esta interpretación se realice en tiempo real.

CAPÍTULO 3. DISEÑO Y DESARROLLO

A lo largo de este capítulo se estudiarán las distintas actividades y estudios que se han llevado a cabo desde el diseño hasta el desarrollo y puesta en producción para la creación de esta interfaz.

Una vez se han estudiado las características y funcionalidades de una interfaz cerebro – máquina y de cada uno de los elementos que la componen, es el momento de integrar el conocimiento existente con las metas perseguidas para este trabajo.

Como primera aproximación, se establecerán los **objetivos que se esperan alcanzar** con la implementación de esta Interfaz Cerebro – Máquina, las **funcionalidades** con las que contará y sus **características** principales.

Habrá que analizar entonces las **diferentes alternativas** que se contemplan para la consecución de este BCI que funcionará a tiempo real, la selección de la interfaz gráfica que se utilizará así como el sistema de adquisición de señal, métodos de estimulación y componentes hardware, justificando así la selección de cada elemento.

Con los elementos seleccionados, comenzará la implementación con el **desarrollo de la plataforma hardware** necesaria para el funcionamiento del sistema de estimulación del BCI basado en SSVEP. El siguiente paso será la **integración entre el hardware y la interfaz gráfica**, de manera que se pueda controlar la estimulación y la selección de frecuencias desde la misma aplicación en la que se realizará la adquisición de señal.

Tras verificar que la conexión entre la plataforma hardware y la interfaz gráfica de usuario es posible y además se realiza en tiempo real, es el momento de realizar el **diseño completo de la aplicación** que actuará como interfaz cerebro – máquina, implementando una interfaz flexible con diferentes funcionalidades.

Con la aplicación realizada, **se integrará el sistema de adquisición EEG** para recibir, acondicionar y procesar las señales de actividad cerebral, para su posterior interpretación y utilización.

3.1 Planteamiento Inicial y Diseño de la Interfaz

Las interfaces cerebro – máquina se definen como "sistema de comunicaciones hardware y software que permiten controlar un ordenador o dispositivo externo únicamente mediante la actividad cerebral [1]".

Con este trabajo se pretende no solamente la creación de una interfaz capaz de controlar un ordenador o dispositivo mediante la actividad cerebral, sino que además sirva como medio de comunicación efectivo para personas que de otra manera no tuvieran forma de comunicarse. Cabe por tanto dedicar un tiempo en planificar de qué manera se va a ayudar con este dispositivo a facilitar la comunicación y, en definitiva, el día a día de estas personas que así lo necesiten.

Se parte de la premisa de que este BCI ha de funcionar en **tiempo real**, siendo **flexible** en lo que respecta a las aplicaciones (que las opciones disponibles puedan adaptarse a las necesidades del usuario que lo vaya a utilizar) y sistemas de estimulación (vía software con estimulación por pantalla o hardware con iluminación externa), así como a métodos de adquisición (casco Emotiv u otros dispositivos de registro de EEG).

Otro de los objetivos que se va a perseguir durante toda la elaboración de este trabajo es la **sencillez** y **facilidad de uso,** procurando que todos los elementos que conforman la Interfaz Cerebro – Máquina se encuentren integrados en una misma aplicación de forma compacta. Se pretende elaborar un BCI que funcione simplemente **ejecutando un único programa,** y que sea ese programa el que se encargue de comunicarse por un lado con el sistema generador de estímulos basados en SSVEP, por otro con el dispositivo de adquisición de señales EEG y que realice el procesamiento de las señales e interprete los resultados que se vayan obteniendo.

Debido precisamente a esta sencillez que se pretende conseguir, se ha decidido implementar una **Interfaz Gráfica de Usuario** que muestre un menú con un total de seis opciones, cada una de estas opciones ligada a un elemento estimulador basado en SSVEP.

A continuación se detallarán las pantallas que esta interfaz gráfica debería mostrar para obtener un BCI con funcionalidad completa. Estos desplegables y las actividades planteadas han sido elegidos con la ayuda del **Centro Cottolengo de Madrid para personas discapacitadas.**

- Paso 1: Bienvenida.
    - Mostrará un mensaje de bienvenida, y pedirá al usuario que se identifique.
    - Si se trata de un nuevo usuario, se le asignará un número de usuario y se guardarán en un fichero de registro de usuarios tanto el número como nombre.
    - Si el usuario ya ha utilizado el programa con anterioridad, bastará con introducir su nombre y se comprobará si está registrado.
    - Una vez creado o verificado el usuario, se procederá a grabar el baseline de la señal que se utilizará para facilitar la detección del SSVEP asociado al estímulo.

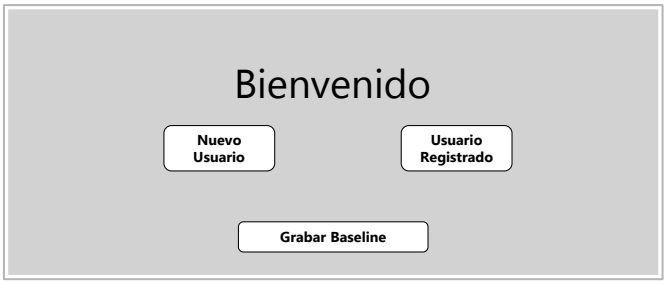

Figura 4. Menú Bienvenida BCI

- Paso 2: Grabación del Baseline
  - Tanto si se trata de un usuario nuevo como de uno registrado, será necesario realizar una grabación de 60 segundos de duración para almacenar los datos de la actividad cerebral durante ese periodo. Esto servirá para eliminar el ruido procedente de la actividad cerebral y de otros elementos del campo visual que puedan afectar a los futuros registros (similar al de una segmentación frente – fondo).
  - Durante este tiempo, el monitor permanecerá de color negro. Al transcurrir los 60 segundos, se indicará al usuario que puede continuar avanzando en el programa.
- Paso 3: Algoritmo de selección de mejores frecuencias de estimulación para cada usuario.
  - Si se trata de un usuario registrado, este paso no será necesario ya que previamente se habrán encontrado sus mejores frecuencias y almacenado en el fichero de registro.
  - En caso contrario, se aplicará un algoritmo que seleccione las mejores frecuencias correspondientes al sujeto.
- Paso 4: Menú de Inicio del BCI.
  - Una vez configurado, se podrá empezar a utilizar la interfaz.
  - Se mostrarán un total de seis opciones, cada una asociada a un estímulo basado en SSVEP oscilando a una frecuencia. Bastará con que el usuario concentre su atención en el estímulo para seleccionar la opción que desee.

Figura 5. Menú de Inicio del BCI

- Juegos
    - Se compondrá de una serie de ejercicios orientados a comprobar la efectividad del BCI, midiendo los tiempos de respuesta y número de aciertos cuando se solicita al usuario mirar a un estímulo en concreto.
- Speller
    - Facilitado para habilitar la comunicación a través de este BCI, siendo éste uno de los objetivos principales de este trabajo.
    - Seleccionando esta opción, se mostrarán las letras del abecedario agrupadas en cinco bloques (la última opción siempre se utilizará para volver hacia atrás). Al seleccionar uno de los bloques, se desglosará de nuevo pudiendo seleccionar una letra en concreto.
- Sí / No
    - Se trata en este caso de otro medio de comunicación, cuyas únicas opciones serán "Sí", "No" o "Siguiente Pregunta".
    - El objetivo será facilitar la comunicación entre el sujeto que utilice el BCI y otra persona encargada de realizar preguntas de manera oral, logrando de esta manera una comunicación efectiva y ágil.
- Easy Life
    - Esta opción se ha realizado con la ayuda del Centro Cottolengo de Madrid para personas discapacitadas, y recoge una serie de actividades básicas que facilitan la vida de los pacientes que allí se encuentran. Entre las opciones aparecen actividades, acciones o sentimientos, pudiendo modificar las opciones disponibles en función de las necesidades del sujeto.
- Re – Calibrar
    - Vuelve a realizar el algoritmo de selección de mejores frecuencias (paso 3).
- Cerrar Sesión
    - Cierra la sesión actual del usuario y vuelve al paso 1.

Una vez se ha dejado planteado **qué** se quiere hacer, se procede a explicar **cómo** se va a hacer: qué elementos se van a utilizar y cómo se va a implementar este diseño.

3.2 Selección de los elementos BCI

Tal y como se ha visto a lo largo del capítulo **"2.1 BCI: Interfaz Cerebro – Máquina",** una interfaz cerebro máquina se compone de cinco etapas:

- Adquisición de la señal
- Preprocesado
- Extracción de las características
- Interpretación y clasificación
- Feedback hacia el software.

Para cada una de estas etapas, es necesario utilizar diferentes elementos que permitan la realización óptima de cada una de las actividades a llevar a cabo. Así pues, para la etapa de adquisición de señal serán necesarios dos dispositivos diferenciados:

- Por un lado, habrá que realizar un **sistema de estimulación visual basado en SSVEP** que emita diferentes estímulos a diferentes frecuencias
- Por otro, se utilizará un **sistema de adquisición de señales electroencefalográficas** capaz de registrar la actividad cerebral de los sujetos

Una vez adquirida la señal, será necesario una **aplicación** capaz de realizar el preprocesado de la señal y la extracción de características para su correcta interpretación y clasificación.

A continuación se realizará un análisis de las diferentes alternativas para cada uno de ellos.

3.2.1 Sistema de Estimulación Visual

Como se ha descrito en el capítulo **"2.3 SSVEP: Steady – State Visually Evoked Potentials",** los estímulos visuales SSVEP provocan en el córtex cerebral una oscilación con la misma frecuencia que la que se está emitiendo en el dispositivo generador de señales.

Cuando el pulso generado oscila a una frecuencia precisa y con una amplitud adecuada, al realizar el procesado de la señal y aplicar una transformada de Fourier aparecerá un pico en el espectro coincidente con la frecuencia del estímulo generado.

Para ello, existen dos maneras diferentes de implementar el sistema de estimulación visual basado en SSVEP, bien mediante un **software** que genere el estímulo parpadeante **a través de un monitor**, o de **manera externa** con un hardware controlador de LEDs independiente del sistema de procesado de señal.

3.2.1.1 Implementación a través de un monitor

Haciendo uso de la pantalla de un ordenador se conseguiría un sistema por pantalla con un número de estímulos configurable, parpadeando cada uno a una frecuencia distinta.

La gran ventaja de que este sistema no se implementase a través de un hardware es que no estaría limitado por el número de elementos y parámetros fijos. **Mediante diferentes parámetros se podría configurar** la distancia que los separa, el brillo, el número de estímulos, los colores, etc… y adaptar dichos parámetros a cada sujeto.

Sin embargo, el problema de los SSVEP cuando se muestran por pantalla [2] es que la frecuencia de oscilación está limitada por la tasa de refresco de la pantalla LCD o del tubo de rayos catódicos del monitor [38], [39]. Otro problema a tener en cuenta es que sólo un pequeño número de estímulos a una determinada frecuencia provocan una señal SSVEP intensa, lo que hace que el rango de frecuencias disponibles en SSVEP sea limitado, detectándose respuestas entre los 4Hz y los 80 o 90Hz [5].

Una pantalla tradicional suele tener una tasa de refresco de 75 Hz, lo que deja 75 frecuencias de estimulación disponibles, una por cada submúltiplo de 75 (entero y decimal). De estas frecuencias, sólo 18 son mayores que 4Hz, y por debajo de 4 Hz no se detectan los estímulos ni se consideran SSVEP. Si además se tiene en cuenta que a estas frecuencias se les tiene que poder aplicar una FFT para su detección, las frecuencias óptimas disponibles se reducen a 7. Si se añade que hay frecuencias que son armónicos y subarmónicos de otras frecuencias, hay que descartar esas frecuencias por no poder distinguir cuál es la frecuencia y cuál el armónico. Esto deja con 4 frecuencias restantes, que son las disponibles para generar los estímulos que el BCI basado en SSVEP ha de detectar [37].

Esto se puede solucionar en parte con el protocolo MFSC (Multiple Frequencies Sequential Coding), que consigue codificar un objetivo utilizando múltiples frecuencias en secuencia durante cada ciclo [38]. Así se consigue implementar muchos más objetivos con las limitadas frecuencias disponibles, en comparación con los códigos tradicionales de codificación de frecuencias.

3.2.1.2 Implementación mediante un hardware externo

Con esta alternativa se busca eliminar la dependencia de un monitor limitado por una tasa de refresco utilizando para ello elementos externos a un ordenador (LEDs), que han de ser capaces a su vez de comunicarse con el software controlador de todo el sistema para establecer los estímulos a las frecuencias seleccionadas.

El inconveniente que presenta esta solución es que una vez establecido el diseño, resulta más complejo variar los parámetros de configuración de diseño. Una vez elegidos los componentes, el número de estímulos, su color o la distancia entre los mismos serán fijos y no se podrán adaptar a cada sujeto. Por otro lado, una arquitectura basada en LEDs no se puede recombinar de forma flexible en iconos o disposiciones espaciales que favorezcan la comprensión de la codificación del estímulo por parte del usuario.

Por contraparte, la ventaja principal de este sistema es que elimina la dependencia con una tasa de refresco, permitiendo utilizar un rango de frecuencias considerablemente más elevado, consiguiendo resultados positivos desde 6 Hz hasta incluso frecuencias por encima de 40 Hz.

La ventaja de añadir a un sistema BCI basado en SSVEPs evocados con LEDs de información adicional presentada en un monitor, es combinar la flexibilidad de la información de la segunda opción con la precisión de la estimuación en frecuencias de la primera.

Es por esta razón por la que se ha decidido implementar el sistema de estimulación visual a partir de seis LEDs y un microcontrolador Arduino Uno con capacidad para establecer una comunicación con el software que captura y analiza la actividad cerebral de los sujetos, a la vez que presenta información adicional en la pantalla de un ordenador

3.2.2 Sistema de adquisición de señal electroencefalográfica

El capítulo "2.1 BCI: Interfaz Cerebro – MáquinaEEG: Obtención de la Señal" del estado del arte narraba los diferentes sistemas de obtención de señal, ya fuesen endógenos, exógenos, de registro de actividad eléctrica o fisiológica.

Para el sistema de adquisición de señal se ha optado por una aproximación no invasiva ya que otras opciones, a pesar de proporcionar una mejor calidad, hacen necesario el uso de cirugía para implantar electrodos en las regiones occipital y parietal del cerebro. Por otro lado, siendo la **electroencefalografía** la que mejor aproximación ofrece para los objetivos de este proyecto, existen diferentes alternativas para realizar la toma de datos.

Algunos de los cascos que se utilizan implican un montaje complejo, lo que supone un alto nivel de incomodidad para el sujeto, debido a la laboriosidad de colocación de los electrodos y su sujeción. Se buscará por tanto un dispositivo cuyo montaje sea lo más rápido posible, con gran estabilidad para su utilización en procesos de larga duración y cuya calidad permita la detección de estímulos basados en SSVEP.

El sistema de registro de EEG que se va a utilizar en este BCI es el EPOC, diseñado por EMOTIV, un dispositivo multicanal comercial de relativo bajo coste y fácil colocación.

Este sistema ofrece una resolución aceptable, y se conecta de manera inalámbrica a un ordenador que puede funcionar tanto en Windows como iOS o Linux [10]. Se trata de un sistema unipolar que utiliza un set de 14 sensores y dos referencias, distribuidos según el sistema Internacional 10-20, estandarizado por la AES (American Electroencephalographic Society). Este sistema estipula que a partir de dos puntos de referencia los electrodos se dispongan en planos que se abran a intervalos de 10% y 20% desde el plano que une las dos referencias. Esta distribución se puede observar en la figura 6

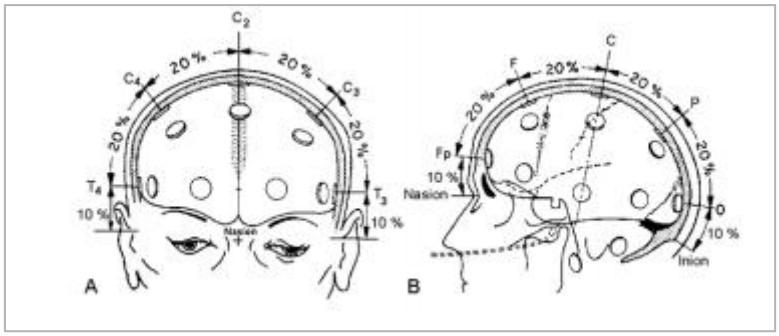

Figura 6. Disposición de los electrodos según el sistema internacional 10-20

Los electrodos de EMOTIV EPOC se colocan en las posiciones AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4, donde F significa Frontal, A indica el lóbulo de la oreja, C la región central, P el parietal, T el temporal y O el occipital. Cabe destacar que en principio estas posciones no son las ideales para registrar SSVEPs, ya que este tipo de señales se detectan mejor con electrodos colocados en regiones occipitales y este caso solo posiciona dos electrodos en esa zona.

Cada uno de estos electrodos corresponde a un canal por el que se registra la actividad cerebral en ese punto. El software de Emotiv que muestra por pantalla las señales, permitiendo guardar los datos que registra, que serán los analizados posteriormente para extraer las características específicas que traducen la actividad cerebral en intenciones que emite el sujeto. En la figura 7 se observa un ejemplo de uso de la aplicación TestBench de Emotiv que muestra el status de los electrodos, indicándose con una luz verde si la calidad del contacto del electrodo es óptima y con una luz amarilla, naranja, rojo o negro si no lo es con esa gradación, así como las diferentes señales que registra cada electrodo. Esta herramienta se utiliza para registrar el EEG durante las pruebas de validación del dispositivo diseñado.

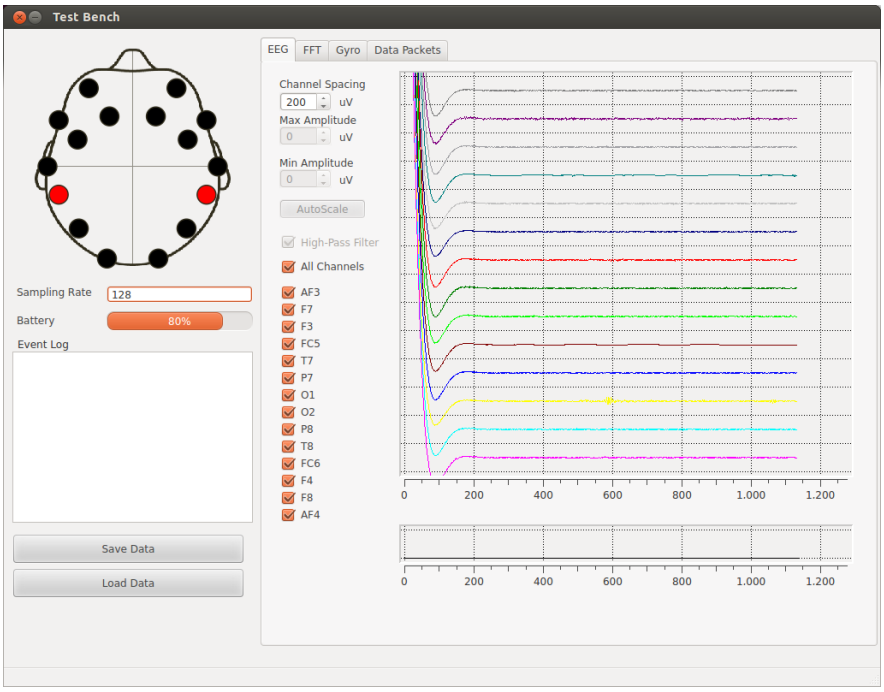

Figura 7. Captura de pantalla del Test Bench de Emotiv

3.2.3 Software de procesado de señal, control de LEDs y presentación de información por pantalla

Con respecto al software, será necesario tener en cuenta las siguientes consideraciones:

- El software controlador del proceso ha de ser capaz de trabajar en tiempo real para habilitar la interfaz cerebro – máquina como sistema de comunicaciones.

- También deberá permitir realizar dos tareas de manera simultánea: por un lado, deberá leer y almacenar la información procedente del sistema de adquisición de electroencefalogramas Emotiv Epoc, y por el otro deberá analizar concurrentemente esta información mediante ventanas temporales que permitan traducir estas señales en instrucciones.

- Deberá contar con un mecanismo de comunicación que permita configurar el hardware externo de estimulación.

- Tiene que contar con un entorno con capacidad de realizar operaciones matemáticas y transformadas discretas de Fourier para el correcto análisis del espectro de la señal, utilizando el algoritmo FFT (Fast Fourier Transform) y diferentes técnicas de enventanado y filtrado.

- El sistema tiene que gestionar de forma coordinada la presentación de información por pantalla, la gestión del control de los LEDs de estimulación y el procesamiento de señales que dan lugar a la interpretación de los comandos que el usuario quiere realizar, y la ejecución de las correspondientes acciones por parte de la interfaz.

Con estas funcionalidades, se podrían haber seleccionado herramientas open-source como EEGLAB [40], OpenVIBE [41] u OpenBCI [42]. Sin embargo, tras un análisis de las mismas se llegó a la conclusión de que no son apropiadas ya que no cumplen todos los requisitos especificados anteriormente, y no están diseñadas para soportar la adaptación simultánea con un hardware controlador externo.

Teniendo en cuenta estas consideraciones se ha optado por utilizar el entorno gráfico Qt con lenguaje de programación orientada a objetos, que es capaz de habilitar la comunicación con diferentes elementos externos a la vez que permite el diseño de una interfaz gráfica que permita crear una interfaz cerebro – máquina con señalización mixta.

3.3 Desarrollo de la plataforma Hardware

Como se ha visto en la selección del sistema de estimulación visual, el dispositivo a utilizar como generador de frecuencias será una placa **Arduino Uno** y un total de **seis LEDs** de color blanco con sus respectivos difusores de diferentes colores.

Los LEDs utilizados son un modelo LW500AM [43] con las siguientes características:

- Alta luminosidad con emisión de color blanca
- Empaquetado de 5mm de diámetro
- Ángulo de visión de 100º
- Coordenadas de color: x=0,31, y = 0,31
- Voltaje típico de 3,3 V con tolerancia de ±0,05 V

Se ha seleccionado el número de estímulos considerando la funcionalidad de la Interfaz Gráfica, que se detallará más adelante en el capítulo "3.4 Diseño de la Interfaz Gráfica de Usuario", ya que con un total de seis estímulos se cubre toda la funcionalidad que esta aplicación necesita.

Para la configuración de los LEDs se ha utilizado el software que Arduino proporciona para su programación, configurando de manera independiente cada uno de ellos para establecer las frecuencias de estimulación, además de contar con un parámetro adicional que permite configurar el brillo de los mismos.

Se comenzará seleccionando los pines que se van a utilizar, escogiendo para tal fin aquellos con funcionalidad PWM configurados como señales de salida.

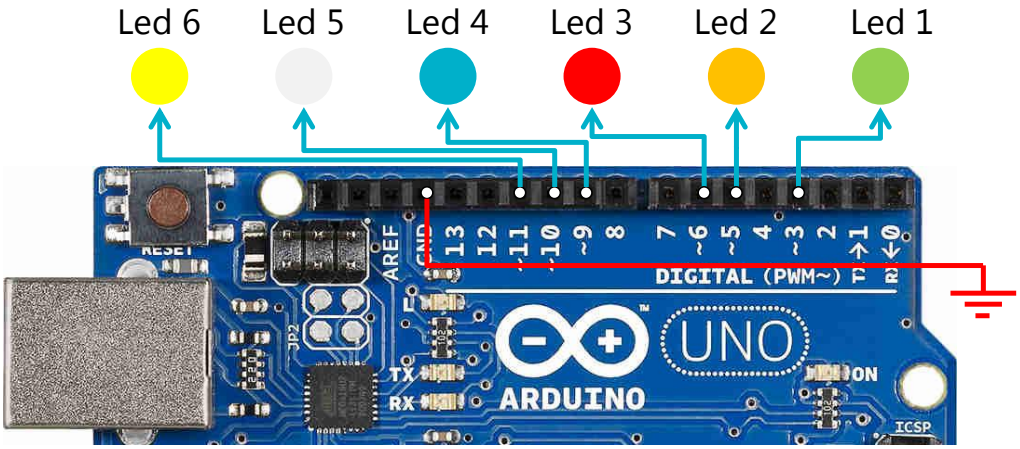

Figura 8. Conexiones PWM entre Arduino Uno y el sistema de estimulación visual [8]

Se muestra a continuación el código necesario para configurar el pin que conecta el LED 1 al microprocesador arduino:

```
const int led1Pin =  3; //El LED 1 se conectará al pin 3
int led1State = LOW;  // Estado inicial de los LEDs a LOW
void setup() {

  pinMode(led1Pin, OUTPUT); // Se declaran los pines como salidas
  analogWrite(led1Pin, led1State); //Configuración inicial a LOW
}
```

La variable led1State podrá tomar valores de entre 0 a 255, siendo 0 la configuración en la que el LED se encuentra apagado y 255 en la que brilla con su máxima intensidad.

Para realizar la configuración de las frecuencias, y debido a que Arduino no tiene soporte multihilo, de manera secuencial se comprueba si el periodo de cada uno de los seis estímulos ha sido alcanzado, cambiando de estado en caso positivo y permaneciendo en su configuración establecida en caso contrario. Para ello es necesario disponer de señales de PWM para habilitar la configuración de la intensidad de brillo de los estímulos, así como para emitir un pulso cuadrado.

Una vez se recibe la frecuencia a configurar en el LED, se calcula el periodo y se inicializa una variable encargada de medir el tiempo transcurrido desde la última vez que el LED cambió de estado. Cuando se alcanza ese valor de periodo, la configuración del LED cambia y los contadores vuelven a comenzar:

```
int MAX_BRILLO = 255;
void EncenderLed1(unsigned long interval1)
{
  //Obtenemos para cada pasada de bucle el tiempo actual en microsegundos
  currentMicros1 = micros();

   // Si la diferencia entre el tiempo actual (currentMicros) y la ultima vez
que el LED parpadeo (previousMicros) es mayor que el periodo de oscilacion, el
LED cambia de estado (Si esta a HIGH pasa a LOW y viceversa).

  //Este procedimiento se repite para los 6 LEDS

     if (interval1==0)
        digitalWrite(led1Pin, LOW);

     else if(currentMicros1 - previousMicros1 >= interval1)
     {
          //Guardo el microsegundo en el que cambio el estado
          previousMicros1 = currentMicros1;

          if (led1State == LOW)
              led1State = MAX_BRILLO;
          else
              led1State = LOW;

          // Paso al pin el valor del estado
          analogWrite(led1Pin, led1State);

      }
```

Una vez se ha realizado la configuración de los pines y de las frecuencias, se realizarán las pruebas para comprobar la precisión del circuito, cuyos resultados se muestran en el capítulo "4.1 Validación de la configuración de las frecuencias".

Una vez se ha ajustado y demostrado que la configuración en los pines es la que realmente se obtiene en los LEDs, la siguiente tarea será establecer los parámetros de distancia, color y brillo que mejor resultados generan en la interfaz. De nuevo, tras las pruebas pertinentes ("4.2 Validación del dispositivo de estimulación a través de un EEG"), el diseño con el que se han conseguido resultados notables ha sido aquel que maximiza el espacio disponible: para el monitor que se utilizará, de dimensiones 60 x 35 cm, se colocarán los LEDs en dos filas, situando tres en el marco superior del monitor y tres en el marco inferior. Se colocarán cuatro LEDs en los extremos del monitor, y dos en el punto medio horizontal entre ambos.

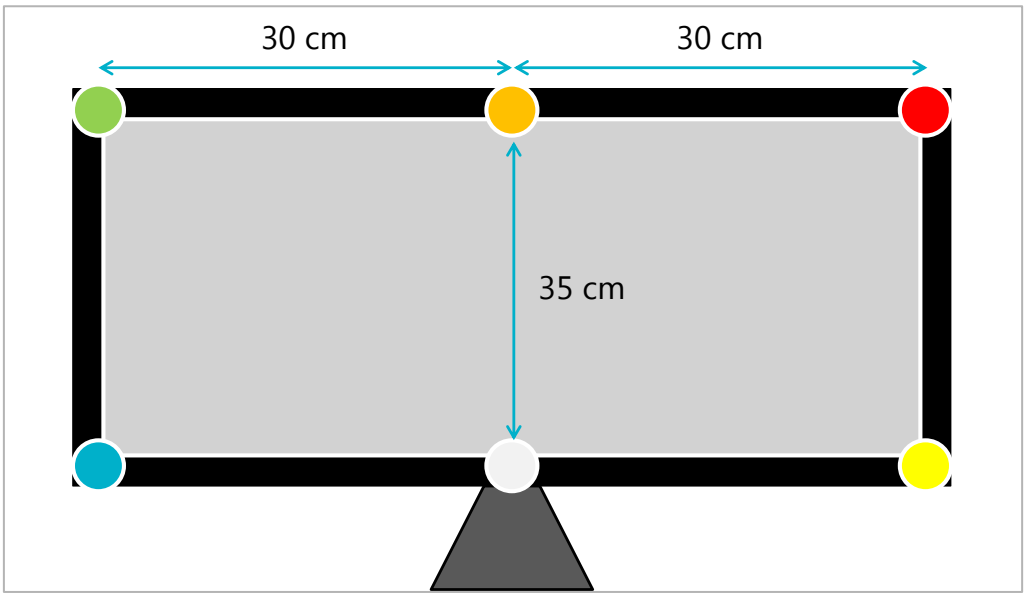

Figura 9. Dimensiones de la configuración de los LEDs

Estos LEDs, que actuarán como dispositivos de estimulación SSVEP, se conectarán a una placa protoboard donde se acondicionarán las señales en caso de ser necesario mediante resistencias, y se conectarán a los pines de arduino habilitados para tal fin.

3.3.1 Integración con QT

Una vez se ha diseñado y comprobado que funciona el dispositivo hardware de estimulación visual, será necesario integrarlo con la aplicación gráfica de usuario que será diseñada en QT. En este apartado se estudiará el acondicionamiento de Arduino para poder ser integrado en dicha aplicación, habilitar la comunicación entre los mismos y poder integrar los diferentes elementos de esta Interfaz Cerebro – Máquina.

La comunicación entre el BCI y los dispositivos de estimulación basados en SSVEP es fundamental debido a varios aspectos:

- Al inicio de la aplicación, la interfaz registrará un **baseline** de 60 segundos de duración. Se trata de un registro de la actividad cerebral sin ningún tipo de estímulo

con el objetivo de eliminar distorsiones en los posteriores registros y, por tanto, los LEDs han de permanecer apagados durante el tiempo que dure este registro. La aplicación deberá comunicarse en este caso con la plataforma Arduino para indicar que se trata de un registro de baseline y que los LEDs han de permanecer apagados.

- Tras este registro de baseline, y en caso de que se trate de un nuevo usuario, será necesario realizar un barrido de frecuencias que seleccione aquellas que mejor se ajusten a cada sujeto y cuyos resultados optimicen el funcionamiento de la interfaz. Este barrido en frecuencias se realizará en tres etapas:

  - Fase 1: Seleccionando uno de las seis luces LED cada vez, se mostrará al usuario durante 20 segundos una frecuencia y se calculará una puntuación para la misma. Este ejercicio se repetirá para todas las frecuencias de entre 6 a 23 Hz.

  - Fase 2: Una vez obtenida la puntuación de todo el rango de frecuencias, se seleccionarán aquellas en las que se ha obtenido una diferencia de amplitud entre picos mayor que un umbral establecido. Se procederá en este caso a realizar otro barrido utilizando pares de frecuencias de manera simultánea, calculando así el score de cada frecuencia teniendo en cuenta las interferencias que produce otra frecuencia oscilando en el campo visual del sujeto.

  - Fase 3: Por último, y en función de las puntuaciones obtenidas en la fase 2, se seleccionarán las 6 frecuencias con las que se ha obtenido un resultado aceptable y se realizará una última comprobación para estudiar la validez de las frecuencias dada la interferencia [44].

  - Existe también una versión simplificada en la que se realiza el barrido de frecuencias de seis en seis, oscilando simultáneamente cada una en los LEDs, y calculando las mejores frecuencias en función de la precisión, amplitud y distancia con respecto a los máximos secundarios. Será esta opción la que se utilizará en las pruebas, de cara a agilizar las mismas.

- Configuradas las seis mejores frecuencias del sujeto, la Interfaz Cerebro – Máquina comenzará su funcionamiento normal, manteniendo las frecuencias con precisión durante toda la sesión.

Por tanto, la comunicación entre el microprocesador Arduino Uno y la Aplicación Gráfica de Usuario desarrollada en QT ha de ser fluida y en tiempo real.

Una opción contemplada para realizar esta comunicación fue a través de ficheros de texto donde la GUI en QT pudiera escribir las frecuencias a configurar y el microprocesador leyese de los mismos, tal y como se hizo durante el Trabajo de Fin de Grado "Diseño de Interfaces Cerebro – Máquina y Hombre – Máquina controlados por señalización biológica" [2].

Sin embargo, mientras que en ese trabajo previo se utilizó un microprocesador Arduino Yen por razones de portabilidad, y teniendo el mismo soporte para tarjeta micro SD y lectura de ficheros, la placa Arduino Uno no dispone de dicha característica, descartando la opción de la comunicación mediante ficheros de texto y haciendo necesaria alguna alternativa.

El método final a utilizar debido a su compatibilidad con los dos programas y fidelidad a su utilización en tiempo real ha sido la **habilitación del puerto serie** para la comunicación e integración de la Interfaz Cerebro – Máquina. En este apartado se describirá la implementación de esta comunicación en el lado del microprocesador, mientras que en el capítulo "3.4.2 Integración con Arduino" del capítulo "0

Diseño de la Interfaz Gráfica de Usuario" se comentarán los pasos que se han seguido para la configuración desde QT.

Atendiendo a las necesidades de comunicación y las diferentes configuraciones que se pueden establecer (baseline, selección de frecuencias, ejecución del BCI), será necesario que en el microprocesador se reciban no solamente las frecuencias a establecer, sino también el estado en el que se encuentra la interfaz.

Por todo esto, en una primera aproximación se decidió un envío a través del puerto serie con un mensaje similar al siguiente:

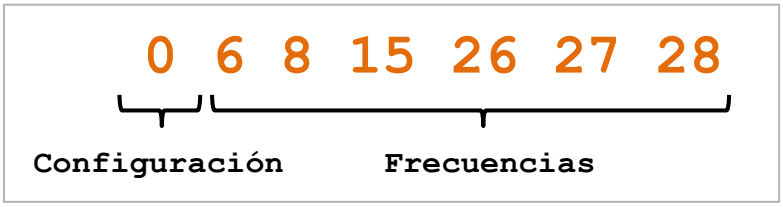

Figura 10. Mensaje recibido en el microprocesador Arduino Uno

El primer dato a recibir será el parámetro **config_led,** que tomará el valor binario 0 o 1. En el programa Arduino, se permanecerá en el bucle de inicio `void setup()` mientras que config_led tome el valor 0, que indica que las frecuencias que se están recibiendo no son las definitivas, sino que se trata del baseline o del algoritmo de selección de mejores frecuencias.

Durante este setup, Arduino leerá los valores recibidos por el puerto serie y cada vez que encuentre que config_led es 0, configurará las frecuencias en los LEDs durante **20 segundos.** Pasados estos 20 segundos, volverá a leer del puerto serie y a realizar los mismos pasos, **hasta que reciba un valor de config_led igual a 1.**

```
while (config_led == 0){

. . .

        contador=0;
        while(contador <=200000) {

        EncenderLeds(interval1, interval2, interval3, interval4, interval5,
interval6);
      contador=contador+1;
. . .
}
```

Es necesario realizar la comprobación de tiempo mediante un contador debido a que cualquier función que pause el programa entrará en conflicto con la configuración de los LEDs.

Una vez se reciba `config_led = 1`, Arduino pasará a la rutina de bucle indefinido, estableciendo los valores de frecuencia finales para permitir la ejecución del BCI indefinidamente. A continuación se muestran algunos ejemplos de configuración:

- **0** 0 0 0 0 0 0: Grabación del baseline. Las seis frecuencias configuradas a 0 (LEDs apagados).

- **0** 6 0 0 0 0 0: Algoritmo de selección de frecuencias, fase 1: Se ilumina únicamente el LED 1 a una frecuencia de 6 Hz.

- **0** 6 8 15 26 27 28: Algoritmo de selección de frecuencias, fase 3: Se iluminan todos los LEDs para verificar la interferencia entre los mismos en el registro de actividad cerebral.

- **1** 6 8 15 26 27 28: Ejecución indefinida del BCI con frecuencias fijas 6, 8, 15, 26, 27, 28.

Se observó que con esta configuración el programa hardware respondía satisfactoriamente cuando se enviaban los mensajes desde la terminal serie facilitada por el programa de Arduino.

Sin embargo, tras la implementación de la terminal que actuará como puerto serie en Qt (ver capítulo "3.4.2 Integración con Arduino") se observó que solamente se recibían bien los mensajes la primera vez que se enviaban, y que en envíos posteriores Arduino no respondía.

Esto se debe a que desde Qt los mensajes se envían en bloques de 8 bytes, y tras este primer envío queda todavía información en los últimos bytes que se leen pasados los 10 segundos. Como esta información restante no concuerda con la que Arduino espera recibir, el puerto serie queda bloqueado.

Adicionalmente a este problema, QT envía paquetes de seguimiento para mantener el puerto serie activado y a la escucha, y estos paquetes son también recibidos por Arduino y malinterpretados.

Para solucionar este problema, se ha tomado la decisión de enviar un **mensaje de cabecera** al paquete que Arduino debe recibir como información. Cuando se recibe información a través del puerto serie, se procesa hasta que se detecta el mensaje de cabecera preestablecido para ambos programas. Esto elimina cualquier paquete que no contiene información y permite una comunicación efectiva y en tiempo real entre el hardware y el software de esta Interfaz Cerebro – Máquina.

Se ha desarrollado por tanto un sistema de estimulación efectivo y preciso, capaz de configurar seis LEDs que actuarán como dispositivos capaces de evocar potenciales SSVEP y que puede ser configurado mediante otro programa (en este caso QT) a través de la habilitación del puerto serie.

3.4 Diseño de la Interfaz Gráfica de Usuario

En este apartado se detallan los pasos que se han seguido para el desarrollo de la GUI que ya se describió en el apartado "3.1 Planteamiento Inicial y Diseño de la Interfaz". Es importante destacar que la interfaz gráfica contribuye a desambiguar los estímulos con los LEDs y redunda en una mejora del control del interfaz.

Para un entendimiento en profundidad de los pasos que se han seguido para la creación de este BCI, se comenzará con una breve introducción al entorno QT, destacando aquellas propiedades que resulten de interés para este desarrollo, así como algunas funcionalidades destacables que han hecho posible el desarrollo de este trabajo.

Como se ha visto en el apartado anterior, durante del desarrollo hardware, esta interfaz necesitará emitir los mensajes de configuración a través de un puerto serie, y por tanto se comenzará el diseño QT desde este hito, habilitando una terminal virtual capaz de enviar mensajes a Arduino.

Una vez se habilite esta comunicación, se construirá sobre este programa la GUI y el menú desplegable con las diferentes actividades que se pueden realizar en esta aplicación. Por último, se realizará la conexión entre esta GUI y el software de adquisición del casco Emotiv de EPOC, integrando así todas las etapas del proceso de implementación de una Interfaz Cerebro – Máquina.

3.4.1 Introducción a QT

Qt es una biblioteca multiplataforma ampliamente usada para **desarrollar aplicaciones con Interfaz Gráfica de Usuario,** así como para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores [12].

Qt utiliza el lenguaje de programación **C++** de forma nativa, y adicionalmente puede ser utilizado en otros lenguajes de programación a través de *bindings*.

Funciona en todas las principales plataformas, y tiene un amplio apoyo. El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales. Esta versatilidad y capacidad de integración es la que la convierte en idónea para desarrollar este BCI.

Todos los programas de QT constan de un fichero **".pro"** que ha de existir siempre, en el que se define el contenido del programa: cabeceras, ficheros, formas y librerías. A modo de ejemplo se muestra en la figura 11 el aspecto de este fichero una vez implementada la GUI, "terminal.pro". Se puede ver en la imagen las librerías adicionales que se van a utilizar en la línea 1 (QtWidgets y QtSerialPort). TARGET define el nombre de la aplicación, y TEMPLATE indica si lo que se va a programar es una aplicación, una librería o un subdirectorio. En este caso, será una aplicación.

Tras estos parámetros, se incluyen los ficheros de fuente (main.cpp y clases) y las cabeceras, así como las formas (ficheros .ui que definen la Interfaz Gráfica de Usuario) a utilizar durante el proyecto.

En el apartado "3.4.3 Implementación de la GUI" se explicará la funcionalidad de cada uno de los ficheros que se han incluido al programa desarrollado.

terminal.pro

```
1  QT += widgets serialport
2
3  TARGET = terminal
4  TEMPLATE = app
5
6  SOURCES += \
7      main.cpp \
8      mainwindow.cpp \
9      settingsdialog.cpp \
10     console.cpp \
11     mythread.cpp
12
13 HEADERS += \
14     mainwindow.h \
15     settingsdialog.h \
16     console.h \
17     mythread.h
18
19 FORMS += \
20     mainwindow.ui \
21     settingsdialog.ui
22
23 RESOURCES += \
24     terminal.qrc
```

Figura 11. Ejemplo de fichero: Terminal.pro

Aunque todos los programas desarrollados en QT constan de este fichero, existen diferentes tipologías que se pueden implementar:

- **QT Console Application:** Programa sin interfaz gráfica que consta de un *.pro* y de un único fichero *main.cpp*, y se utilizaría mediante la consola de comandos.

- **Qt Quick Application:** No tiene cabeceras, y consta de un *.pro*, un *main.cpp* y un fichero *.qml* que se utilizaría para implementar una GUI sencilla. Qt proporciona herramientas predefinidas que se pueden utilizar para tal fin.

- **Qt Widgets Application:** Crea una interfaz gráfica ".ui" que traduce de manera automática a .xml, constando además de cabeceras .h y ficheros .cpp que interactúan entre ellos para la ejecución de los programas. Es posible crear más de una GUI e ir llamándolas de manera recurrente a lo largo del programa.

Por sus características, para la elaboración de este trabajo se creará un proyecto de tipo Qt Widgets Application. Cada fichero adicional que se incluya será tratado como una clase, con su fichero de fuente y su cabecera, siendo utilizados desde el objeto principal mainwindow de tipo QMainWindow.

En esta tipología de proyectos, el fichero main.cpp será un programa sencillo que realiza únicamente dos acciones:

- En primer lugar, define una aplicación "a" de clase QApplication. Esta clase, la aplicación, recibe los parámetros del programa si los hubiera y llama al Event Loop: un bucle que espera a que cambien los inputs de la aplicación mediante **señales**. Estas señales pueden ser la pulsación de un botón, el cambio de una variable, pulsación de una tecla, etc. Por tanto, al ejecutar la aplicación predefinida "a", desde ese momento en adelante el programa funcionará mediante un mecanismo de emisión de señales a las que se conectan otros elementos del programa (**mecanismo SIGNALS & SLOTS**).

- En segundo lugar, se define una **ventana** w de clase MainWindow, definida posteriormente en la clase mainwindow (mainwindow.cpp y mainwindow.h). Esta ventana contendrá QWidgets, QObjects y demás clases y objetos que se integran para formar una GUI y que generan/se activan mediante señales.

main.cpp

```
1  /*********************************
34
35  #include <QApplication>
36
37  #include "mainwindow.h"
38
39  int main(int argc, char *argv[])
40  {
41      QApplication a(argc, argv);
42      MainWindow w;
43      w.show();
44      return a.exec();
45  }
```

Figura 12. Contenido del fichero principal del programa: main.cpp

Main.cpp finaliza mostrando la ventana que se ha creado, y ejecutando la aplicación que llama al Event Loop.

Qt utiliza un sistema jerárquico de objetos, con un sistema parental de padres e hijos (Parenting System). En la figura 12, que se muestra a continuación, se representa la jerarquía de clases de la que se compone Qt [45].

La clase más básica se denomina *QObject*, y la mayoría de las clases heredan de ella. Tiene capacidades de gestión de eventos, mecanismo de signals & slots y es el padre del Parenting System. Todo objeto que herede de QObject puede tener padre e hijos. Cuando un objeto se destruya, se destruirán con él todos los hijos que hereden de él. Al definir un nuevo objeto será necesario indicar quién es el padre del que hereda.

De ella hereda QWidget, entre otras, que será la clase seleccionada para las ventanas de esta aplicación BCI. QWidget también es capaz de utilizar los mecanismos de Signals & Slots, el Parenting System y puede responder ante eventos. Contiene propiedades que se pueden utilizar para describir la ventana: posición, tamaño, señales emitidas por el teclado o el ratón, etc. La mayoría de elementos gráficos heredarán de QWidget.

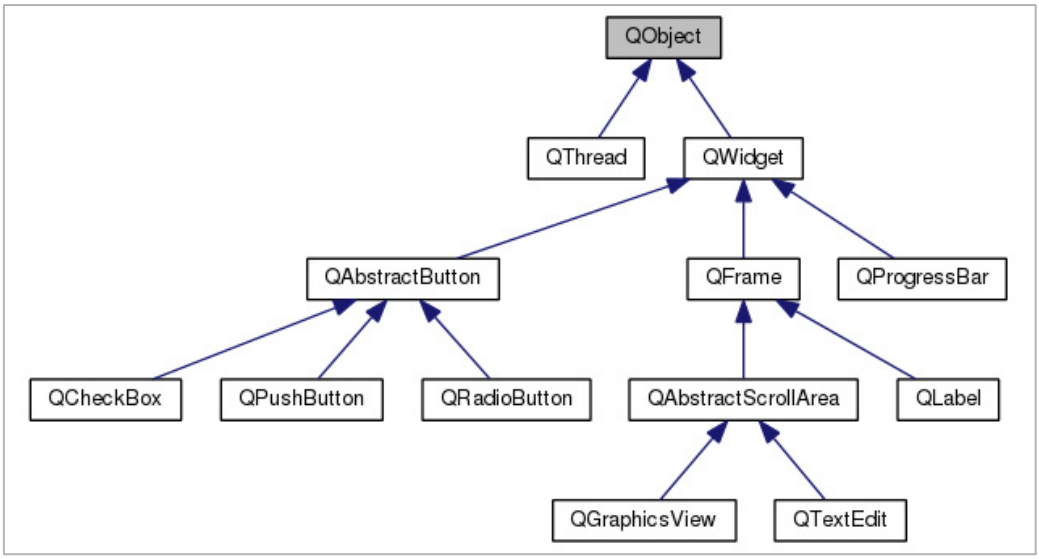

Figura 13. Sistema jerárquico de clases en Qt

De cara a simplificar el código, cada clase se definirá en ficheros diferentes. Por tanto, en cada subclase se definirán los elementos que se componen, y se integrarán en una sola ventana para componer el programa completo. El programa final constará de las siguientes clases:

- **Mainwindow:** Se trata de la ventana principal, de tipo QMainWindow (QWidget), desde la cual se irá avanzando en el BCI. Es la GUI desde la que se ejecutará el programa, y se llamará a la misma desde main.cpp.

- **SettingsDialog:** Se trata de otra ventana, QDialog, también heredando de QWidget, que emergerá al pulsar el botón "Settings" contenido en Mainwindow. En esta ventana se podrán configurar los parámetros del puerto serie para habilitar la comunicación con Arduino.

- **Console:** Se trata de un objeto QPlainTextEdit (QWidget) que, a pesar de ser un objeto dentro de la ventana, no se encuentra sólo dentro de la ventana, sino que se comunica también con SettingsDialog y el resto de clases ya que se ha implementado con el objetivo de convertirla en la terminal virtual de comandos desde la cual se pueden enviar mensajes a Arduino (puerto serie).

- **Mythread:** Esta clase será un hilo de tipo QThread, que hereda directamente de QObject y no de QWidget, pudiendo de esta manera ejecutar procesos de forma paralela al programa. Se utilizará para poder de manera simultánea coordinar la iluminación de los estímulos con Arduino, la GUI y la lectura, procesado e interpretación de la actividad cerebral registrada con el casco Emotiv.

Estas cuatro clases estarán en permanente contacto, interrelacionándose entre ellas mediante mecanismos de Signals & Slots. Será necesario hacer uso de las propiedades de Qt para la elaboración de esta GUI, teniendo en cuenta el sistema jerárquico de clases, la gestión de eventos y el mecanismo setter y getter de cada clase, subclase y objeto.

3.4.2 Integración con Arduino

Tal y como se ha visto en el apartado anterior, para la integración con Arduino se comenzará explicando las clases *SettingsDialog* y *Console*, así como las señales que habrá que emitir desde la ventana principal para invocar a estas clases.

3.4.2.1 Settings Dialog: Ventana de Configuración

Esta ventana es llamada desde la GUI principal (MainWindow), y puede ser invocada en cualquier momento de la ejecución, aunque el objetivo de la misma es poder configurar los parámetros del puerto serie y este paso debería ser el primero a la hora de poner en funcionamiento la interfaz.

En esta ventana no se conecta el puerto serie, simplemente se pueden ajustar los parámetros del mismo. Para realizar la conexión será necesario pulsar el botón de conexión.

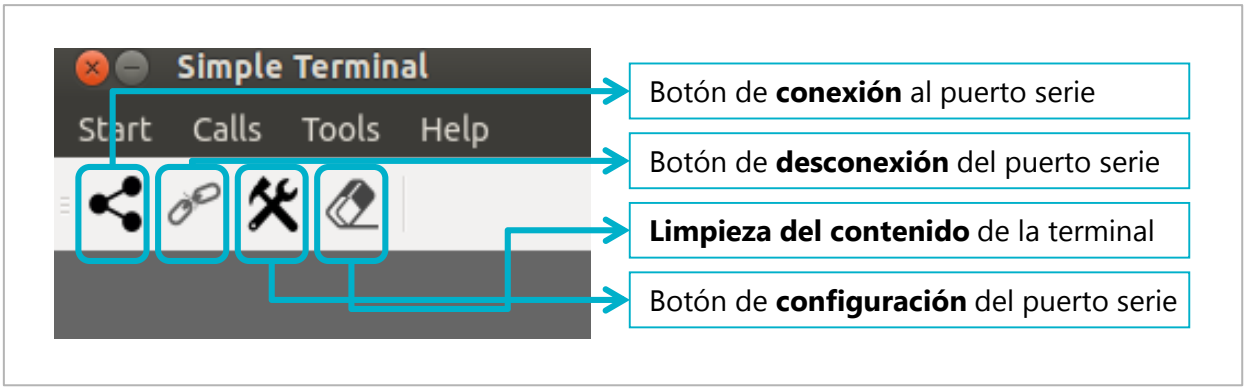

Figura 14. Captura de pantalla de la esquina superior izquierda de la GUI

Como se puede apreciar en la figura 14, el **botón de configuración** hará aparecer una ventana emergente "Settings", que se muestra debajo en la figura 15.

En la ventana Settings se podrá seleccionar el puerto serie al que se quiere conectar, que tendrá que coincidir con el puerto en el que esté escuchando el microprocesador Arduino. Se

proporcionará también información adicional sobre el puerto, pudiendo también seleccionar el Baud Rate (que en este caso será siempre de 9600 para poder coincidir con el de Arduino), el tamaño de los paquetes de información, si se desea emitir un bit de paridad y configuración del flujo de datos.

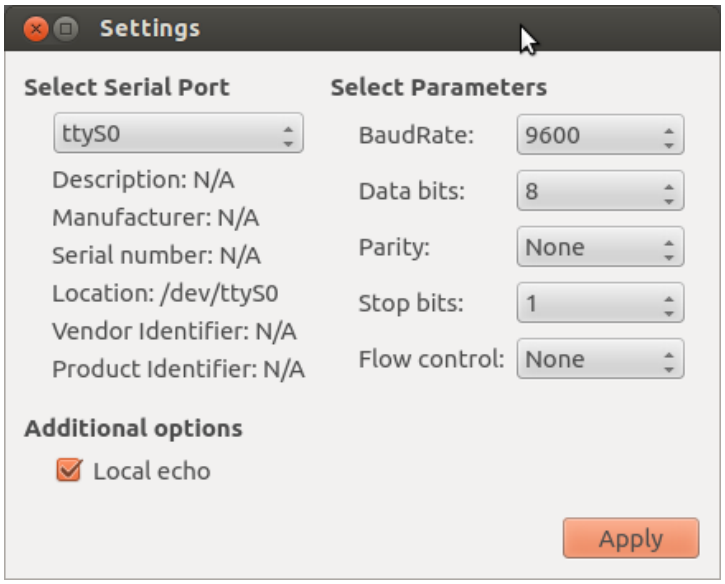

Figura 15. Ventana de configuración del puerto serie

Para poder tener acceso a estos parámetros y a la información del puerto serie, se hará uso de las siguientes librerías proporcionadas por Qt:

```
#include <QtSerialPort/QSerialPortInfo>
#include <QIntValidator>
```

Mediante las mismas se consigue acceder a la información del puerto serie que permite configurarlo con las siguientes funciones:

```
private slots:
    void showPortInfo(int idx);
    void apply();
    void checkCustomBaudRatePolicy(int idx);
    void checkCustomDevicePathPolicy(int idx);

private:
    void fillPortsParameters();
    void fillPortsInfo();
    void updateSettings();
```

Se proporciona información adicional sobre cada una de estas funciones en el "Anexo II: Código Implementación GUI".

La ventana Settings Dialog será una ventana emergente de tipo QDialog que configurará los parámetros del puerto serie gracias a la librería QtSerialPort/QSerialPortInfo facilitada por Qt.

3.4.2.2 *Console: La terminal de Comandos*

Mientras que a través de la ventana Settings Dialog es posible configurar el puerto, es necesario un elemento adicional encargado de enviar la información a través del puerto que se ha implementado. Es por ello que se utiliza una clase Console, que no deja de ser un objeto de texto plano QPlainTextEdit que se implementa como terminal virtual. El texto que se escribe, bien de manera manual o insertándolo durante el programa, se muestra en la consola a la vez que es enviado a través del puerto serie. De esta manera, y tal y como se ha explicado en el apartado "3.3.1 Integración con QT", enviando un mensaje con una cabecera que el Arduino es capaz de identificar como mensaje de configuración de los LEDs, se enviará el mensaje a través de Qt y se recepcionará en el Arduino.

Desde una visión más global, un usuario se registrará y se le asignará un número de usuario. Una vez se hayan detectado sus mejores frecuencias, se almacenarán en un fichero del tipo "frecuencias_S1.txt" (para el caso del Usuario 1), de manera que el contenido de este .txt sea idéntico al mensaje que Arduino espera recibir. A la hora de ejecutar el programa, Qt leerá de este fichero las frecuencias del usuario que haya entrado a utilizar el BCI, y las adjuntará en la consola, que se ha convertido en una terminal virtual en permanente comunicación con Arduino. Se muestra a continuación un extracto del código Qt que habilita estas funciones:

```
QFile file(RUTA_Usuario); // Lee el fichero de frecuencias del Usuario

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        console->appendPlainText("El fichero no se ha abierto");
        }

        while (!file.atEnd()) {

    //Se leen las frecuencias y se guardan como QString
            frecuencias_completas=file.readLine();
            lista_frecuencias= frecuencias_completas.split(" ",
QString::SkipEmptyParts);
            frecuencia1=lista_frecuencias.at(0);
            frecuencia2=lista_frecuencias.at(1);
            frecuencia3=lista_frecuencias.at(2);
            frecuencia4=lista_frecuencias.at(3);
            frecuencia5=lista_frecuencias.at(4);
            frecuencia6=lista_frecuencias.at(5);
        }
        frecuencias_completas =  cabecera + " 1 " + frecuencia1 + " " +
frecuencia2 + " " + frecuencia3 + " " + frecuencia4 + " " + frecuencia5 + " " +
frecuencia6;
        console->appendPlainText(frecuencias_completas);

        emit console->getData(frecuencias_completas.toLocal8Bit());
```

Donde `console->getData` es una señal (SIGNAL) conectada al SLOT "`writeData`", que recibe el parámetro de `frecuencias_completas` emitido por la señal y realiza la siguiente función:

```
void MainWindow::writeData(const QByteArray &data)
{
    serial->write(data);
    console->appendPlainText("Se han mandado datos al puerto serie:");
    console->putData(data);

}
```

La conexión entre SIGNAL y SLOT en este caso se realiza de la siguiente manera:

```
connect(console,SIGNAL(getData(QByteArray)), this, SLOT(writeData(QByteArray)));
```

Con estos dos objetos, la ventana de configuración del puerto serie y la terminal virtual que se muestra desde la GUI, es posible realizar la conexión de manera adecuada con Arduino.

En el siguiente apartado se detallará la implementación de la Interfaz Gráfica de Usuario completa.

3.4.3 Implementación de la GUI

El objetivo que se persigue con la implementación de esta Interfaz Gráfica de Usuario es el que se ha descrito previamente a lo largo del apartado "3.1 Planteamiento Inicial y Diseño de la Interfaz", es decir, una aplicación compuesta por seis objetos (cajas cuadradas de diferentes colores) con opciones alternativas que permitan al sujeto que utilice el BCI realizar una serie de juegos, deletrear palabras, contestar a preguntas de Sí/No e informar de una serie de acciones y sentimientos gracias a la aplicación *Easy Life*.

En este apartado se va a describir el código que se ha elaborado, las diferentes funciones y objetos a los que se han invocado para lograr hacer real la planificación inicial de la GUI.

Debido a la extensión de este apartado de la implementación, y a la cantidad de funciones implementadas, se va a dividir este apartado en diferentes secciones:

- Funciones implementadas para la configuración de la GUI.
    - Aquellas funciones de inicialización de variables, creación y borrado de la ventana, etc.
- Conexiones entre señales y slots.
    - Se trata de una única función que se encarga de realizar las interconexiones del programa; por ejemplo, que al pulsar un determinado botón se ejecuten una serie de funciones del código.
- Funciones invocadas a partir de la pulsación de botones.
    - A cada botón le corresponde una función, y dependiendo del momento y del estado en el que se encuentre, se realizarán una serie de acciones.
- Máquina de estados.
    - Debido a la gran cantidad de opciones y desplegables a medida que se avanza por la aplicación, se ha creado una máquina de estados que, en función del estado en el que se encuentre cuando se detecta un estímulo, se mueve hacia un estado nuevo u otro.
- Slots que se activan cuando se detecta un estímulo.

- Por último, se describen las medidas que se toman cuando se detecta un estímulo. Inicialmente se realizará la conexión de manera virtual a partir de seis botones, que actuarán como estímulos detectados procedentes del registro de un EEG. Posteriormente, en el apartado "3.5 Comunicación entre GUI y EEG Emotiv" se describirá cómo se ha sustituido la funcionalidad de estos botones por la emisión de una señal cuando el registro de EEG detecta una frecuencia.

3.4.3.1 Funciones de Configuración de la GUI

El programa comenzará invocando a la clase MainWindow que será el objeto principal, de tipo QWidget, del que heredarán el resto de objetos implementados a lo largo del programa.

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)

{
```

En esta primera función principal, se inicializan todas las variables a utilizar a lo largo del programa, se crea una interfaz gráfica "ui" instanciada en la clase MainWindow, y se establecen sus dimensiones en la pantalla. Se instancia también la consola virtual, y se fijan sus dimensiones en la pantalla:

```
// Se crea la consola que irá mostrando el proceso y actuará como terminal.
    console = new Console(this);
    console->setEnabled(true);

    //Se fija como una ventana en el centro
    //Fijo ancho y alto
    console->setFixedSize(ancho_consola,alto_consola);
    //Lo centro en la ventana

    console->move(ancho_pantalla/2-ancho_consola/2, alto_pantalla/2);
```

También se llama a la función `setInicio()`, que se encarga de esconder todos los objetos de la interfaz gráfica para poder mostrar únicamente el mensaje de bienvenida y la solicitud de registro del usuario. Esto se hace así debido a que todos los objetos, ya sean botones, mensajes de estado, etiquetas, texto plano y demás formas, se crean desde un primer momento y se encuentran dentro de la GUI. Es decir, no se van creando conforme avanza la aplicación sino que existen desde su creación, pero no se muestran. A modo de ejemplo, se muestra un extracto de la función setInicio(). La función completa se encuentra en el "Anexo II Código Implementación GUI".

```
void MainWindow::setInicio()
{
. . .
    //Se muestra el texto de bienvenida
    ui->Bienvenido->move(ancho_pantalla/2-ui->Bienvenido->width()/2,
alto_pantalla/4);

. . .
    //Se configuran los botones de Nuevo Usuario y Usuario Regustrado
```

```
    ui->Boton_NuevoUsu->move(ancho_pantalla/2-ui->Boton_NuevoUsu->width()/2-
distancia_botones/2, alto_pantalla/2);

. . .
    //Todo lo demás se esconde hasta que se pulse uno de los dos botones
    ui->boton_inicio->hide();
    ui->label_InfoNuevoUsu->hide();
    ui->label_introducenombre->hide();
    ui->line_NombreUsu->hide();
    console->hide();
    ui->estimulo1->hide();
. . .
}
```

Tras la llamada a esta función, se habilita el puerto serie y se establecen las opciones que se mostrarán en el menú superior izquierdo de la interfaz (menú de Inicio, botones de configuración del puerto serie, etc).

Las funciones de configuración que se invocan en este momento o en otros a lo largo del código son las siguientes (para ver el detalle de las mismas consultar el "Anexo II Código Implementación GUI"):

- **~MainWindow:** Si se invoca, finaliza el programa.
- **setInicio():** Como se ha visto en este apartado, coloca los botones en sus posiciones y esconde los estímulos hasta que se realice la configuración inicial.
- **getInstance():** Se utilizará para obtener una instancia de la ventana MainWindow desde otra clase (por ejemplo, desde un hilo que se va a ejecutar en paralelo a este programa).
- **Delay(int seconds):** En Qt no es posible llamar a funciones como sleep, wait o pause debido al mecanismo de espera a cambios y a señales. Por tanto, es necesario implementar la función delay de manera manual para no pausar el sistema.
- **showStatusMessage(const QString &message):** Configuración de un mensaje de estado en la barra inferior del programa. Cuando se llama a esta función, se muestra el mensaje que se recibe como parámetro (de tipo QString).
- **openSerialPort():** Esta función se ejecutará cuando se pulse el botón de Conexión, y configura los parámetros del puerto serie.
- **closeSerialPort():** Desconecta el puerto serie, se ejecutará al pulsar el botón de desconexión.
- **about():** Muestra una breve descripción del programa. Se ejecuta al pulsar el botón About.
- **writeData(const QByteArray &data):** Envía los datos que recibe como parámetro al puerto serie a través de la terminal virtual de comandos.

- **readData():** Recibe los datos desde el puerto serie y los muestra en la terminal virtual de comandos.

- **handleError():** En caso de producirse un error durante la conexión al puerto serie, informa del mismo y cierra dicha conexión.

Por último, después de la inicialización se llamará a la función `initActionsConnections()`, que establecerá las conexiones internas de la GUI.

3.4.3.2 Mecanismo SIGNALS & SLOTS

Las conexiones entre señales y slots se realizan en la función `initActionsConnections()`, y engloba todas aquellas conexiones que garantizan el correcto funcionamiento del programa.

Para realizar una conexión, se hace uso de la siguiente estructura definida por Qt:

```
connect(objeto A, SIGNAL(señal A(parámetros)), objeto B, SLOT (slot B(parámetros)))
```

Con esta estructura, cuando se emita una señal "señal A" definida dentro de la clase "objeto A", se ejecutará el código del slot "slot B", dentro de la clase "objeto B". Los parámetros de la señal A y el slot B tendrán que ser del mismo tipo, ya que la señal pasará esos parámetros al slot.

Una señal se puede emitir al pulsar un botón, al hacer click en el ratón, al tomar una variable un valor concreto o simplemente al emitirse desde un punto del código (`emit señal A`).

Por tanto, en esta función se engloban todos los `connect` necesarios para este programa, y se deben establecer desde el inicio del mismo.

Habrá que conectar los botones de acción del menú superior izquierdo a las funciones correspondientes a los mismos:

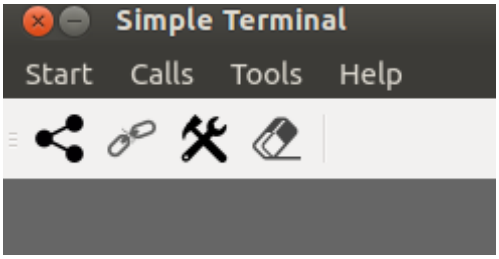

Figura 16. Barra de opciones de la esquina superior izquierda

Para estas opciones, las conexiones serán las siguientes:

```
//Conexión de los botones de acción
connect(ui->actionConnect, SIGNAL(triggered()), this, SLOT(openSerialPort()));
connect(ui->actionDisconnect, SIGNAL(triggered()), this, SLOT(closeSerialPort()));
connect(ui->actionQuit, SIGNAL(triggered()), this, SLOT(close()));
connect(ui->actionConfigure, SIGNAL(triggered()), settings, SLOT(show()));
connect(ui->actionClear, SIGNAL(triggered()), console, SLOT(clear()));
connect(ui->actionAbout, SIGNAL(triggered()), this, SLOT(about()));
connect(ui->actionAboutQt, SIGNAL(triggered()), qApp, SLOT(aboutQt()));
```

Para el Puerto serie, las conexiones serán:

```
// Si ocurre un error en la apertura del puerto serie, notificar en la ventana principal
    connect(serial, SIGNAL(error(QSerialPort::SerialPortError)), this,
SLOT(handleError(QSerialPort::SerialPortError)));

// Si el puerto serie está preparado para ser leído, llama a la GUI para que lo lea
    connect(serial, SIGNAL(readyRead()), this, SLOT(readData()));

//Desde la consola mandar datos al puerto serie
    connect(console, SIGNAL(getData(QByteArray)), this, SLOT(writeData(QByteArray)));
```

Por último, se definen también las conexiones entre la GUI y la detección de un estímulo asociado a uno de los seis objetos de referencia. Se crean también estas mismas conexiones para la pulsación de los "Botones de Ayuda", que se implementarán inicialmente para emular el comportamiento de la detección de estímulos:

```
    //Conexion de estados. Cuando se emite la señal Signal_EstimuloX, se activan
    // los procedimientos de detección de estímulos
    connect(this, SIGNAL(Signal_Estimulo1()), this, SLOT(Estimulo1Detectado()));
    connect(this, SIGNAL(Signal_Estimulo2()), this, SLOT(Estimulo2Detectado()));
    connect(this, SIGNAL(Signal_Estimulo3()), this, SLOT(Estimulo3Detectado()));
    connect(this, SIGNAL(Signal_Estimulo4()), this, SLOT(Estimulo4Detectado()));
    connect(this, SIGNAL(Signal_Estimulo5()), this, SLOT(Estimulo5Detectado()));
    connect(this, SIGNAL(Signal_Estimulo6()), this, SLOT(Estimulo6Detectado()));


    //Se activan también estos mismos procedimientos si se pulsan los botones de ayuda
    connect(ui->pushButton_1, SIGNAL(clicked()), this, SLOT(Estimulo1Detectado()));
    connect(ui->pushButton_2, SIGNAL(clicked()), this, SLOT(Estimulo2Detectado()));
    connect(ui->pushButton_3, SIGNAL(clicked()), this, SLOT(Estimulo3Detectado()));
    connect(ui->pushButton_4, SIGNAL(clicked()), this, SLOT(Estimulo4Detectado()));
    connect(ui->pushButton_5, SIGNAL(clicked()), this, SLOT(Estimulo5Detectado()));
    connect(ui->pushButton_6, SIGNAL(clicked()), this, SLOT(Estimulo6Detectado()));
```

Estas son las conexiones necesarias para el funcionamiento de la GUI. Para pulsaciones de otros botones, se realizará directamente en las funciones asignadas a los mismos, definidas a continuación.

3.4.3.3 Configuración de los Botones

En este apartado se definirá la funcionalidad de tres botones principales de configuración: el botón de creación de Nuevo Usuario (NuevoUsu), el de validación de Usuario Registrado (RegUsu) y el botón que se utiliza tanto como para grabar el baseline como para comenzar el programa (botón inicio).

En Qt, al crear un objeto de tipo QWidget como puede ser un botón, se puede crear una función de tipo `on_button_clicked()`, que se ejecutará de manera automática al pulsar el botón al que hace referencia. Por tanto, existen dentro de este alcance tres funciones que se corresponden a cada uno de los tres botones:

- void MainWindow::on_Boton_NuevoUsu_clicked()
- void MainWindow::on_Boton_RegUsu_clicked()

- void MainWindow::on_boton_inicio_clicked()

En este apartado se describe el proceso que se realiza para ejecutar las actividades asociadas a cada botón, mientras que en el "Anexo II Código Implementación GUI" se proporciona una el código completo utilizado para implementar dichas funciones.

Cuando se ejecuta la GUI, la pantalla que se mostrará será la de la figura 17. Tras un mensaje de bienvenida, se mostrarán dos botones de cara a iniciar sesión. La primera opción será la de **Nuevo Usuario**, si es la primera vez que se utiliza el programa, y la segunda la de **Usuario Registrado**, si el sujeto está almacenado en la base de datos de la aplicación.

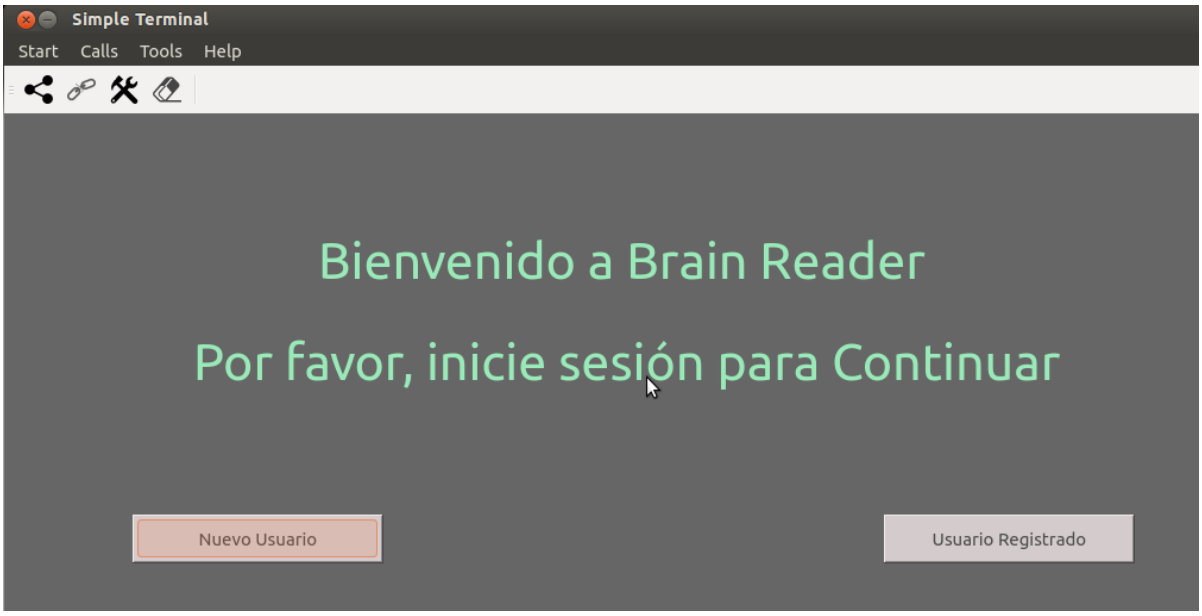

Figura 17. Pantalla Inicial de la GUI

Cuando se pulsa el botón **"Nuevo Usuario"**, aparecerá el siguiente mensaje por pantalla:

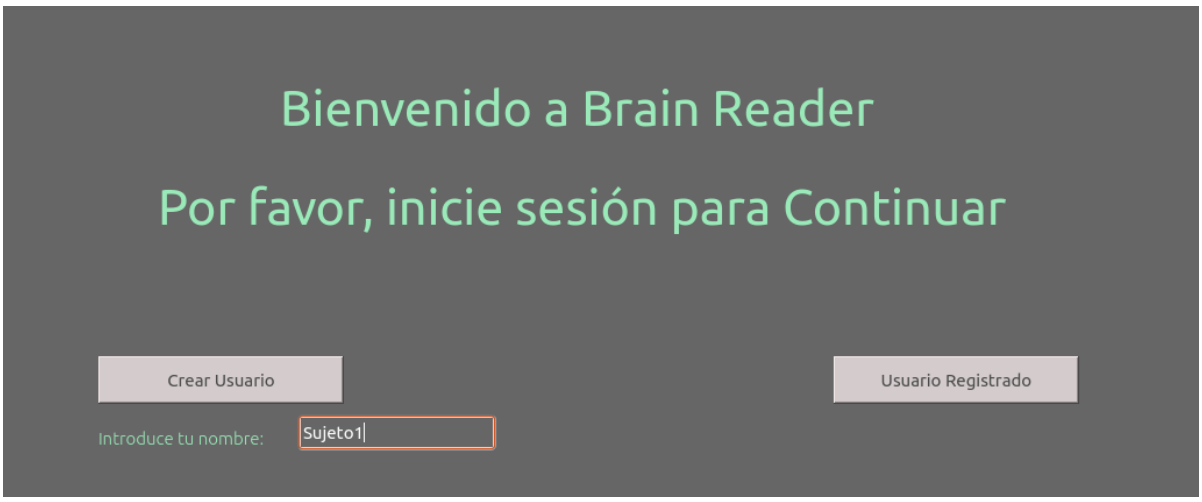

Figura 18. GUI al pulsar el botón de Crear Usuario

Se observa que aparece un mensaje y un cuadro de texto en el cual el usuario puede registrarse. Además, el contenido de texto del botón ha cambiado de "Nuevo Usuario" a "Crear Usuario". Cuando se pulse el botón de "Crear Usuario" (tratándose del mismo objeto QWidget), el programa leerá del cuadro de texto, almacenará su contenido (Sujeto1) y le asignará un número de usuario. Para ello, abrirá el fichero utilizado como base de datos y leerá uno a uno cada usuario registrado, verificando así que se trata de un usuario nuevo. Después, al alcanzar el final del fichero, utilizará el número del usuario del último registro y lo incrementará en una unidad

Una vez guarde el registro del usuario, aparecerá un nuevo botón que al pulsarse comenzará a grabar el baseline.

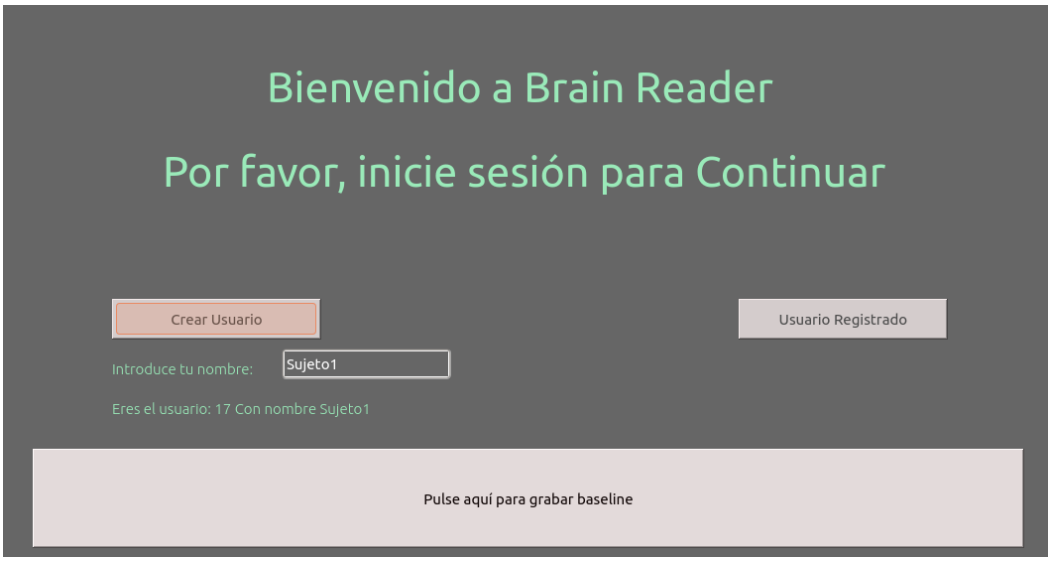

Figura 19. El Usuario se registra y se puede comenzar a grabar el baseline

La segunda opción sería comprobar si el usuario existe y se encuentra registrado. Se volverá a abrir el fichero utilizado como base de datos de registro y se leerá cada entrada, comparándose con el nombre introducido. En caso de encontrar una coincidencia, se buscará su número de registro y de nuevo aparecerá la opción de comenzar a grabar el baseline. Si no se encontrase dicho nombre de usuario en el registro, la GUI informará del error, y dará opción a volver a introducir usuario o bien crear uno nuevo.

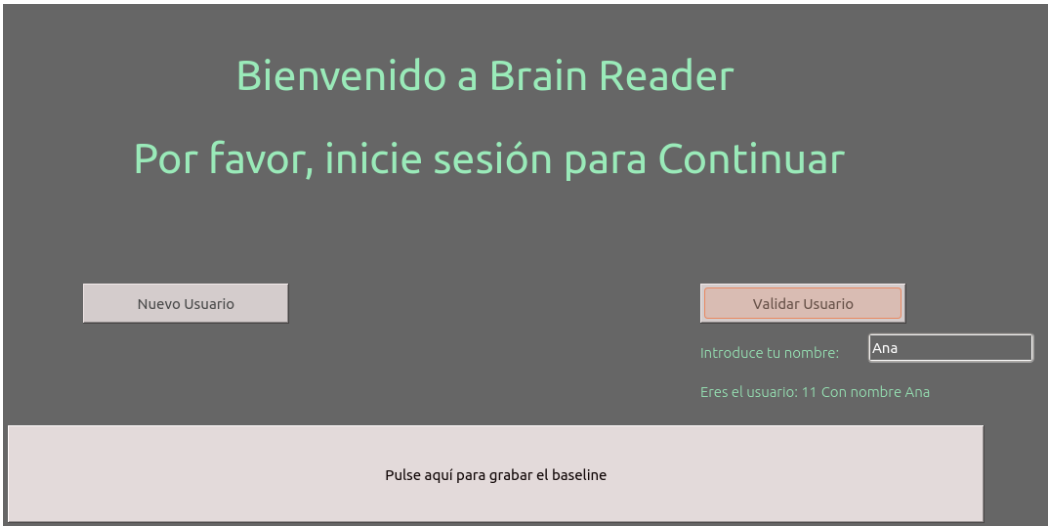

Figura 20. Validación de un Usuario registrado

El tercer botón (botón inicio) se trata por tanto del encargado de tres acciones diferentes, en función del momento del programa en el que se pulse. La primera vez que se muestra, tras el registro inicial, su contenido muestra el mensaje "Pulse aquí para grabar el baseline". Al apretar el botón, se realizarán las siguientes acciones:

- En primer lugar, desaparecerán todos los objetos de la pantalla (incluyendo el botón inicio), quedando ésta vacía y con el fondo negro.

- Una vez establecida la pantalla negra, desde la GUI se llamará a otro programa que se va a ejecutar en paralelo, y que ha de encontrarse en la misma carpeta de ejecución que la del programa de la GUI. Este programa, de nombre "baseline", recibe como argumento el número de usuario del sujeto y comienza una comunicación con el casco Emotiv EPOC.

- Por un lado, el programa baseline leerá durante un minuto la información de registro de la actividad cerebral a través de los electrodos, almacenándolos en un fichero "baseline.txt" y "brutos_baseline.txt" dentro de una carpeta "Usuario_X", donde X es el número del usuario [44].

- Por otro lado, la GUI permanecerá durante un minuto "apagada", indicando al microprocesador Arduino que no encienda ningún LED, y sin mostrar ningún objeto en la pantalla. Para esto es necesario hacer uso de la función `delay(int segundos)`, implementada de manera manual debido a las restricciones de Qt a la hora de pausar una interfaz gráfica.

- Finalmente, transcurrido este tiempo, el programa baseline finalizará, se cerrará el proceso y volverá a aparecer el botón inicio, esta vez con el mensaje "Pulse aquí para comenzar el programa".

Para realizar la llamada al programa baseline, que se ejecutará en paralelo, se utiliza el objeto QProcess de la librería Qt

```
#include <QProcess>

. . .
void MainWindow::on_boton_inicio_clicked(){

. . .
            //Llamada al programa baseline
            QProcess *procesoBaseline = new QProcess(this);
            QString programaBaseline = ARCHIVO_BASELINE;
            QStringList argumentos;
            argumentos << numero_usuario;

            procesoBaseline->start(programaBaseline, argumentos);
            procesoBaseline->waitForStarted();
}
```

*Cabe destacar que el diseño de esta GUI se ha realizado de manera que el casco a emplear, en este caso Emotiv EPOC, pueda **ser sustituido por otro casco** en función de las circunstancias. Sería suficiente con sustituir el ejecutable "ARCHIVO_BASELINE" con el ejecutable correspondiente a la grabación del baseline de otro casco, reforzando así la **flexibilidad y universalidad** de este BCI.*

La segunda vez que se pulsa el botón de inicio, se muestra por pantalla el menú principal con los seis estímulos configurados y en posición alineada con los LEDs.

Figura 21. Menú Principal que da comienzo al BCI

Se realizarán en este caso las siguientes actividades:

- Inicialmente, se llamará a la función `setPosiciones()`, que hará que aparezcan por pantalla los seis estímulos con su configuración inicial, mostrada en la figura 21.

- Se inicializará la variable "estado" al estado inicial, 10 y se llamará a la máquina de estados (ver siguiente apartado, Máquina de Estados).

- Se abrirá el fichero de mejores frecuencias asignado al usuario registrado, previamente calculado en la selección de frecuencias, almacenando cada valor de las seis frecuencias, y configurando el mensaje que se habrá de enviar al microprocesador Arduino.

- Se añadirá la cabecera correspondiente al mensaje, y se enviará la configuración de frecuencias a Arduino a través de la consola virtual.

```
emit console->getData(frecuencias_completas.toLocal8Bit());
```

- Por último, de la misma manera que se ha llamado al programa "baseline" para grabar el registro de baseline, en este caso se crea un proceso "ejecucionEmotiv" que será el encargado de aquí en adelante de leer la actividad cerebral del sujeto a través del casco Emotiv EPOC, procesar la señal e interpretar los resultados obtenidos, devolviendo a la GUI la frecuencia que se ha detectado al realizar la transformada de Fourier a los registros recibidos por los electrodos, utilizando el algoritmo desarrollado en trabajos anteriores [37] [44].

El programa ejecucionEmotiv recibe como argumentos de entrada dos parámetros: en primer lugar, el número de usuario para acceder a su registro de baseline y a sus mejores frecuencias, y en segundo lugar el id de una Cola que se ha creado previamente. Será en esta cola donde el programa ejecucionEmotiv vaya introduciendo las frecuencias que vaya detectando, y la GUI recibirá a través de un hilo dichos resultados (esto se explicará más adelante en el apartado "3.5 Comunicación entre GUI y EEG Emotiv").

La llamada al programa que se ejecutará en paralelo se realizará de nuevo mediante la creación de un elemento QProcess, de la siguiente manera:

```
        QProcess *procesoejecucionEmotiv = new QProcess(this);
        QString programaejecucionEmotiv = ARCHIVO_EJECUCIONEMOTIV;
        QString idCola_string = QString::number(idCola);
        QStringList argumentos_ejecucionEmotiv;
        argumentos_ejecucionEmotiv << numero_usuario << idCola_string;

        procesoejecucionEmotiv->start(programaejecucionEmotiv,
argumentos_ejecucionEmotiv);
        procesoejecucionEmotiv->waitForStarted();
```

De nuevo, cambiando el fichero ejecutable se podría adaptar esta interfaz gráfica a diferentes sistemas de adquisición de señal sin perder la interoperabilidad.

Una vez realizados estos pasos, la Interfaz Cerebro – Máquina se encuentra preparada para utilizarse, con los LEDs activados y funcionando como estímulos BCI basados en SSVEP, el casco

Emotiv enviando información, un proceso ejecutándose en paralelo recibiendo los datos e interpretándolos y por último una Interfaz Gráfica mostrando las diferentes opciones que se pueden mostrar al usuario, todo ello funcionando en paralelo.

3.4.3.4 Máquina de Estados

Volviendo al menú de la imagen mostrada en el apartado anterior, Figura 21, se observa que ese estado se trata del menú inicial, al que se le ha asignado el estado "10".

Como se ha visto en la introducción a Qt, una vez se llama a una GUI se ejecuta el "Event Loop", entendido como un bucle indefinido que espera a que cambien los inputs de la aplicación mediante la emisión de señales, en lugar de ser un código de programación que se ejecuta secuencialmente.

En este menú principal se muestran seis opciones diferentes: Juegos, Speller, Sí/No, Easy Life, Recalibrar y Cerrar Sesión. Cabe esperar por tanto que bajo este modelo de programación, la navegación a través de la aplicación se realice a través de señales asociadas a cada uno de los estímulos. Para tal fin, se han definido seis señales, una asociada a cada caja mostrada por pantalla, que a su vez está asociada a un LED y este LED está asociado a una determinada frecuencia:

```
signals:
    //Señales que se emiten cuando se detecta un estímulo
    void Signal_Estimulo1();
    void Signal_Estimulo2();
    void Signal_Estimulo3();
    void Signal_Estimulo4();
    void Signal_Estimulo5();
    void Signal_Estimulo6();
```

En una primera aproximación, y para facilitar la flexibilidad de esta GUI, se asoció la emisión de cada señal a la pulsación de un botón colocado en cada una de las cajas. Posteriormente se ha incluido también la emisión de estas señales cuando se detecta una frecuencia desde el casco Emotiv.

Sin embargo, no es suficiente con la emisión de una señal para saber cuál es la intención del usuario, ya que las actividades a realizar no serán las mismas si se detecta el estímulo 1 cuando el menú muestra opciones de deletreo que cuando muestra un menú que ofrece diferentes Juegos.

Por tanto, es imprescindible que la GUI sea capaz de detectar en todo momento en qué menú está y a qué menú puede ir, y esto se ha implementado a partir de una **máquina de estados**.

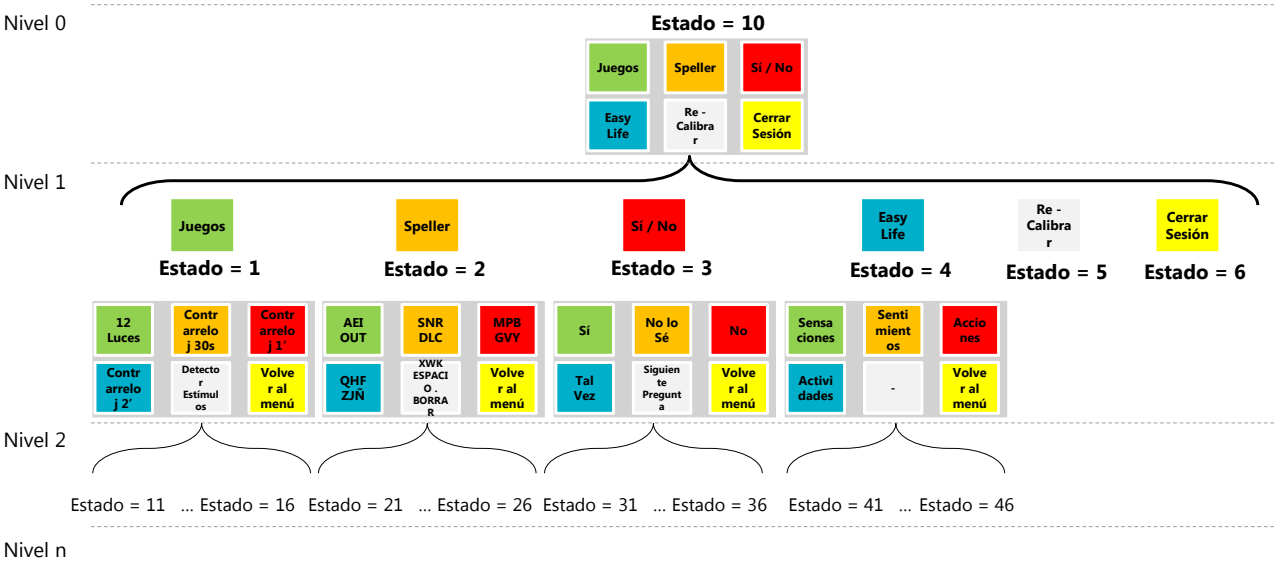

Figura 22. Máquina de Estados

Se creará para este fin una función principal controladora de estados: `void MainWindow::checkEstados(int estados)`. Esta función recibirá como argumento el estado en el que se encuentre, y llamará a diferentes funciones encargadas de la configuración de la GUI. Se muestra a continuación un fragmento del código de esta función:

```
void MainWindow::checkEstados(int estados)
{
    switch (estados)
    {
        //Si el estado es 1, entra a JUEGOS
        case 1:
            setEstado1();
            break;
        //Si el estado es 2, entra a SPELLER
        case 2:
            setEstado2();
            break;

. . .
}
```

Para cada uno de los posibles estados (existen tres niveles de profundidad), se llamará a una función diferente que configurará los estímulos y realizará las actividades correspondientes. Los estados posibles serán los siguientes:

- (1) Juegos. Con la finalidad de medir la efectividad del BCI, se proponen cinco pruebas:
  - (11) 12 Luces: Se trata de un juego para medir el tiempo que el sujeto tarda en detectar correctamente 12 Luces, almacenando también el número de fallos. A través de la terminal se indicará cuál es la luz a la que ha de fijar su atención

cada vez, asignando un número a cada estímulo, y al finalizar mostrará el resultado obtenido.

- (12) Contrarreloj 30 segundos: Cuenta el número de estímulos capaz de detectar en 30 segundos. Al finalizar muestra el resultado por pantalla e indica si ha superado un récord previo o no.
- (13) Contrarreloj 1 minuto: Cuenta el número de estímulos capaz de detectar en 1 minuto. Al finalizar muestra el resultado por pantalla e indica si ha superado un récord previo o no.
- (14) Contrarreloj 3 minutos: Cuenta el número de estímulos capaz de detectar en 3 minutos. Al finalizar muestra el resultado por pantalla e indica si ha superado un récord previo o no.
- (15) Detector Estímulos. El usuario fijará su atención en el estímulo que desee, y la aplicación informará de cuál es el estímulo que ha detectado. Servirá para medir de manera manual la tasa de acierto del sistema.
- (10) Volver al menú

- (2) Speller: Se ha configurado mediante una agrupación de letras que tiene en cuenta la frecuencia de aparición de cada letra en el vocabulario Al seleccionar cualquiera de las opciones que se muestran a continuación, el siguiente nivel será mostrar en cada estímulo una de las letras, y en el siguiente cambio de estado se identificará una de las letras y se volverá al estado 2.
  - (21) A E I O U T.
  - (22) S N R D L C
  - (23) M P B G V Y
  - (24) Q H F G J Ñ
  - (25) X W K ESPACIO . BORRAR
  - (10) Volver al menú

- (3) Sí / No: Diseñado con el fin de proporcionar una comunicación rápida y sencilla, a través de esta opción un usuario sería capaz de responder a preguntas realizadas de forma oral con las siguientes opciones:
  - (31) Sí
  - (32) No lo sé
  - (33) No
  - (34) Tal vez

- (35) Siguiente Pregunta
- (10) Volver al Menú

- (4) Easy Life: Esta opción recopila algunas de las principales acciones que muchas personas con discapacidad física desean comunicar y en ocasiones no le resulta posible. Se ha realizado con la ayuda del centro Cottolengo de Madrid, dejando flexibilidad para configurar las opciones de manera adaptada al sujeto:

  - (41) Sensaciones
    - (411) Calor
    - (412) Frío
    - (413) Hambre
    - (414) Sueño
    - (415) Cansancio
    - (4) Volver
  - (42) Sentimientos
    - (421) Triste
    - (422) Alegre
    - (423) Soledad
    - (424) Amor
    - (425) Aburrimiento
    - (4) Volver
  - (43) Acciones
    - (431) Ir al baño
    - (432) Comer
    - (433) Salir a la calle
    - (434) Ir a la habitación
    - (435) Dar un paseo
    - (4) Volver

  - (44) Actividades
    - (441) Encender TV
    - (442) Apagar TV
    - (443) Escuchar música
    - (444) Leer

        - (445) Llamar a alguien
        - (4) Volver
    - (45) Opciones adaptadas al usuario
    - (10) Volver al Menú

- (5) Recalibrar
    - Vuelve al estado previo a la grabación del baseline y registro de mejores frecuencias.

- (6) Cerrar Sesión
    - Finaliza la ejecución de la aplicación y de todos los procesos que se ejecutan en paralelo.

Un ejemplo de código de configuración sería el siguiente:

```
void MainWindow::setEstado2()
{
    //Fijar contenido de los cubos
      ui->label_1->setText("A E I \n O U T");
      ui->label_2->setText("S N R \n D L C");
      ui->label_3->setText("M P B \n G V Y");
      ui->label_4->setText("Q H F \n Z J Ñ");
      ui->label_5->setText("X W K \n espacio . borrar");
      ui->label_6->setText("Volver \n al menú");
}
```

Y, cuando se seleccionase una letra, se mostraría a través de pantalla de la siguiente manera:

```
void MainWindow::checkEstados(int estados)
{
    switch (estados)
    {
. . .
        case 211:
            cadena_speller=cadena_speller + "A";
            console->appendPlainText(cadena_speller);
            estado = 2;

            break;
```

Estas serían todas las posibilidades a escoger dentro de la GUI implementada. En el "Anexo II Código Implementación GUI" se muestra el código que configura cada una de las opciones mencionadas.

3.4.3.5 SLOTS de Detección de Estímulos

Por último, cabe destacar las funciones que se ejecutan cuando se detecta un estímulo, bien por la pulsación de uno de los botones de ayuda o bien porque se ha emitido desde el hilo que se ejecuta en paralelo. Cuando esto ocurre, se ejecutará uno de estos seis SLOTS:

```
private slots:
        . . .

    void Estimulo1Detectado();
    void Estimulo2Detectado();
    void Estimulo3Detectado();
    void Estimulo4Detectado();
    void Estimulo5Detectado();
    void Estimulo6Detectado();
```

Desde cada uno de estos slots, se realizará una tarea sencilla: en primer lugar, se comprobará en qué estado se encuentra la GUI cuando se ha detectado el estímulo. Posteriormente, se cambiará el estado dependiendo del estado actual y el estado detectado (por ejemplo, si la GUI se encuentra en el estado 4 y se detecta el estímulo 2, el estado resultante pasará a ser el 42). Por último, se llamará a la función checkEstados(int estado) descrita en el apartado anterior, realizando de esta manera los cambios oportunos.

Con todo esto queda integrada la máquina de estados con la detección de estímulos, componiendo en una única Interfaz Cerebro – Máquina una combinación de diferentes elementos intercomunicados y dependientes.

3.5 Comunicación entre GUI y EEG Emotiv

A lo largo del capítulo " 3.4 Diseño de la Interfaz Gráfica de Usuario"se ha mencionado en diferentes momentos las llamadas a procesos externos y a un hilo que se va a ejecutar en paralelo. Se ha querido diferenciar este apartado del anterior debido a que uno de los objetivos de este trabajo es el de construir una Interfaz Gráfica para BCI flexible y adaptable a diferentes sistemas de adquisición.

En este apartado se desarrolla una de las posibles opciones de integración con un sistema de adquisición, en este caso el registro de EEG Emotiv EPOC. Al realizarse mediante una llamada a un proceso externo, resultará sencillo ser sustituido en un futuro por otro sistema de adquisición de diferentes características.

Tal y como se ha visto durante el desarrollo, cuando se pulsa el botón "Pulse aquí para comenzar el programa" se hace una llamada al programa "ejecucionEmotiv" que se va a ejecutar como un proceso paralelo de tipo QProcess. Sin embargo, entre cada uno de los procesos ha de existir una comunicación, de manera que "ejecucionEmotiv" pueda informar a la GUI de los estímulos que identifica. Para habilitar esta comunicación, después de estudiar las opciones disponibles y realizar varias pruebas, se ha implementado de la siguiente manera:

- En primer lugar, para habilitar la comunicación entre procesos se ha hecho uso de una **cola de mensajes**.

- Además, Qt no permite bloqueos para esperar a la recepción de un mensaje, ya que funciona mediante aparición de eventos, y la lectura de una cola requiere estar a la escucha en un bucle indefinido. Por esta razón se ha creado un **hilo de tipo QThread,** que empieza a ejecutarse al inicio del programa y emite las señales correspondientes a la detección de estímulos.

A continuación se describirá cada uno de estos elementos.

3.5.1.1 Creación de la Cola de Mensajes

Uno de los objetivos principales de este proyecto es la sencillez a la hora de utilizar la aplicación, evitando salidas y entradas a la aplicación para configurar elementos externos. Por tanto, la llamada a otros programas ha de resultar invisible para el usuario, cuyos únicos conocimientos tendrán que ser acerca del funcionamiento de la GUI.

Por tanto, resulta imprescindible facilitar la comprensión del programa haciendo automática la llamada a otros elementos que componen el BCI.

Para tal fin, se ha decidido que sea desde el programa principal, "main.cpp", antes de realizar la llamada a "MainWindow", cuando se cree una cola de mensajes que posteriormente se pasará como parámetro de entrada al programa "ejecucionEmotiv".

La variable **idCola** se definirá como un entero global, de manera que su valor se pueda leer desde cualquier función, clase o slot, y la cola se definirá entonces de la siguiente manera:

```
#include <syscall.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/msg.h> //Para incluir IPC_PRIVATE, IPC_CREAT e IPC_EXCL

int idCola;

int main(int argc, char *argv[])
{
    //Primero: Crear la cola de comunicación entre el casco y la GUI
    idCola = msgget(IPC_PRIVATE, IPC_PERMISOS|IPC_CREAT|IPC_EXCL);
    if (idCola < 0)
        qDebug("[WARNING] Fallo al crear la cola de comunicación con el robot");
    if(idCola > 0)
        qDebug("ID de la cola de mensajes creada");
. . .
}
```

Una vez se crea la cola, la aplicación se ejecutará con normalidad. Cuando llegue el momento de comenzar el BCI, tras el registro del usuario, la grabación del baseline y la selección de frecuencias, se realizará la llamada al programa ejecucionEmotiv que recibirá como parámetro esta identificación de cola.

Desde el programa ejecucionEmotiv, se recibirá el identificador, se aplicará el algoritmo de procesado a las señales recibidas desde el Emotiv EPOC y se introducirá el resultado en la cola de mensajes.

3.5.1.2 Recepción de estímulos desde QThread

Volviendo a la GUI, tras la creación de la cola de mensajes se crea el hilo que se va a lanzar en paralelo desde el principio hasta el fin de la utilización de la interfaz. La llamada al hilo, así como las conexiones necesarias para su intercomunicación se realizan desde el archivo principal main.cpp:

```
#include <QApplication>
#include "mainwindow.h"
#include "mythread.h"

// Llamadas al sistema
#include <syscall.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/msg.h> //Para incluir IPC_PRIVATE, IPC_CREAT e IPC_EXCL


int idCola;

int main(int argc, char *argv[])
{
 . . .
    QApplication a(argc, argv);

    //Se crea la ventana principal de la GUI
    MainWindow w;
```

```
    //Se crea el hilo que se ejecutará en paralelo, comunicándose con el casco
Emotiv
    MyThread mythread;

    // Mecanismo SIGNALS & SLOTS: Conexión entre el hilo y la GUI
    QObject::connect(& mythread, SIGNAL(Signal_Thread_Estimulo1()), & w,
SLOT(Estimulo1Detectado()));
    QObject::connect(& mythread, SIGNAL(Signal_Thread_Estimulo2()), & w,
SLOT(Estimulo2Detectado()));
    QObject::connect(& mythread, SIGNAL(Signal_Thread_Estimulo3()), & w,
SLOT(Estimulo3Detectado()));
    QObject::connect(& mythread, SIGNAL(Signal_Thread_Estimulo4()), & w,
SLOT(Estimulo4Detectado()));
    QObject::connect(& mythread, SIGNAL(Signal_Thread_Estimulo5()), & w,
SLOT(Estimulo5Detectado()));
    QObject::connect(& mythread, SIGNAL(Signal_Thread_Estimulo6()), & w,
SLOT(Estimulo6Detectado()));


    //Comienzan el hilo y la ventana GUI
    mythread.start();
    w.show();

    return a.exec();
}
```

Una vez creado el hilo, ejecutará lo siguiente indefinidamente hasta que se cierre la GUI:

- Se crearán seis contadores, cada uno asociado a la detección de un estímulo. Dado que el algoritmo de ejecución devuelve un resultado tras una ventana temporal de 2 segundos, se considerará que un estímulo ha sido detectado cuando se produzcan **tres resultados iguales consecutivos**.

- Esperará hasta que se reciba un mensaje a través de la cola de mensajes desde el programa ejecucionEmotiv.

```
errorTransmision = msgrcv(idCola, &mensaje, sizeof(mensaje),
0, MSG_NOERROR);
        if (errorTransmision < 0) {
            qDebug("Recepcion de datos fallida ");
        }
```

- Cuando lo reciba, comparará el valor obtenido con la última frecuencia recibida.
  - Si son iguales, incrementará el contador asociado a tal frecuencia en una unidad.
  - Si son diferentes, reseteará los seis contadores de estímulos.
- Tras esta comprobación, actualizará el valor de la última frecuencia recibida ajustándose al último mensaje.

- Una vez se ha recibido el valor de frecuencia, realizará una comparación entre esta frecuencia y las seis configuradas en los LEDs. En caso de encontrar una coincidencia, se incrementará en una unidad el contador asociado a ese estímulo.

- Cuando el contador alcance su valor máximo (tras varias pruebas se ha establecido este límite como tres frecuencias consecutivas iguales) se emitirá una señal que se ha conectado al SLOT de detección de frecuencias de la GUI y se reseteará el contador.

- La GUI ejecutará el código correspondiente a la detección del estímulo.

Con todos estos pasos, quedaría configurada la Interfaz Cerebro – Máquina y quedarían integrados todos los elementos que la conforman.

En definitiva, se ha desarrollado una Interfaz Cerebro – Máquina a partir de una aplicación implementada en Qt que muestra una Interfaz Gráfica de Usuario, un sistema estimulador basado en SSVEP a través de Arduino Uno, seis LEDs y un circuito acondicionador, un registro de EEG Emotiv EPOC y un sistema de procesado de señal. Cada uno de estos elementos está integrado e intercomunicado entre sí, constituyendo así una señalización mixta que además dota al BCI de una gran flexibilidad y capacidad de ser utilizado cambiando alguno de sus elementos por otro dispositivo. Este sistema altamente flexible permite también incorporar algoritmos de adaptación personalizada al usuario.

CAPÍTULO 4. EXPERIMENTOS Y RESULTADOS

En el capítulo anterior se ha descrito el proceso de elaboración e implementación de esta Interfaz Cerebro – Máquina que utiliza señalización mixta y adaptación personalizada al sujeto. A lo largo de este capítulo se detallarán las pruebas que validan su precisión, así como su capacidad como medio de comunicación.

De la misma manera que el "CAPÍTULO 3 Diseño y Desarrollo" se ha estructurado diferenciando la plataforma hardware del desarrollo software, en este capítulo se comenzará detallando las pruebas que se han realizado para **validar la precisión** del sistema **generador de estímulos SSVEP**.

Además, será necesario corroborar que los estímulos que se generan realmente se traducen en la generación de SSVEPs detectables, de manera que al registrar la actividad cerebral de esta región el resultado sea un pulso a la frecuencia configurada en los estímulos. Se validará tanto la **detección de un estímulo aislado** como la **detección de estímulos dada la interferencia** en el campo visual que surge de la presentación simultánea de estímulos.

4.1 Validación de la configuración de las frecuencias

Como primera aproximación, es necesario verificar que las frecuencias que se establecen en los LEDs se reproducen de manera fiel y precisa, ya que cualquier desviación en las mismas resultaría en una incorrecta interpretación de las instrucciones.

Para ello se ha hecho uso de un osciloscopio conectado a los pines a los que se habrán de conectar los LEDs. Esto validará la placa Arduino Uno como microprocesador capaz de generar estímulos en un BCI basado en SSVEP.

Frecuencia Arduino	Frecuencia Osciloscopio	Frecuencia Arduino	Frecuencia Osciloscopio	Frecuencia Arduino	Frecuencia Osciloscopio
1	1,000	**13**	12,999	**25**	24,980
2	2,000	**14**	14,000	**26**	25,970
3	3,000	**15**	14,999	**27**	26,980
4	4,000	**16**	15,999	**28**	27,950
5	5,000	**17**	16,998	**29**	28,990
6	5,999	**18**	17,999	**30**	29,980
7	6,999	**19**	18,990	**40**	39,980
8	7,999	**20**	19,980	**50**	49,890
9	8,999	**21**	20,980	**60**	59,820
10	9,998	**22**	21,970	**70**	69,810
11	11,000	**23**	22,980	**80**	79,710
12	12,000	**24**	23,980	**90**	89,670

Tabla 3. Validación de la configuración de frecuencias a través de un osciloscopio

También se han realizado pruebas con un fotodiodo para verificar que los LEDs también funcionan sin retardo y son capaces de reproducir las frecuencias establecidas, obteniendo idénticos resultados a los obtenidos con el osciloscopio.

Se verifica por tanto que con un dispositivo de estimulación basado en SSVEPs para una Interfaz Cerebro – Máquina realizado mediante un microprocesador Arduino Uno se obtienen resultados precisos mediante una configuración en tiempo real para el control de las frecuencias de los LEDs de estimulación.

4.2 Validación del dispositivo de estimulación a través de un EEG

Además de corroborar la precisión de las frecuencias en los LEDs, es necesario comprobar que la distancia a la que se han colocado, su luminosidad, color, opacidad, distancia del sujeto y demás parámetros favorecen la detección de estímulos SSVEP.

Se han realizado pruebas en un total de cinco sujetos, observando los resultados obtenidos y verificando que estos parámetros son correctos para hacer funcionar una Interfaz Cerebro – Máquina.

Por tanto, las dimensiones y parámetros elegidos a partir de las pruebas han sido los siguientes:

- **Distancia** entre elementos: 30 cm en horizontal y 35 cm en vertical.
- **Brillo** de los LEDs: 200 sobre el máximo de 255.
- Distancia y posición de los **sujetos**: de cara a maximizar los resultados, se situarán de frente al monitor, de manera que la distancia entre el usuario y la pantalla sea de 60 cm, con la espalda erguida y los brazos apoyados sobre la mesa, de manera relajada y evitando toda tensión muscular.
- **Color** de los LEDs: Se han utilizado tres colores diferentes, debido a la disponibilidad de los mismos: verde, blanco y rojo. Todos los LEDs poseen las mismas características.

Para comprobar la validez de estos parámetros, a modo de ejemplo se muestran los siguientes resultados en uno de los sujetos, donde la estimulación se realizaba fijando la atención en uno de los seis LEDs, mientras que en los otros cinco se emitían diferentes frecuencias.

En las dos primeras imagenes se muestran los resultados de pruebas realizadas cuando en los estímulos se muestran las frecuencias de 6, 7, 8, 9, 10 y 11 Hz. Para 6, 8 y Hz se considerarían estímulos detectados, ya que se cumplen tres condiciones:

- En primer lugar, el máximo de la función obtenida se encuentra exactamente a 6.00, 8.00 y 9.00 Hz, sin alterarse por la actividad cerebral, factores externos o interferencia con otras frecuencias.
- En segundo lugar, la amplitud de este pico con respecto al resto de la señal supera un umbral determinado (para este caso, 1dB), pudiendo decir con seguridad que el estímulo se ha detectado.
- Por último, la distancia con respecto al segundo máximo es lo suficientemente grande como para no afectar al resultado, manteniéndose por debajo del umbral.

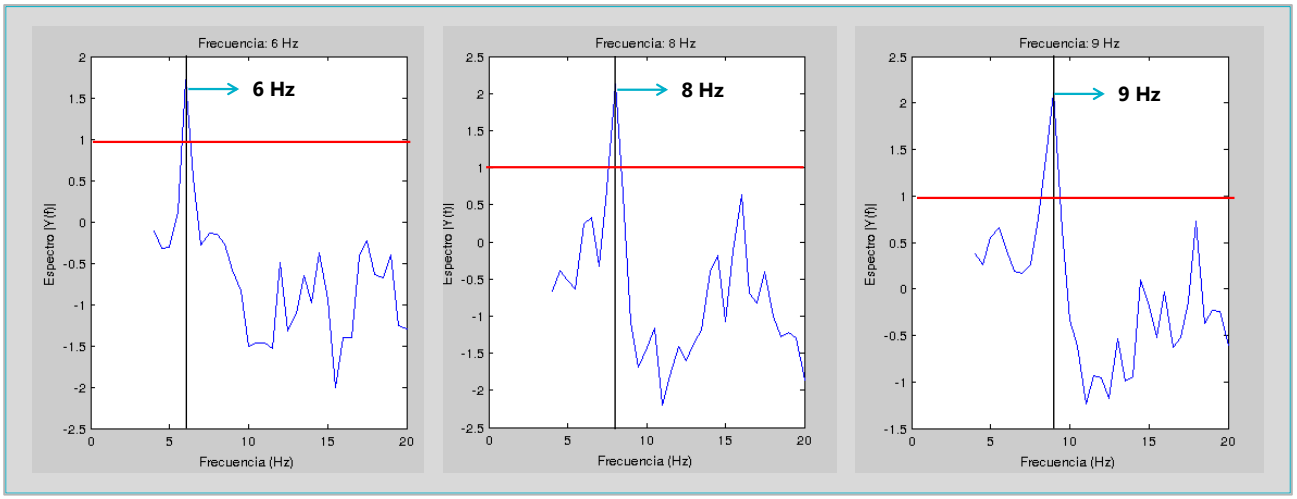

Figura 23. Resultados exitosos a frecuencias de 6, 8 y 9 Hz

Sin embargo, en la siguiente imagen se observa que no siempre se cumplen estos tres requisitos simultáneamente, ya que aunque en algunos casos sí que se obtiene un pico en frecuencia, la amplitud para considerarse válida es insuficiente y la distancia con el máximo secundario es demasiado estrecha.

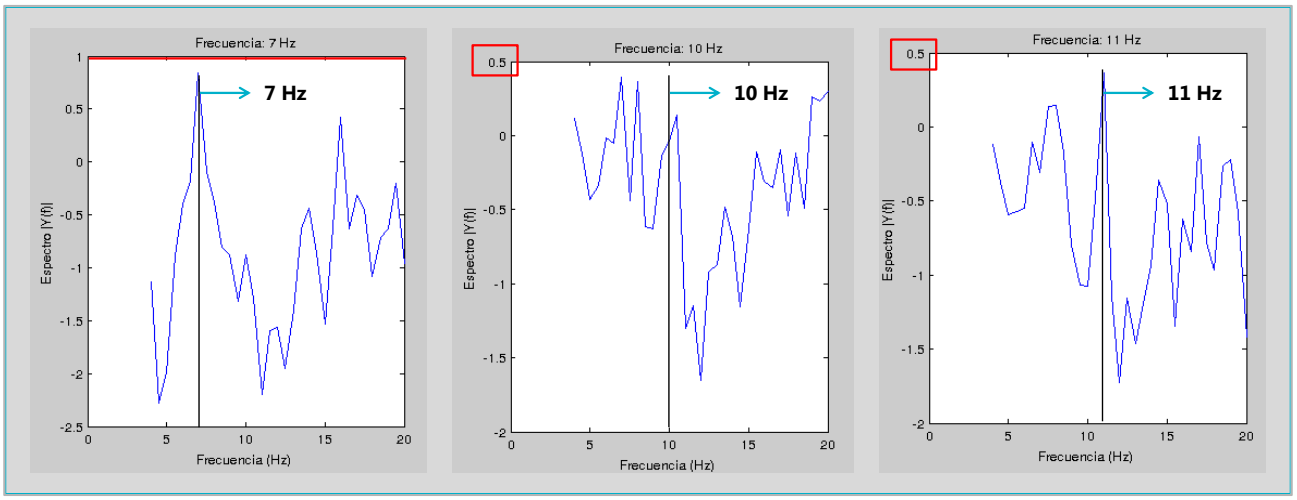

Figura 24. Resultados con amplitud insuficiente a 7, 10 y 11 Hz

Se muestra también un tercer caso, con frecuencias configuradas a 23, 24, 25, 26, 27 y 28 Hz oscilando simultáneamente. Para este caso, se observa un resultado que en ningún caso podrá considerarse positivo, ya que no se cumplen ninguno de los tres requisitos previamente mencionados. Cabe destacar sin embargo que en esta misma prueba sí que se obtuvieron resultados exitosos para 26, 27 y 28 Hz.

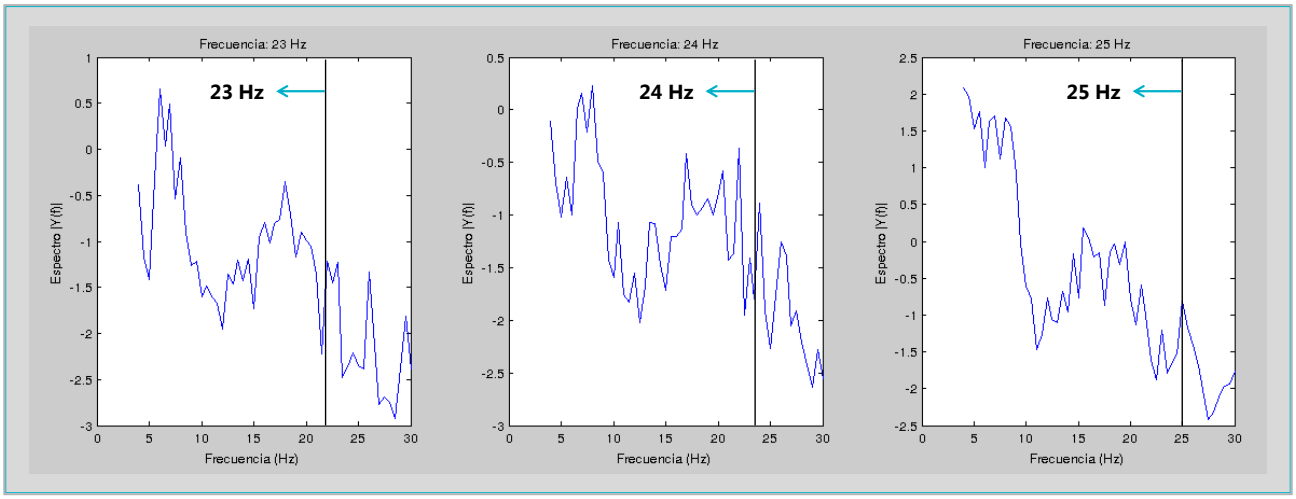

Figura 25. Resultados inválidos tanto en amplitud como en frecuencia a 23, 24 y 25 Hz

Queda demostrada por tanto la validez de este dispositivo de estimulación, siendo capaz de generar potenciales evocados SSVEP en la región occipital y obteniendo resultados en frecuencia que se corresponde con los emitidos desde el microprocesador.

Por último, cabe destacar la mejora observada en los resultados con respecto a las pruebas realizadas a lo largo del trabajo "Diseño de Interfaces Cerebro – Máquina y Hombre – Máquina controlados por señalización biológica" [2]. Esto se debe principalmente a la distancia entre estímulos, reduciendo considerablemente la interferencia entre los mismos. También ha influido la variación de las características de los LEDs y de los difusores, haciendo más sencilla la concentración y fijación en un estímulo concreto.

Por contraparte, la distancia entre LEDs establecida para este trabajo sólo es aplicable a aquellos BCIs que no tengan como objetivo la elaboración de un dispositivo portátil, ya que para tal fin serían recomendables las distancias del anterior trabajo, focalizado en dispositivos portátiles de bajo coste.

El número de resultados positivos también destaca con respecto a aquellos BCIs basados en dispositivos de estimulación SSVEP que se muestran por pantalla a través de un monitor, debido a la tasa de refresco y el efecto aliasing que se produce en los mismos [44], [46].

4.3 Verificación de la intercomunicación

Una vez construida la Interfaz Cerebro – Máquina, es necesario verificar en un primer lugar que la interconexión entre cada una de las partes se realiza satisfactoriamente y que cuando un cambio se produce en una de las partes, se realizan las actividades pertinentes en los siguientes.

La diferentes formas de intercomunicación quedan validadas de la siguiente manera:

Comunicación entre ...		¿Existe?	¿Cómo se ha validado?
GUI	**Microprocesador**	✓	Se han lanzado paquetes de información desde Qt y observado la respuesta en Arduino.
GUI	**Baseline**	✓	Al realizar una llamada al programa baseline desde la GUI, el casco Emotiv EPOC comienza a funcionar y graba un registro de baseline.
GUI	**Hilo QThread**	✓	Desde la GUI se configuran las frecuencias y desde el hilo se reciben correctamente. Validado mediante comandos qDebug.
GUI	**ejecucionEmotiv**	✓	Al realizar una llamada al programa ejecucionEmotiv desde la GUI, enviándole el número de usuario e id de Cola, el casco Emotiv EPOC comienza a funcionar y analizar señales.
ejecucionEmotiv	**Hilo QThread**	✓	A través de la cola de mensajes, el programa controlador del Emotiv EPOC informa al hilo de las frecuencias detectadas.
Hilo QThread	**GUI**	✓	Cuando se emite la señal "`Signal_Thread_Estimulo1()`", los estímulos mostrados por pantalla cambian en consecuencia.
Microprocesador	**GUI**	✗	El microprocesador escucha del puerto serie e interpreta los resultados, pero se trata de un dispositivo receptor y no emisor.
Baseline	**GUI**	✗	Aunque se ejecuten en paralelo, baseline no necesita enviar información a la interfaz.
ejecucionEmotiv	**GUI**	✗	ejecucionEmotiv envía la información al hilo, y este a la GUI, pero por las características de Qt la comunicación no es posible.
Hilo QThread	**ejecucionEmotiv**	✗	El hilo recibirá los mensajes de ejecucionEmotiv desde la cola, pero no habrá necesidad de enviar información adicional desde el hilo.

4.4 Pruebas con sujetos: Validación de un BCI Mixto

Para validar que la aplicación creada funciona como Interfaz Cerebro – Máquina, y que todas las partes funcionan de manera independiente y a la vez estando intercomunicadas, se han realizado pruebas en un total de siete sujetos, cuatro hombres y tres mujeres con edades comprendidas entre los 17 y los 29 años. Estos resultados son los obtenidos para la primera vez que cada usuario ha utilizado la interfaz. Se esperaría que tras varias sesiones los tiempos de respuesta y porcentajes de acierto aumentasen notablemente.

Las pruebas se han realizado siguiendo los siguientes pasos:

4.4.1 PASO 0 - Ajuste del casco Emotiv

Como preparación inicial, es necesario ajustar el casco de Emotiv EPOC al sujeto. Este paso puede abarcar entre 2 y 15 minutos, dependiendo de la dificultad de colocación del dispositivo.

En primer lugar, es necesario colocar los electrodos en el casco. Sobre ellos, se colocarán unas almohadillas empapadas en una solución salina para mejorar su conductividad, y se situarán los electrodos en las posiciones indicadas. Un programa proporcionado por Emotiv, TestBench, indicará por pantalla el nivel de conectividad de los electrodos, indicando en color verde aquellos electrodos donde el contacto con el cuero cabelludo es exitoso y en amarillo, naranja, rojo o negro aquellos en los que el contacto aún no es suficiente.

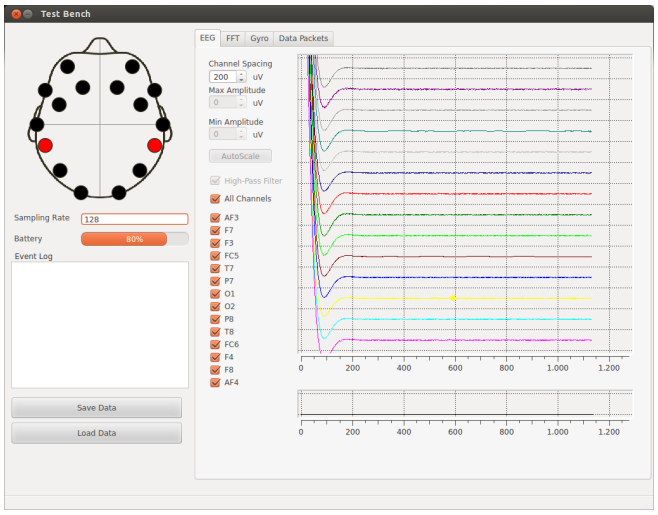

Figura 26. Captura de pantalla del programa de TestBench de ajuste de electrodos

Cabe mencionar que debido al uso a lo largo de los dos últimos años que se le ha dado al casco que se utiliza en el GNB, el desgaste ha provocado que la conductividad sea menor y que en la comunicación inalámbrica no se transmita el estado de cada electrodo, por lo que el resultado en el TestBench siempre aparece con todos los electrodos en color negro, y los dos de referencia de color rojo. Esto no significa realmente que no exista contacto con los electrodos, ya que en los registros se optiene una calidad aceptable de las señales y en las pruebas se obtienen buenos resultados, pero se conseguiría una mayor fiabilidad con un casco que informara de los umbrales de conductancia.

De cara a disminuir la tensión muscular y el exceso de actividad física y corporal se realizarán unos ejercicios de respiración y relajación previos al análisis. Las pruebas se realizarán siempre en el mismo laboratorio, únicamente con luz solar y con el laboratorio en silencio.

Se monitorizará también la frecuencia cardiaca a través de un dispositivo para verificar que los resultados se corresponden a un estado relajado y no se producen interferencias debido a alteraciones físicas.

Sujeto	Pulsaciones antes de la relajación	Pulsaciónes tras la relajación
1	82 ppm	69 ppm
2	78 ppm	64 ppm
3	70 ppm	65 ppm
4	75 ppm	64 ppm
5	85 ppm	73 ppm
6	79 ppm	70 ppm
7	92 ppm	80 ppm

Tabla 4. Frecuencia cardiaca en los sujetos.

4.4.2 PASO 1 - Selección de mejores frecuencias para el sujeto

Para un análisis preciso de las mejores frecuencias que corresponden a cada usuario, se realizará de un barrido de 6 a 23 Hz y se analizarán las frecuencias resultantes de manera tanto gráfica como numérica. Mediante este barrido se adaptan las frecuencias de estimulación a cada usuario de forma personalizada.

Para ello, se realizarán tres análisis consecutivos: de 6 a 11 Hz, se mostrarán las frecuencias de manera simultánea y se tomará una muestra de 20 segundos para cada una de las frecuencias. Para el segundo análisis se repetirá este mismo procedimiento pero las frecuencias de estimulación serán de 12 a 17 Hz, y en el último análisis cambiarán a un barrido de 18 a 23 Hz. Es importante por tanto destacar que estos resultados se han obtenido cuando **se producen interferencias de otras frecuencias en el campo visual**.

En la siguiente tabla se muestran los resultados obtenidos para cada frecuencia. En caso de ser seleccionada como válida, se indicará la amplitud de la señal obtenida. Si en el espectro no se detecta un máximo de amplitud por encima del umbral y distancia con respecto al máximo secundario suficiente, la frecuencia no será válida.

Se muestran en color rojo los resultados que han descartado la frecuencia como válida, y con fondo verde aquellas que cumplen y se podrán utilizar en las siguientes fases. Adicionalmente, dentro de las que se han seleccionado como válidas se destacan en negrita aquellas cuyos resultados superan notablemente los umbrales establecidos.

Freq	Parámetro	S1	S2	S3	S4	S5	S6
Umbral de Amplitud		**0,7 dB**	**0,7 dB**	**-2 dB**	**0,7 dB**	**0 dB**	**0,6 dB**
6 Hz	Máx	6,0 Hz	12,0 Hz	**18,0 Hz**	**6,0 Hz**	6,0 Hz	6,0 Hz
	Amplitud	1,7 dB	1,2 dB	-	**2,9 dB**	0,7 dB	1,2 dB
	Dist. 2º Max	2,0 dB	0,2 dB	-	**2,5 dB**	1,16 dB	0,8 dB
7 Hz	Máx	7,0 Hz	14,0 Hz	**25,3 Hz**	7,0 Hz	**8,5 Hz**	7,0 Hz
	Amplitud	0,8 dB	0,7 dB	-	0,8 dB	-	0,7 dB
	Dist. 2º Max	0,4 dB	0,5 dB	-	0,7 dB	-	1,0 dB
8 Hz	Máx	8,0 Hz	**6,5 Hz**	**27,0 Hz**	8,0 Hz	**17,5 Hz**	**8,0 Hz**
	Amplitud	2,1 dB	**-**	-	2,5 dB	-	**1,9 dB**
	Dist. 2º Max	2,2 Hz	**-**	-	2,5 dB	-	**1,5 dB**
9 Hz	Máx	9,0 Hz	**14,5 Hz**	**23,5 Hz**	**9,0 Hz**	9 Hz	9,0 Hz
	Amplitud	2,1 dB	**-**	-	**2,4 dB**	0,4 dB	**0,4 dB**
	Dist. 2º Max	2,0 dB	**-**	-	**3,0 dB**	**0,0 dB**	**0,1 dB**
10 Hz	Máx	**7,0 Hz**	**6,0 Hz**	**28,5 Hz**	20,0 Hz	**8,5 Hz**	**8,0 Hz**
	Amplitud	-	**-**	-	0,8 dB	-	-
	Dist. 2º Max	-	**-**	-	**0,2 dB**	-	-
11 Hz	Máx	11,0 Hz	**12,0 Hz**	**5,5 Hz**	22,0 Hz	11,0 Hz	11,0 Hz
	Amplitud	**0,4 Hz**	**-**	-	**0,4 dB**	**-0,2 dB**	0,9 dB
	Dist. 2º Max	**0,4 Hz**	**-**	-	**0,2 dB**	0,1 dB	1,0 dB
12 Hz	Máx	12,0 Hz	**25,5 Hz**	12,0 Hz	**5,0 Hz**	**8,5 Hz**	**5,5 Hz**
	Amplitud	**0,6 dB**	**-**	-1,5 dB	-	-	-
	Dist. 2º Max	1,4 dB	**-**	0,5 dB	-	-	-
13 Hz	Máx	**6,4 Hz**	13,0 Hz	**14,0 Hz**	**26,0 Hz**	**11,5 Hz**	13,0 Hz
	Amplitud	-	1,1 dB	-	**2,5 dB**	-	1,2 dB
	Dist. 2º Max	-	**0,0 dB**	-	**1,1 dB**	-	0,6 dB
14 Hz	Máx	14,0 Hz	**12,0 Hz**	14,0 Hz	**28,0 Hz**	14,0 Hz	28,0 Hz
	Amplitud	**0,2 dB**	-	-1,9 dB	**2,1 dB**	**-1,0 dB**	**-0,5 dB**

	Dist. 2º Max	0,8 dB	-	0,5 dB	**2,0 dB**	0,3 dB	0,6 dB
15 Hz	Máx	15,0 Hz	**18,5 Hz**	**15,0 Hz**	**15,0 Hz**	**14,5 Hz**	**23,5 Hz**
	Amplitud	1,2 dB	**-**	**-0,4 dB**	**2,1 dB**	-	-
	Dist. 2º Max	1,9 dB	**-**	**1,3 dB**	**2,2 dB**	-	-
16 Hz	Máx	16,0 Hz	**18,2 Hz**	16,0 Hz	**16,0 Hz**	16,0 Hz	**11,0 Hz**
	Amplitud	1,1 dB	**-**	-1,4 dB	**2,5 dB**	**-0,74 dB**	**-**
	Dist. 2º Max	1,7 dB	**-**	0,9 dB	**2,5 dB**	**0,2 dB**	**-**
17 Hz	Máx	**23,0 Hz**	**12,0 Hz**	17,0 Hz	**17,0 Hz**	**5,0 Hz**	**5,5 Hz**
	Amplitud	-	**-**	-1,4 dB	**3,1 dB**	-	-
	Dist. 2º Max	-	**-**	0,7 dB	**3,0 dB**	-	-
18 Hz	Máx	18,0 Hz	**5,5 Hz**	**18,0 Hz**	**18,0 Hz**	18 Hz	**20,0 Hz**
	Amplitud	**-0,2 dB**	**-**	**-0,6 dB**	**2,7 dB**	0,6 dB	-
	Dist. 2º Max	0,6 dB	**-**	**1,4 dB**	**2,8 dB**	-0,8 dB	-
19 Hz	Máx	**14,4 Hz**	19,0 Hz	**18,5 Hz**	**28,0 Hz**	**12,5 Hz**	19,0 Hz
	Amplitud	-	**0,3 dB**	-	-	-	**0,0 dB**
	Dist. 2º Max	-	**0,0 dB**	-	-	-	**0,1 dB**
20 Hz	Máx	**14,5 Hz**	20,0 Hz	20,0 Hz	**20,0 Hz**	**4,0 Hz**	**21,3 Hz**
	Amplitud	-	1,0 dB	-1,4 dB	**2,4 dB**	-	-
	Dist. 2º Max	-	1,0 dB	0,6 dB	**2,3 dB**	-	-
21 Hz	Máx	21,0 Hz	**11,0 Hz**	**21,0 Hz**	**21,0 Hz**	**5,0 Hz**	21,0 Hz
	Amplitud	**-0,3 dB**	-	**-0,9 dB**	**2,4 dB**	-	0,8 dB
	Dist. 2º Max	**0,3 dB**	-	**1,1 dB**	**3,1 dB**	-	1,2 dB
22 Hz	Máx	22,0 Hz	**11,0 Hz**	22, 0 Hz	**22,0 Hz**	**8,5 Hz**	22,0 Hz
	Amplitud	**-0,1 Hz**	**-**	-1,0 dB	**2,5 dB**	-	1,2 dB
	Dist. 2º Max	**0,5 dB**	**-**	-0,6 dB	**1,8 dB**	-	1,1 dB
23 Hz	Máx	23,0 Hz	**11,0 Hz**	23,0 Hz	**22,0 Hz**	**14,0 Hz**	23,0 Hz
	Amplitud	0,1 dB	**-**	-1,3 dB	-	-	0,9 dB
	Dist. 2º Max	2,3 dB	**-**	-0,7 dB	-	-	0,5 dB

TOTAL	7 FREQS	3 FREQS	10 FREQS	13 FREQS	2 FREQS	8 FREQS

Tabla 5. Resultados barrido en frecuencia

De estos resultados, cabe destacar lo siguiente:

- En el usuario 1 se detectaron un total de siete frecuencias que cumplían con los niveles de aceptación teniendo en cuenta que se trata de un análisis donde intervienen seis luces a diferentes frecuencias en el campo visual. **Los resultados son exitosos** y se puede continuar con la siguiente fase.

- En el caso del usuario 2 y el usuario 5, no se detectaron frecuencias válidas suficientes, por lo que se utilizarán **frecuencias prefijadas** para probar la GUI, siendo conscientes de la baja probabilidad de acierto que existirá.

- Para el usuario 3 y 5 se han registrado una **gran cantidad de frecuencias válidas,** siendo un total de 10 en el primer caso y 8 en el segundo, de las cuales se han escogido las mejores.

- En el usuario 4 los resultados han **superado las expectativas** esperadas para este casco, con señales de una gran amplitud con respecto a la media en un total de 11 frecuencias, y un total de 13 frecuencias exitosas.

- Para el sujeto 7 no se realizaron estas pruebas y **no se continuó con el procedimiento** establecido para el resto de experimentos, debido a que al realizar el ajuste del casco Emotiv EPOC se observó que las señales presentaban una distorsión muy por encima de los umbrales considerados normales. Se probó a repetir el ajuste del casco, y los resultados fueron idénticos. Se desconoce la causa que puede provocar este comportamiento.

Estos resultados superan notablemente los obtenidos en trabajos anteriores, tanto en aquellos en los que se ha utilizado estimulación por pantalla [44], [46] como aquellos construidos mediante hardware [2].

4.4.3 PASO 2 - Medición de tiempo de respuesta a estímulos y porcentaje de aciertos

En este ejercicio, se medirá el tiempo de respuesta, así como el número de fallos obtenidos en el juego "12 Luces". Para estas pruebas se ha reducido el número de luces a detectar a 5, de cara a agilizar el proceso de análisis.

Cabe destacar que, debido a las características del algoritmo y su enventanado de longitud de dos segundos, y el criterio de consideración de estímulo detectado, que se activa cuando se reciben tres resultados consecutivos iguales, el **tiempo mínimo de detección será de 6 segundos por estímulo.**

Sujeto	Número de Fallos	Duración	Tiempo medio por estímulo	% Aciertos
1	**1 fallo**	**86 segundos**	**14,2 segundos**	**83,3%**
2	6 fallos	456 segundos	41,4 segundos	**45,5%**
3	9 fallos	212 segundos	15,1 segundos	**35,7%**
4	2 fallos	126 segundos	18,1 segundos	71,4%
5	2 fallos	164 segundos	23,4 segundos	71,4%
6	**0 fallos**	**102 segundos**	**20,4 segundos**	**100%**

Tabla 6. Medición de tiempo de respuesta ante estímulos.

En este caso, se puede observar que, aunque el sujeto 4 fue el que mejores resultados proporcionó en el análisis de frecuencias, en este caso son los sujetos 1 y 6 en los que se obtienen los mejores resultados.

En el caso del sujeto 2 no se obtienen resultados concluyentes debido al tiempo de detección de estímulos y al número de fallos.

El sujeto 3, aunque obtiene una tasa de acierto muy baja, reduce considerablemente el tiempo de detección de estímulos llegando a 15,1 segundos de media.

Los resultados, salvo el sujeto 2, se pueden considerar como aceptables, teniendo un amplio recorrido de mejora en mejores condiciones del casco y tras un entrenamiento. Cabe destacar el porcentaje de aciertos por encima de 80% para el sujeto 1 y el porcentaje del 100% en el sujeto 2.

4.4.4 PASO 3 - Utilización como dispositivo de comunicación

Siendo la comunicación mediante este BCI el objetivo principal de su implementación, se ha querido comprobar su utilización a través de la aplicación "Easy Life", desarrollada a partir de las indicaciones recibidas por el centro Cottolengo de Madrid para personas discapacitadas.

Para ello, se solicitará a los usuarios que, a partir de su concentración, seleccionen en primer lugar la aplicación "Easy Life" desde el menú principal. Una vez ahí, y viendo el menú desplegable, se solicitará que piensen en aquella instrucción que quieran comunicar y sin decir nada se muevan hasta el estímulo correspondiente, verificando después si se trataba de la opción deseada o no.

- El **sujeto 1,** tras visualizar las opciones disponibles, comunicó su intención de seleccionar el mensaje "tengo calor", para lo cual fueron necesarias tres iteraciones de la interfaz (Easy Life // Sensaciones // Calor). Esta prueba se realizó con éxito y la opción resultante fue la que el usuario seleccionó.

- En el **sujeto 2** no se pudo realizar esta prueba ya que el casco no detectaba ningún estímulo para cualquier opción que desease seleccionar.

- Para el caso del **sujeto 3**, se realizaron dos pruebas en las que se comunicaron los mensajes "Sentimientos // Alegría" y "Acciones // Encender TV".

- En el **sujeto 4** se pudo comunicar el mensaje "Alegría" durante tres veces consecutivas. Este resultado es tremendamente positivo, ya que para ello es necesario realizar tres rondas de tres iteraciones y se realizó sin ningún fallo.

- Para los **sujetos 5 y 6,** de nuevo se detectó un mensaje recibido para cada uno.

Esta prueba valida la utilización del dispositivo como interfaz de comunicación en la que se pueden transmitir mensajes con una tasa de acierto elevada.

4.4.5 PASO 4 - Encuesta de percepción del usuario.

Por último, tras la sesión de experimentos, se solicitó a los sujetos que realizasen una encuesta en la que comentasen los principales problemas y detalles de hubiesen percibido durante la utilización de la interfaz. Los resultados fueron los siguientes:

Parámetro	Puntuación de los Sujetos					
	1	**2**	**3**	**4**	**5**	**6**
Comodidad del casco	8	6	8	10	10	8
Comodidad de las luces	4	5	6	0	5	8
Facilidad de uso de la GUI	9	10	8	10	10	10
Valoración de los colores de cada LED	Color blanco resulta molesto	Indiferente	Todos cómodos	Color verde molesto. Color blanco muy cómodo	Amarillas únicas cómodas	Indiferente

Tabla 7. Encuesta de percepción tras realizar las pruebas.

Como comentarios adicionales, se indicó lo siguiente:

- El sujeto 1 indicó que, a medida que las frecuencias aumentaban, aumentaba también la capacidad de concentración en las mismas y disminuía la interferencia del resto de luces. Otros sujetos coincidieron en esta valoración.

- El sujeto 2 comentó la incomodidad que este casco y la posición suponían para su cuello, provocándole dolor en cuello y espalda que lo llevaba a cambiar de posición constantemente. Además indicó su cansancio y dificultad para concentrarse en los estímulos.

- Todos los sujetos menos el 2 indicaron que el casco les resultaba cómodo una vez ajustado, y que no suponía una molestia.

- El sujeto 5 destacó que las luces le provocaban fatiga e incomodidad, y el usuario 4 que le resultaban muy molestas. El sujeto 1 y 3 no presentan indicaciones de incomodidad provocadas por las luces.

4.4.6 Consideraciones

- Se han conseguido resultados mejores a los obtenidos mediante una estimulación por pantalla, debido a la distancia entre los LEDs y a no tener limitación debido a la tasa de refresco [2], [44], [46].

- Observando los resultados del sujeto 2, queda patente la existencia del BCI Illiteracy, fenómeno que remarca el hecho de que no todas las personas son válidas para la utilización de un dispositivo de estas características, debido a la fisionomía y condiciones biológicas de cada persona.

- Estos resultados se han obtenido en circunstancias que podrían dar un recorrido de mejora notable. En primer lugar, tras repetidas utilizaciones del BCI se ha demostrado que se obtienen mejores resultados, ya que se requiere de un proceso de aprendizaje. Además, la utilización de electrodos y esponjas de contacto nuevas, así como un casco Emotiv EPOC plenamente operativo y en el que se hubiera podido realizar el ajuste de la posición de los electros mejoraría notablemente la precisión de las señales.

Queda demostrada por tanto la funcionalidad completa de esta Interfaz Cerebro – Máquina con señalización mixta, validando su comportamiento en tiempo real, flexibilidad, facilidad de utilización y pudiendo utilizarse como medio de comunicación.

CAPÍTULO 5. CONCLUSIONES Y ESTUDIO FUTURO

A lo largo de este trabajo se ha analizado en profundidad la situación actual de la investigación relacionada con las Interfaces Cerebro – Máquina, y se ha diseñado y desarrollado una aplicación en tiempo real con señalización mixta, proporcionando resultados que la validan.

Este Trabajo de Fin de Máster comenzaba enumerando una serie de objetivos que se pretendía cumplir durante el planteamiento del mismo. Por tanto, resulta conveniente volver a analizar los mismos para verificar si el trabajo ha estado alineado con ellos:

- *"El objetivo de este Trabajo de Fin de Máster es la creación de una Interfaz Cerebro – Máquina en tiempo real que sea universal y flexible, tanto con respecto a sus aplicaciones como sistemas de estimulación basados en SSVEP o métodos de adquisición."*

  - Queda validada la implementación **en tiempo real** de una BCI compuesta por un sistema de estimulación, una interfaz gráfica y un sistema de adquisición y procesado de señal de manera que cada elemento está **intercomunicado** pero es **independiente**, dotándolo de gran **flexibilidad** a la hora de intercambiar componentes para implementar distintas funcionalidades.

- *"Se implementará para ello una interfaz gráfica de usuario que permita realizar una señalización mixta, es decir, que habilite la comunicación a través de una señalización por pantalla y otra simultánea mediante un hardware externo, siendo capaces de intercomunicarse entre sí."*

- Se ha desarrollado y demostrado la validez de **una Interfaz Gráfica de Usuario** que se comunica con un procesador externo para habilitar la **señalización** de una manera **mixta**, siendo la GUI la que proporciona información adicional al usuario para la interpretación de la codificación de estímulos y además emite instrucciones de configuración al microprocesador Arduino que actúa como hardware externo para el control de los LEDs de estimulación.

- *"Se crearán nuevas funcionalidades que el usuario podrá seleccionar, que permitirán utilizar este BCI como sistema de comunicación."*

  - Se ha creado un menú con diferentes aplicaciones, incluyendo un deletreador, un sistema de comunicación de respuestas rápidas y la herramienta Easy Life con mensajes básicos que **habilitan la comunicación** de manera que se pueda realizar única y exclusivamente **a través del BCI.**

- *"Por último, se busca que esta interfaz realice una selección de estímulos adaptada a cada sujeto, por lo que se trabajará en la búsqueda de aquellas frecuencias que maximicen las respuestas de los usuarios. "*

- Se ha utilizado una selección de estímulos adaptada a cada sujeto, demostrando que con este ajuste se puede utilizar la Interfaz con mejores resultados y almacenando en un fichero las mejores frecuencias para cada sujeto.

En definitiva, se ha diseñado, implementado y validado una Interfaz Cerebro – Máquina utilizando una señalización mixta mediante hardware y software, demostrando obtener mejores resultados que en otros diseños de BCI. Se ha integrado dentro de una GUI que se encarga de gestionar todos los procesos adyacentes, y se han realizado pruebas de validación de la misma, así como comprobaciones en seis sujetos donde se han conseguido frecuencias cuya amplitud, precisión y distancia con respecto a máximos secundarios eran satisfactorias en las pruebas con los voluntarios. Además, el número de aciertos y tiempo de respuesta obtenido en las pruebas refleja un porcentaje elevado de resultados exitosos, validando la funcionalidad de la interfaz.

Es importante destacar que los resultados se han obtenido con un casco de bajo coste que solo contiene dos electrodos en la zona occipital y no en las mejores posiciones para detectar SSVEPs. Esto recalca aún más la eficacia del sistema de estimulación y adaptación al usuario, así como la señalización mixta que favorece la comprensión y la concentración del usuario.

Como líneas de estudio futuro, como primer ámbito principal de actuación se realizarán pruebas con un número más elevado de usuarios, de manera que se puedan extraer resultados cuantitativos con un análisis estadístico riguraso. La interfaz desarrollada es fácilmente adaptable a cualquier casco de EEG, simplemte sustituyendo el módulo de adquisición de la señal. Un casco con mayor resolución y en mejor estado que el casco de Emotiv EPOC empleado redundaría en unos mejores resultados.

Se pretende también realizar estudios con personas que sufren algún tipo de discapacidad física o motora pero que su capacidad cognitiva no se encuentre alterada, de manera que se pueda validar la interfaz como medio de comunicación por sí mismo.

Además, se trabajará durante los meses de Julio y Agosto de 2016 en elaborar una publicación que refleje tanto el trabajo que se ha realizado como los resultados obtenidos.

De cara a mejorar la Interfaz Cerebro – Máquina, se propone trabajar en el algoritmo de detección de estímulos, incorporando también información sobre la fase del estímulo [25], [47] logrando reducir el tiempo de procesamiento y obtención de resultados, buscando lograr tiempos de hasta 1 segundo por estímulo [48], utilizando para ello registros de EEG que aporten una mayor precisión y velocidad de procesamiento. Por otro lado se puede incorporar un algoritmo de detección de la mejor referencia en el registro de EEG que se está desarrollando en el GNB [49].

También se podrá adaptar la configuración de la GUI a las necesidades de cada sujeto, buscando incrementar el número de funcionalidades y mejorando el diseño actual.

CAPÍTULO 6. REFERENCIAS

En este capítulo se enumeran las diferentes fuentes de información que se han utilizado para el estudio, elaboración y diseño de este Trabajo.

[1] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan, "Brain-computer interfaces for communication and control," *Clin. Neurophysiol.*, vol. 113, no. 6, pp. 767–791, 2002.

[2] A. I. Sotomayor Romillo, "Diseño de Interfaces Cerebro - Máquina y Hombre - Máquina controlados por señalización biológica," p. 106, 2013.

[3] N. E. and da S. F. L. (2004), "Electroencephalography: Basic Principles, Clinical Applications and Related Fields." .

[4] G. R. Müller-Putz, R. Scherer, C. Brauneis, and G. Pfurtscheller, "Steady-state visual evoked potential (SSVEP)-based communication: impact of harmonic frequency components.," *J. Neural Eng.*, vol. 2, no. 4, pp. 123–130, 2005.

[5] C. S. Herrmann, "Human EEG responses to 1-100 Hz flicker: resonance phenomena in visual cortex and their potential correlation to cognitive phenomena," *Exp. Brain Res.*, vol. 137, no. 3–4, pp. 346–353, 2001.

[6] B. Blankertz, C. Sannelli, S. Halder, E. M. Hammer, A. Kübler, K.-R. Müller, G. Curio, and T. Dickhaus, "Neurophysiological predictor of SMR-based BCI performance.," *Neuroimage*, vol. 51, no. 4, pp. 1303–1309, 2010.

[7] C. Vidaurre and B. Blankertz, "Towards a cure for BCI illiteracy.," *Brain Topogr.*, vol. 23, no. 2, pp. 194–198, 2010.

[8] "http://www.arduino.cc." .

[9] M. Byczuk, P. Poryzał a, and a. Materka, "On diversity within operators' EEG responses to LED-produced alternate stimulus in SSVEP BCI," *Bull. Polish Acad. Sci. Tech. Sci.*, vol. 60, no. 3, pp. 447–453, Jan. 2012.

[10] "https://emotiv.com/epoc.php." .

[11] "https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario." .

[12] "http://www.qt.io/." .

[13] F. Galan, M. Nuttin, E. Lew, P. W. Ferrez, G. Vanacker, J. Philips, and J. del R. Millan, "A brain-actuated wheelchair: Asynchronous and non-invasive Brain-computer interfaces for continuous control of robots," *Clin. Neurophysiol.*, vol. 119, no. 9, pp. 2159–2169, 2008.

[14] "http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html." .

[15] C. Guger, B. Z. Allison, B. Großwindhager, R. Prückl, C. Hintermüller, C. Kapeller, M. Bruckner, G. Krausz, and G. Edlinger, "How many people could use an SSVEP BCI?," *Front. Neuroprosthetics*, vol. 6, p. 169, 2012.

[16] L. F. Nicolas-Alonso and J. Gomez-Gil, "Brain Computer Interfaces, a Review," *Sensors*, vol. 12, no. 2, pp. 1211–1279, 2012.

[17] P. F. Diez, S. M. Torres Müller, V. A. Mut, E. Laciar, E. Avila, T. F. Bastos-Filho, and M. Sarcinelli-Filho, "Commanding a robotic wheelchair with a high-frequency steady-state visual evoked potential based brain-computer interface.," *Med. Eng. Phys.*, vol. 35, no. 8, pp. 1155–1164, 2013.

[18] O. Friman, I. Volosyak, and A. Graser, "Multiple Channel Detection of Steady-State Visual Evoked Potentials for Brain-Computer Interfaces," *IEEE Trans. Biomed. Eng.*, vol. 54, no. 4, pp. 742–750, 2007.

[19] S. Ruiz, K. Buyukturkoglu, M. Rana, N. Birbaumer, and R. Sitaram, "Real-time fMRI brain computer interfaces: self-regulation of single brain regions to networks.," *Biol. Psychol.*, vol. 95, pp. 4–20, 2014.

[20] O. Jensen, A. Bahramisharif, R. Oostenveld, S. Klanke, A. Hadjipapas, Y. O. Okazaki, and M. A. J. van Gerven, "Using brain-computer interfaces and brain-state dependent stimulation as tools in cognitive neuroscience.," *Front. Psychol.*, vol. 2, p. 100, 2011.

[21] G. Schalk and E. C. Leuthardt, "Brain-computer interfaces using electrocorticographic signals," *IEEE Rev. Biomed. Eng.*, vol. 4, pp. 140–154, 2011.

[22] J. Sulzer, S. Haller, F. Scharnowski, N. Weiskopf, N. Birbaumer, M. L. Blefari, A. B. Bruehl, L. G. Cohen, R. C. DeCharms, R. Gassert, R. Goebel, U. Herwig, S. LaConte, D. Linden, A. Luft, E. Seifritz, and R. Sitaram, "Real-time fMRI neurofeedback: progress and challenges.," *Neuroimage*, vol. 76, pp. 386–399, 2013.

[23] F. Cincotti, D. Mattia, F. Aloise, S. Bufalari, G. Schalk, G. Oriolo, A. Cherubini, M. G. Marciani, and F. Babiloni, "Non-invasive brain-computer interface system: towards its application as assistive technology.," *Brain Res. Bull.*, vol. 75, no. 6, pp. 796–803, 2008.

[24] S. Halder, D. Agorastos, R. Veit, E. M. Hammer, S. Lee, B. Varkuti, M. Bogdan, W. Rosenstiel, N. Birbaumer, and A. Kübler, "Neural mechanisms of brain-computer interface control.," *Neuroimage*, vol. 55, no. 4, pp. 1779–1790, 2011.

[25] C. Jia, X. Gao, B. Hong, and S. Gao, "Frequency and phase mixed coding in SSVEP-based brain--computer interface.," *IEEE Trans Biomed Eng*, vol. 58, no. 1, pp. 200–206, 2011.

[26] R. Fazel-Rezai, B. Z. Allison, C. Guger, E. W. Sellers, S. C. Kleih, and A. Kübler, "P300 brain computer interface: current challenges and emerging trends.," *Front. Neuroeng.*, vol. 5, no. July, p. 14, 2012.

[27] C. Brunner, B. Z. Allison, C. Altstätter, and C. Neuper, "A comparison of three brain-computer interfaces based on event-related desynchronization, steady state visual evoked potentials, or a hybrid approach using both signals.," *J. Neural Eng.*, vol. 8, no. 2, p. 25010, 2011.

[28] M. Kamrunnahar, N. S. Dias, S. J. Schiff, and B. J. Gluckman, "Model-based responses and features in Brain Computer Interfaces.," *Conf Proc IEEE Eng Med Biol Soc*, vol. 2008, pp. 4482–4485, 2008.

[29] J.-H. Lim, H.-J. Hwang, C.-H. Han, K.-Y. Jung, and C.-H. Im, "Classification of binary intentions for individuals with impaired oculomotor function: 'eyes-closed' SSVEP-based brain-computer interface (BCI).," *J. Neural Eng.*, vol. 10, no. 2, p. 26021, 2013.

[30] M. van Gerven, J. Farquhar, R. Schaefer, R. Vlek, J. Geuze, A. Nijholt, N. Ramsey, P. Haselager,

L. Vuurpijl, S. Gielen, and P. Desain, "The brain-computer interface cycle," *J. Neural Eng.*, vol. 6, no. 4, p. 41001, 2009.

[31] E. Yin, Z. Zhou, J. Jiang, F. Chen, Y. Liu, and D. Hu, "A novel hybrid BCI speller based on the incorporation of SSVEP into the P300 paradigm.," *J. Neural Eng.*, vol. 10, no. 2, p. 26012, 2013.

[32] N. Birbaumer, "Breaking the silence: brain-computer interfaces (BCI) for communication and motor control," *Psychophysiology*, vol. 43, no. 6, pp. 517–532, 2006.

[33] G. R. Müller-Putz and G. Pfurtscheller, "Control of an electrical prosthesis with an SSVEP-based BCI.," *IEEE Trans. Biomed. Eng.*, vol. 55, no. 1, pp. 361–364, 2008.

[34] F. Di Russo, S. Pitzalis, T. Aprile, G. Spitoni, F. Patria, A. Stella, D. Spinelli, and S. A. Hillyard, "Spatiotemporal analysis of the cortical sources of the steady-state visual evoked potential.," *Hum. Brain Mapp.*, vol. 28, no. 4, pp. 323–334, 2007.

[35] I. Volosyak, H. Cecotti, and A. Gräser, "Impact of Frequency Selection on LCD Screens for SSVEP Based Brain-Computer Interfaces," in *Proc. 4th Int. IEEE/EMBS Conference on Neural Engineering NER 09*, 2009, pp. 706–713.

[36] I. Volosyak, "SSVEP-based Bremen-BCI interface--boosting information transfer rates.," *J. Neural Eng.*, vol. 8, no. 3, p. 36020, 2011.

[37] J. Fernandez-Vargas, H. U. Pfaff, F. B. Rodriguez, and P. Varona, "Assisted closed-loop optimization of SSVEP-BCI efficiency," *Front. Neural Circuits*, vol. 7, p. Article 27, 2013.

[38] Y. Zhang, P. Xu, T. Liu, J. Hu, R. Zhang, and D. Yao, "Multiple Frequencies Sequential Coding for SSVEP-Based Brain-Computer Interface.," *PLoS One*, vol. 7, no. 3, p. e29519, 2012.

[39] Z. Wu, Y. Lai, Y. Xia, D. Wu, and D. Yao, "Stimulator selection in SSVEP-based BCI.," *Med. Eng. Phys.*, vol. 30, no. 8, pp. 1079–1088, 2008.

[40] "https://es.mathworks.com/EEGLAB." .

[41] "http://openvibe.inria.fr/." .

[42] "http://openbci.com/." .

[43] D. Approval, A. Rev, and D. No, "Specification LW500AM," vol. 24, no. December, 2010.

[44] Á. Morán García, "Diseño de Interfaces Cerebro - Máquina controlados mediante registros de EEG," 2015.

[45] "https://wiki.qt.io/Qt_for_Beginners." .

[46] A. A. Santamaría, "Adaptación de estímulos en interfaces cerebro-máquina que utilizan potenciales visuales evocados," 2016.

[47] M. Abu-Alqumsan and A. Peer, "Advancing the detection of steady-state visual evoked potentials in brain–computer interfaces," *J. Neural Eng.*, vol. 13, no. 3, p. 36005, 2016.

[48] X. Chen, Y. Wang, M. Nakanishi, X. Gao, T.-P. Jung, and S. Gao, "High-speed spelling with a noninvasive brain–computer interface," *Proc. Natl. Acad. Sci.*, vol. 112, no. 44, pp. 1–10, 2015.

[49] F. Gembler, P. Stawicki, and I. Volosyak, "Autonomous Parameter Adjustment for SSVEP-

Based BCIs with a Novel BCI Wizard," vol. 9, no. December, pp. 1–12, 2015.

ANEXOS

En este apartado se proporciona el código software que se ha implementado para el desarrollo de esta Interfaz Cerebro – Máquina con señalización mixta.

Anexo I: Manual de Usuario

De cara a facilitar la utilización posterior de esta interfaz, en este Anexo se facilita un manual de usuario con los pasos a seguir para poder utilizar esta interfaz.

- PASO 1. Si se va a utilizar el BCI en un ordenador diferente al que se ha utilizado ahora, en el GNB, se deberán colocar los LEDs alrededor del marco de un monitor de dimensiones similares a las utilizadas en las pruebas (60 x 35 cm), y verificar que en dicho ordenador se encuentran instaladas las librerías de ejecución del casco Emotiv EPOC, así como Qt.

- PASO 2. Verificar que el microprocesador Arduino Uno se encuentra conectado a la red. El programa de utilización se encuentra ya compilado y subido a la placa.

- PASO 3. Humedecer las almohadillas de los electrodos, colocar los electrodos en el casco Emotiv y colocar en el cuero cabelludo de acuerdo a las posiciones establecidas por el fabricante [10]. Hacer uso del programa TestBench facilitado por Emotiv para ajustar los electrodos.

- PASO 4. Abrir el programa "terminal.pro" en Qt, y ejecutar.

- PASO 5. Pulsar el botón "Conectar" para realizar la conexión con Arduino desde la GUI (ver *Figura 14* Captura de pantalla de la esquina superior izquierda de la GUI).

- PASO 6. En la GUI se irán mostrando los menús detallados a lo largo del CAPÍTULO 3 Diseño y Desarrollo).

Anexo II: Código Implementación Arduino

A continuación se muestra el código de implementación utilizado para establecer las frecuencias de cada uno de los estímulos en los LEDs. En este código se proporciona también visibilidad sobre el proceso de establecimiento de la comunicación con la GUI que ejecuta el BCI.

```
//////////////////////////////////////////
// Inicializacion de constantes.
//////////////////////////////////////////

// Numero de puertos a los que se conectan los LEDs. Se seleccionan los pines
PWM
const int led1Pin =  3;
const int led2Pin =  5;
const int led3Pin =  6;
const int led4Pin =  9;
const int led5Pin =  10;
const int led6Pin =  11;


// Estado inicial de los LEDs a LOW
int led1State = LOW;
int led2State = LOW;
int led3State = LOW;
int led4State = LOW;
int led5State = LOW;
int led6State = LOW;

//Almacena el tiempo en microsegundos desde la última vez que se actualiza el
LED
unsigned long previousMicros1 = 0;
unsigned long previousMicros2 = 0;
unsigned long previousMicros3 = 0;
unsigned long previousMicros4 = 0;
unsigned long previousMicros5 = 0;
unsigned long previousMicros6 = 0;

// Inicializacion de las variables auxiliares
double auxi1;
double auxi2;
double auxi3;
double auxi4;
double auxi5;
double auxi6;



// cast para pasar double a long
unsigned long interval1;
unsigned long interval2;
unsigned long interval3;
unsigned long interval4;
unsigned long interval5;
unsigned long interval6;


//frecuencias y config_led
int trama_config = 0;
```

```
bool config_led;
float led1f;
float led2f;
float led3f;
float led4f;
float led5f;
float led6f;

unsigned long currentMicros1 = 0;
unsigned long currentMicros2 = 0;
unsigned long currentMicros3 = 0;
unsigned long currentMicros4 = 0;
unsigned long currentMicros5 = 0;
unsigned long currentMicros6 = 0;

//Otras variables
long contador;
long milis_inicio;
long milis_fin;

////////////////////////////////////////////
// Setup del arduino (Declaracion de pines)
////////////////////////////////////////////

void setup() {
  // Se declaran los pines como salidas
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(led3Pin, OUTPUT);
  pinMode(led4Pin, OUTPUT);
  pinMode(led5Pin, OUTPUT);
  pinMode(led6Pin, OUTPUT);

  //Inicializacion para que haga el bucle al menos una vez

  config_led=0;
  Serial.begin(9600);

  while (config_led == 0){

        if (Serial.available() > 0) {

                    while (trama_config !=99){
                      trama_config=Serial.parseInt();
                    }
                    trama_config=0;
                    config_led=Serial.parseInt();
                    led1f=Serial.parseFloat();
                    led2f=Serial.parseFloat();
                    led3f=Serial.parseFloat();
                    led4f=Serial.parseFloat();
                    led5f=Serial.parseFloat();
                    led6f=Serial.parseFloat();

                    Serial.print("config_LED: ");
                    Serial.println(config_led);
                    Serial.print("LED1: ");
                    Serial.println(led1f);
                    Serial.print("LED2: ");
                    Serial.println(led2f);
                    Serial.print("LED3: ");
```

```
                        Serial.println(led3f);
                        Serial.print("LED4: ");
                        Serial.println(led4f);
                        Serial.print("LED5: ");
                        Serial.println(led5f);
                        Serial.print("LED6: ");
                        Serial.println(led6f);

                        // Paso de las frecuencias a periodo en microsegundos
                        auxi1= (1000000/(2*led1f));
                        auxi2= (1000000/(2*led2f));
                        auxi3= (1000000/(2*led3f));
                        auxi4= (1000000/(2*led4f));
                        auxi5= (1000000/(2*led5f));
                        auxi6= (1000000/(2*led6f));

                        // cast para pasar double a long
                        interval1 = (long)auxi1;
                        interval2 = (long)auxi2;
                        interval3 = (long)auxi3;
                        interval4 = (long)auxi4;
                        interval5 = (long)auxi5;
                        interval6 = (long)auxi6;

                        contador=0;
                        while(contador <=200000) {

                           EncenderLeds(interval1, interval2, interval3,
interval4, interval5, interval6);
                           contador=contador+1;

                        }
                        led1State = LOW;
                        led2State = LOW;
                        led3State = LOW;
                        led4State = LOW;
                        led5State = LOW;
                        led6State = LOW;
                        analogWrite(led1Pin, led1State);
                        analogWrite(led2Pin, led2State);
                        analogWrite(led3Pin, led3State);
                        analogWrite(led4Pin, led4State);
                        analogWrite(led5Pin, led5State);
                        analogWrite(led6Pin, led6State);

                  }
            }
            contador=0;
}



////////////////////////////////////////////
// Codigo de programa
////////////////////////////////////////////

//Bucle que se repetira incondicionalmente
void loop()
{
```

```
  EncenderLeds(interval1, interval2, interval3, interval4, interval5,
interval6);

}

void EncenderLeds(unsigned long interval1, unsigned long interval2, unsigned
long interval3, unsigned long interval4, unsigned long interval5, unsigned long
interval6)
{
  //Obtenemos para cada pasada de bucle el tiempo actual en microsegundos
  currentMicros1 = micros();
  currentMicros2 = micros();
  currentMicros3 = micros();
  currentMicros4 = micros();
  currentMicros5 = micros();
  currentMicros6 = micros();

// Codigo LED1

   // Si la diferencia entre el tiempo actual (currentMicros) y la ultima vez
que el LED parpadeo (previousMicros) es mayor que el periodo de oscilacion, el
LED cambia de estado (Si esta a HIGH pasa a LOW y viceversa).

     if (interval1==0)
        digitalWrite(led1Pin, LOW);

     else if(currentMicros1 - previousMicros1 >= interval1)
     {
          //Guardo el microsegundo en el que cambio el estado
          previousMicros1 = currentMicros1;

          if (led1State == LOW)
              led1State = 255;
          else
              led1State = LOW;

          // Paso al pin el valor del estado
          analogWrite(led1Pin, led1State);

      }

    //Este procedimiento se repetirá para los 6 LEDS
```

Anexo III: Código Implementación GUI

A continuación se mostrarán los ficheros de texto más relevantes para la implementación de la Interfaz Gráfica de Usuario.

6.1.1 Headers

El programa consta de cuatro ficheros de cabecera:

- Console.h
- MainWindow.h
- MyThread.h
- SettingsDialog.h

Por su relevancia y para un mayor entendimiento del programa, se detallarán en este apartado los ficheros MainWindow.h y MyThread.h.

6.1.1.1 MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#define num_estimulos 6


////////////////////////////////////////////////////////
/// Constantes para fijar Objetos
////////////////////////////////////////////////////////

//Parámetros de configuración para el monitor del GNB
#define ancho_pantalla 1900
#define alto_pantalla 1020

//Monitor de un PC portatil
//#define ancho_pantalla 1250 // 1350 pantalla pequeña
//#define alto_pantalla 950// 700 pantalla pequeña

#define margen_superior 40// 20 pantalla pequeña
#define margen_lateral 20 // 15 pantalla pequeña

#define distancia_botones ancho_pantalla/3

#define ancho_estimulo (ancho_pantalla/3)-2*margen_lateral
#define alto_estimulo (alto_pantalla/2)-2*margen_superior - margen_superior/2


////////////////////////////////////////////////////////
/// Rutas a Ficheros de Control
////////////////////////////////////////////////////////
```

```
// Fichero que almacena los usuarios registrados en el sistema
#define RUTA_RegUsus "/home/gnb/Ana_Sotomayor/terminal_v5.1/RegUsus.txt"

//Fichero que almancena las frecuencias del sujeto que se enviarán a los LEDs
#define RUTA_S1 "/home/gnb/Ana_Sotomayor/terminal_v5.1/frecuencias_S1.txt"

//Fichero que almacena los tiempos y resultados en el juego 12 LUCES
#define RUTA_12LUCES
"/home/gnb/Ana_Sotomayor/terminal_v5.1/resultados12Luces.txt"

//Fichero que almacena el récord del juego 12 LUCES
#define RUTA_RECORD12LUCES
"/home/gnb/Ana_Sotomayor/terminal_v5.1/record12Luces.txt"

// Constante para el juego 12 LUCES
#define MAX_LUCES 12

//////////////////////////////////////////////////////
/// Constantes para la creación de la cola de mensajes
//////////////////////////////////////////////////////

#define IPC_LONGITUD 128
#define IPC_PERMISOS 0600
#define IPC_MENSAJE_CABECERA 2
#define IPC_MENSAJE_DATOS 1
#define IPC_MENSAJE_FIN_TRANSMISION 9
#define IPC_ROBOT_DATOS 1
#define IPC_ROBOT_FIN 2

//////////////////////////////////////////////////////
/// Programas que han de ejecutarse en paralelo a la GUI
//////////////////////////////////////////////////////

#define ARCHIVO_BASELINE "/home/gnb/Ana_Sotomayor/terminal_v5.1/baseline"
#define ARCHIVO_EJECUCIONEMOTIV
"/home/gnb/Ana_Sotomayor/terminal_v5.1/ejecucionEmotiv"

//////////////////////////////////////////////////////
/// Librerías
//////////////////////////////////////////////////////

#include <QtCore/QtGlobal>
#include <QMainWindow>

#include <QtSerialPort/QSerialPort>
#include <QFile>
#include <QPushButton>
#include <QElapsedTimer>
#include <QMutex>

//////////////////////////////////////////////////////
/// Signals, slots y funciones. Implementación de la clase MainWindow
//////////////////////////////////////////////////////

QT_BEGIN_NAMESPACE

class QLabel;

namespace Ui {
class MainWindow;
```

```
}

QT_END_NAMESPACE

class Console;
class SettingsDialog;
class MyThread;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    static MainWindow * getInstance();
    ~MainWindow();

    void delay(int seconds);

signals:
    //Señales que se emiten cuando se detecta un estímulo
    void Signal_Estimulo1();
    void Signal_Estimulo2();
    void Signal_Estimulo3();
    void Signal_Estimulo4();
    void Signal_Estimulo5();
    void Signal_Estimulo6();

private slots:
    void openSerialPort();
    void closeSerialPort();
    void about();
    void writeData(const QByteArray &data);
    void readData();

    void handleError(QSerialPort::SerialPortError error);

    // Funciones que se activan cuando se detecta un estímulo

    void Estimulo1Detectado();
    void Estimulo2Detectado();
    void Estimulo3Detectado();
    void Estimulo4Detectado();
    void Estimulo5Detectado();
    void Estimulo6Detectado();

    //Funciones de llamada de los botones

    void on_boton_inicio_clicked();
    void on_Boton_NuevoUsu_clicked();
    void on_Boton_RegUsu_clicked();

private:
    static bool instanceFlag;
    static MainWindow * myinstance;
    void initActionsConnections();

    //Funciones de configuración de estados de la GUI
    void setInicio();
    void setPosiciones();
    void checkEstados(int estados);
```

```
    void setEstado0();
    void setEstado1();
    void setEstado2();
    void setEstado3();
    void setEstado4();
    void setEstado5();
    void setEstado6();
    void setEstado21();
    void setEstado22();
    void setEstado23();
    void setEstado24();
    void setEstado25();
    void setEstado41();
    void setEstado42();
    void setEstado43();
    void setEstado44();
    void setEstado45();
    void setEstadoJuegos();



private:
    void showStatusMessage(const QString &message);
    Ui::MainWindow *ui;
    QLabel *status;
    Console *console;
    SettingsDialog *settings;
    QSerialPort *serial;

    int contador_inicio;

    QString numero_usuario;
    QString nuevo_usuario;
    int int_numero_usuario;

    QString frecuencias_completas;
    QString frecuencia1;
    QString frecuencia2;
    QString frecuencia3;
    QString frecuencia4;
    QString frecuencia5;
    QString frecuencia6;

    QString cadena_speller;
    QString nombre_usuario;

    QStringList lista_frecuencias;
    QStringList lista_usuarios;

    QElapsedTimer temporizador_baseline;

    //Estimulos: Objetos cuadrados que se muestran por pantalla

    QWidget Estimulo1;
    QWidget Estimulo2;
    QWidget Estimulo3;
    QWidget Estimulo4;
    QWidget Estimulo5;
    QWidget Estimulo6;
```

```
    //Contenido de cada estímulo: Texto informativo
    QLabel *Label1;
    QLabel *Label2;
    QLabel *Label3;
    QLabel *Label4;
    QLabel *Label5;
    QLabel *Label6;

    //Botones de ayuda: Simulador de estímulos detectados
    QPushButton Boton1;
    QPushButton Boton2;
    QPushButton Boton3;
    QPushButton Boton4;
    QPushButton Boton5;
    QPushButton Boton6;

    //Control de estados
    int estado;
    int estado_NU;
    int estado_UE;
    bool existe_usuario;

    //Juegos
    int contador12luces;
    int mirarluz;
    QElapsedTimer temporizador12l;
    int numero_fallos;

};

#endif // MAINWINDOW_H
```

6.1.1.2 MyThread.h

```
#ifndef MYTHREAD_H
#define MYTHREAD_H

#include <QThread>
#include "mainwindow.h"

// Librerías de configuración del hilo y la cola
#include <syscall.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/msg.h>


// Constantes para la creación de la cola de mensajes
#define IPC_DATOS 1
#define IPC_FIN 2
#define LIMITE_CONTADOR 3 // Cada 3 positivos consecutivos se considera un
acierto

#define MAX_ESTIMULOS 6

//Contenedor que recibirá el mensaje almacenado en la cola que contiene el
resultado
```

```
// de la ejecución del Emotiv
typedef struct
{
    long tipoComunicacion;
    int ComandoRobot;

}MensajeRobot;


class MyThread : public QThread
{
    Q_OBJECT

//Mecanismo Signals & Slots: Estas señales activarán las funciones que activan
los estímulos
signals:
    void Signal_Thread_Estimulo1();
    void Signal_Thread_Estimulo2();
    void Signal_Thread_Estimulo3();
    void Signal_Thread_Estimulo4();
    void Signal_Thread_Estimulo5();
    void Signal_Thread_Estimulo6();

private:
    static bool instanceFlagThread;
    static MyThread * myinstanceThread;


public:
    static MyThread* getInstance();
    void run();
};

#endif // MYTHREAD_H
```

6.1.2 Sources

El programa consta de cuatro ficheros fuente, cada uno de ellos asociado a su cabecera:

- Console.cpp
- MainWindow.cpp
- MyThread.cpp
- SettingsDialog.cpp

Por su relevancia y para un mayor entendimiento del programa, se detallarán en este apartado los ficheros MainWindow.cpp y MyThread.cpp.

6.1.2.1 MainWindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "console.h"
#include "settingsdialog.h"
#include "mythread.h"

#include <QMessageBox>
#include <QLabel>
#include <QtSerialPort/QSerialPort>
#include <QTextStream>
#include <QThread>
#include <QtGlobal>
#include <QElapsedTimer>
#include <QTime>
#include <QProcess>

//La variable idCola se recibe desde el Main
extern int idCola;

//Variables globales para ser leídas desde el hilo
int frecuencia1_int;
int frecuencia2_int;
int frecuencia3_int;
int frecuencia4_int;
int frecuencia5_int;
int frecuencia6_int;

////////////////////////////////////////////////////////////////////////////
/// Ventana Principal: Se crea MainWindow
////////////////////////////////////////////////////////////////////////////

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)

{
```

```
    ////////////////////////////////////////////
    /// Inicialización de variables
    ////////////////////////////////////////////

    frecuencia1_int = 0;
    frecuencia2_int = 0;
    frecuencia3_int = 0;
    frecuencia4_int = 0;
    frecuencia5_int = 0;
    frecuencia6_int = 0;

    contador_inicio=0;

    cadena_speller=" ";
    estado_NU=0;
    estado_UE=0;
    existe_usuario=false;
    contador12luces=0;
    numero_fallos=0;


    // Se crea la interfaz gráfica en el Main Window
    ui->setupUi(this);

    //Se puede seleccionar un tamaño fijo o mostrar a pantalla completa
    setFixedSize(ancho_pantalla, alto_pantalla);
    //showFullScreen();


    // Se crea la consola que irá mostrando el proceso y actuará como terminal.
    console = new Console(this);
    console->setEnabled(true);

    //Se fija como una ventana en el centro
    //Fijo ancho y alto
    console->setFixedSize(ancho_consola,alto_consola);
    //Lo centro en la ventana
    console->move(ancho_pantalla/2-ancho_consola/2, alto_pantalla/2);


    //Llamada a función de configuración inicial de ventana: Se esconden los
estímulos
    setInicio();


    // Habilitación del puerto serie para comunicación con Arduino
    serial = new QSerialPort(this);
    settings = new SettingsDialog;


    // Configuración inicial de los botones superiores
    ui->actionConnect->setEnabled(true);
    ui->actionDisconnect->setEnabled(false);
    ui->actionQuit->setEnabled(true);
    ui->actionConfigure->setEnabled(true);

    // Configuración de la barra de estado
    status = new QLabel;
    ui->statusBar->addWidget(status);
```

```
    //Función que crea las conexiones SIGNALS & SLOTS
    initActionsConnections();
}

////////////////////////////////////////////////////////////////////////////////
/// Función ~MainWindow
///     Encargada de borrar la GUI, así como todos los objetos heredados de
///     la misma
////////////////////////////////////////////////////////////////////////////////
MainWindow::~MainWindow()
{
    delete settings;
    delete ui;
}

bool MainWindow::instanceFlag = false;
MainWindow* MainWindow::myinstance = NULL;

////////////////////////////////////////////////////////////////////////////////
/// Función getInstance()
///     Sirve para obtener una instancia de la ventana MainWindow
////////////////////////////////////////////////////////////////////////////////
MainWindow* MainWindow::getInstance()
  {
    static QMutex mutex;
    if (!myinstance){
      mutex.lock();
      if (!myinstance){
        myinstance = new MainWindow;
        instanceFlag = true;
      }
      mutex.unlock();
    }

    return myinstance;
  }


////////////////////////////////////////////////////////////////////////////////
/// Función delay(int seconds)
///     En Qt no es posible llamar a funciones como sleep, wait o pause, debido
al
///     mecanismo de espera a cambios y a señales. Por tanto, es necesario
///     implementar la función delay que no pausa el sistema
////////////////////////////////////////////////////////////////////////////////
void MainWindow::delay(int seconds)
{
    QTime dieTime= QTime::currentTime().addSecs(seconds);
    while (QTime::currentTime() < dieTime)
        QCoreApplication::processEvents(QEventLoop::AllEvents, 100);
}

////////////////////////////////////////////////////////////////////////////////
/// Función showStatusMessage(const QString &message)
///     Al llamar a la función muestra el mensaje que recibe como parámetro
///     Sirve para mostrar la conexión al puerto serie
////////////////////////////////////////////////////////////////////////////////
void MainWindow::showStatusMessage(const QString &message)
{
    status->setText(message);
}
```

```
////////////////////////////////////////////////////////////////////////////
/// Función openSerialPort()
///     Se ejecuta al pulsar el botón de conexión.
///     Se configuran los parámetros de configuración del puerto serie.
////////////////////////////////////////////////////////////////////////////

void MainWindow::openSerialPort()
{

    SettingsDialog::Settings p = settings->settings();
    serial->setPortName(p.name);
    serial->setBaudRate(p.baudRate);
    serial->setDataBits(p.dataBits);
    serial->setParity(p.parity);
    serial->setStopBits(p.stopBits);
    serial->setFlowControl(p.flowControl);
    if (serial->open(QIODevice::ReadWrite)) {
        console->setEnabled(true);
        console->setLocalEchoEnabled(p.localEchoEnabled);
        ui->actionConnect->setEnabled(false);
        ui->actionDisconnect->setEnabled(true);
        ui->actionConfigure->setEnabled(false);
        showStatusMessage(tr("Connected to %1 : %2, %3, %4, %5, %6")

.arg(p.name).arg(p.stringBaudRate).arg(p.stringDataBits)

.arg(p.stringParity).arg(p.stringStopBits).arg(p.stringFlowControl));
    } else {
        QMessageBox::critical(this, tr("Error"), serial->errorString());

        showStatusMessage(tr("ERROR al abrir el puerto serie"));
    }
}

////////////////////////////////////////////////////////////////////////////
/// Función closeSerialPort()
///     Se ejecuta al pulsar el botón de Desconectar Puerto Serie
////////////////////////////////////////////////////////////////////////////
void MainWindow::closeSerialPort()
{
    if (serial->isOpen())
        serial->close();
    console->setEnabled(false);
    ui->actionConnect->setEnabled(true);
    ui->actionDisconnect->setEnabled(false);
    ui->actionConfigure->setEnabled(true);
    showStatusMessage(tr("Disconnected"));
}

////////////////////////////////////////////////////////////////////////////
/// Función about()
///     Se ejecuta al pulsar el botón About, muestra información de la GUI
////////////////////////////////////////////////////////////////////////////
void MainWindow::about()
{
    QMessageBox::about(this, tr("Interfaz Cerebro Máquina"),
                       tr("Este BCI permite la comunicación entre personas a
través "
                          "de la actividad cerebral, habilitando un medio
efectivo "
```

```
                            "de interrelación en personas con alguna
discapacidad."));
}

////////////////////////////////////////////////////////////////////////////////
/// Función writeData(const QByteArray &data)
///     Envío de datos al puerto serie
///     Envía al puerto serie el QByteArray que recibe como parametro
////////////////////////////////////////////////////////////////////////////////
void MainWindow::writeData(const QByteArray &data)
{
    serial->write(data);
    console->appendPlainText("Se han mandado datos al puerto serie");
    console->putData(data);
}

////////////////////////////////////////////////////////////////////////////////
/// Función readData()
///     Recepción de datos desde el puerto serie. Se muestran en la consola
////////////////////////////////////////////////////////////////////////////////
void MainWindow::readData()
{
    QByteArray data = serial->readAll();
    console->putData(data);
    console->appendPlainText("Se han leido datos del puerto serie");
}

////////////////////////////////////////////////////////////////////////////////
/// Función handleError(QSerialPort::SerialPortError error)
///     En caso de error, se cierra el puerto serie
////////////////////////////////////////////////////////////////////////////////
void MainWindow::handleError(QSerialPort::SerialPortError error)
{
    if (error == QSerialPort::ResourceError) {
        QMessageBox::critical(this, tr("Critical Error"), serial-
>errorString());
        closeSerialPort();
    }
}

////////////////////////////////////////////////////////////////////////////////
///  Función initActionsConnections()
///     Mecanismo de conexiones SIGNALS & SLOTS. Conexiones necesarias para
///     el funcionamiento del programa
////////////////////////////////////////////////////////////////////////////////
void MainWindow::initActionsConnections()
{
    //Conexión de los botones de acción
    connect(ui->actionConnect, SIGNAL(triggered()), this,
SLOT(openSerialPort()));
    connect(ui->actionDisconnect, SIGNAL(triggered()), this,
SLOT(closeSerialPort()));
    connect(ui->actionQuit, SIGNAL(triggered()), this, SLOT(close()));
    connect(ui->actionConfigure, SIGNAL(triggered()), settings, SLOT(show()));
    connect(ui->actionClear, SIGNAL(triggered()), console, SLOT(clear()));
    connect(ui->actionAbout, SIGNAL(triggered()), this, SLOT(about()));
    connect(ui->actionAboutQt, SIGNAL(triggered()), qApp, SLOT(aboutQt()));

    // Si ocurre un error en la apertura del puerto serie, notificar en la
ventana principal
```

```
    connect(serial, SIGNAL(error(QSerialPort::SerialPortError)), this,
SLOT(handleError(QSerialPort::SerialPortError)));

    // Si el puerto serie está preparado para ser leído, llama a la GUI para que
le lea
    connect(serial, SIGNAL(readyRead()), this, SLOT(readData()));

    //Desde la consola mandar datos al puerto serie
    connect(console, SIGNAL(getData(QByteArray)), this,
SLOT(writeData(QByteArray)));

    //Conexion de estados. Cuando se emite la señal Signal_EstimuloX, se activan
    // los procedimientos de detección de estímulos
    connect(this, SIGNAL(Signal_Estimulo1()), this, SLOT(Estimulo1Detectado()));
    connect(this, SIGNAL(Signal_Estimulo2()), this, SLOT(Estimulo2Detectado()));
    connect(this, SIGNAL(Signal_Estimulo3()), this, SLOT(Estimulo3Detectado()));
    connect(this, SIGNAL(Signal_Estimulo4()), this, SLOT(Estimulo4Detectado()));
    connect(this, SIGNAL(Signal_Estimulo5()), this, SLOT(Estimulo5Detectado()));
    connect(this, SIGNAL(Signal_Estimulo6()), this, SLOT(Estimulo6Detectado()));


    //Se activan también estos mismos procedimientos si se pulsan los botones de
ayuda
    connect(ui->pushButton_1, SIGNAL(clicked()), this,
SLOT(Estimulo1Detectado()));
    connect(ui->pushButton_2, SIGNAL(clicked()), this,
SLOT(Estimulo2Detectado()));
    connect(ui->pushButton_3, SIGNAL(clicked()), this,
SLOT(Estimulo3Detectado()));
    connect(ui->pushButton_4, SIGNAL(clicked()), this,
SLOT(Estimulo4Detectado()));
    connect(ui->pushButton_5, SIGNAL(clicked()), this,
SLOT(Estimulo5Detectado()));
    connect(ui->pushButton_6, SIGNAL(clicked()), this,
SLOT(Estimulo6Detectado()));

}

////////////////////////////////////////////////////////////////////////////
/// Función setInicio()
///     Coloca los botones en sus posiciones, y esconde los estímulos hasta que
///     se realice la configuración inicial.
////////////////////////////////////////////////////////////////////////////
void MainWindow::setInicio()
{
    estado =0;
    contador_inicio=0;

    //Se muestra el texto de bienvenida
    ui->Bienvenido->move(ancho_pantalla/2-ui->Bienvenido->width()/2,
alto_pantalla/4);
    ui->InicieSesion->move(ancho_pantalla/2-ui->InicieSesion->width()/2,
alto_pantalla/3);

    //Se configuran los botones de Nuevo Usuario y Usuario Regustrado
    ui->Boton_NuevoUsu->move(ancho_pantalla/2-ui->Boton_NuevoUsu->width()/2-
distancia_botones/2, alto_pantalla/2);
    ui->Boton_RegUsu->move(ancho_pantalla/2-ui->Boton_RegUsu-
>width()/2+distancia_botones/2, alto_pantalla/2);

    //Todo lo demás se esconde hasta que se pulse uno de los dos botones
```

```
    ui->boton_inicio->hide();
    ui->label_InfoNuevoUsu->hide();
    ui->label_introducenombre->hide();
    ui->line_NombreUsu->hide();
    console->hide();
    ui->estimulo1->hide();
    ui->estimulo2->hide();
    ui->estimulo3->hide();
    ui->estimulo4->hide();
    ui->estimulo5->hide();
    ui->estimulo6->hide();

}

////////////////////////////////////////////////////////////////////////////
/// Función on_Boton_NuevoUsu_clicked(): Creación de un nuevo Usuario
///     Solicita un nombre de Usuario, le asigna un número de Usuario y
///     lo almacena en el fichero de registro de usuarios
////////////////////////////////////////////////////////////////////////////
void MainWindow::on_Boton_NuevoUsu_clicked()
{

    // estado_NU será igual a 1 al pulsar el botón por segunda vez.
    // En este caso se ha recibido ya el nombre de usuario, y se registra

    if (estado_NU==1){ //Almacenar Usuario

        QFile file_regusus(RUTA_RegUsus);

        if (!file_regusus.open(QIODevice::ReadWrite | QIODevice::Text)){
            ui->label_introducenombre->setText("ERROR! No se encuentra el
registro");
        }

        //Se lee hasta el final el registro de usuarios
        while (!file_regusus.atEnd()) {

            numero_usuario=file_regusus.readLine();
            lista_usuarios=numero_usuario.split(" ", QString::SkipEmptyParts);
            numero_usuario=lista_usuarios.at(0);
            nombre_usuario=lista_usuarios.at(1);

        }
        //Cuando se alcanza el número del último usuario, se suma 1 y se
almacena como nuevo
        int_numero_usuario=numero_usuario.toInt();
        int_numero_usuario=int_numero_usuario +1;
        numero_usuario=QString::number(int_numero_usuario);

        ui->label_introducenombre->show();
        nombre_usuario=ui->line_NombreUsu->text();

        nuevo_usuario=numero_usuario + " " + nombre_usuario;
        nombre_usuario="Eres el usuario: " + numero_usuario + " Con nombre " +
nombre_usuario;
        ui->label_introducenombre->setText(nombre_usuario);
        ui->label_introducenombre->move(ui->Boton_NuevoUsu->x(), ui-
>Boton_NuevoUsu->y()+100);
        ui->label_introducenombre->setFixedSize(300,20);

        file_regusus.close();
```

```
        //Se vuelve a abrir el fichero en modo Append (añadir al final del
fichero)
        if (!file_regusus.open(QIODevice::Append | QIODevice::Text))
            return;

        QTextStream out(&file_regusus);
        out << nuevo_usuario << "\n";
        file_regusus.close();

        //Una vez se almacena el usuario, se muestra el botón de grabar Baseline
        ui->boton_inicio->show();
        ui->boton_inicio->setText("Pulse aquí para grabar baseline");
        ui->boton_inicio->setFixedSize(ancho_consola, alto_consola);
        ui->boton_inicio->move(ancho_pantalla/2-ancho_consola/2,
alto_pantalla*3/4);

    }

    //La primera vez que se pulsa el botón de Nuevo Usuario, se solicita un
nombre
    ui->label_InfoNuevoUsu->show();
    ui->line_NombreUsu->show();
    ui->label_InfoNuevoUsu->move(ui->Boton_NuevoUsu->x(), ui->Boton_NuevoUsu-
>y()+50);
    ui->line_NombreUsu->move(ui->label_InfoNuevoUsu->x()+ui->label_InfoNuevoUsu-
>width() +20, ui->Boton_NuevoUsu->y()+50);
    estado_NU=1;
    ui->Boton_NuevoUsu->setText("Crear Usuario");

}

////////////////////////////////////////////////////////////////////////////
/// Función on_Boton_RegUsu_clicked(): Validación de Usuario Registrado
///     Solicita un nombre de Usuario, se verifica que se encuentre en el
///     registro y se busca su número de usuario para establecer sus frecuencias
////////////////////////////////////////////////////////////////////////////

void MainWindow::on_Boton_RegUsu_clicked()
{
    // estado_UE será igual a 1 al pulsar el botón por segunda vez.
    // En este caso se ha recibido ya el nombre de usuario, y se valida
    if (estado_UE==1){ //Validar Usuario

        QFile file_regusus(RUTA_RegUsus);

        if (!file_regusus.open(QIODevice::ReadWrite | QIODevice::Text)){
            ui->label_introducenombre->setText("ERROR! No se encuentra el
registro");
        }

        nombre_usuario=ui->line_NombreUsu->text();
        existe_usuario = false;

        while (!file_regusus.atEnd()) {

            numero_usuario=file_regusus.readLine();
            lista_usuarios=numero_usuario.split(" ", QString::SkipEmptyParts);
            numero_usuario=lista_usuarios.at(0);
```

```
            if ((nombre_usuario.compare(lista_usuarios.at(1))== -1) &&
(existe_usuario==false)){
                existe_usuario=true;
                int_numero_usuario=numero_usuario.toInt();
            }
        }

         if (existe_usuario==true){

            numero_usuario=QString::number(int_numero_usuario);
            ui->label_introducenombre->show();
            nombre_usuario="Eres el usuario: " + numero_usuario + " Con nombre "
+ nombre_usuario;
            ui->label_introducenombre->setText(nombre_usuario);
            ui->label_introducenombre->move(ui->Boton_RegUsu->x(), ui-
>Boton_RegUsu->y()+100);
            ui->label_introducenombre->setFixedSize(300,20);

            ui->boton_inicio->show();
            ui->boton_inicio->setText("Pulse aquí para grabar el baseline");
            ui->boton_inicio->setFixedSize(ancho_consola, alto_consola);
            ui->boton_inicio->move(ancho_pantalla/2-ancho_consola/2,
alto_pantalla*3/4);

         }
         else
            ui->label_InfoNuevoUsu->setText("Usuario no existe. Introduce
nombre");
     }

    ui->label_InfoNuevoUsu->show();
    ui->line_NombreUsu->show();
    ui->label_InfoNuevoUsu->move(ui->Boton_RegUsu->x(), ui->Boton_RegUsu-
>y()+50);
    ui->line_NombreUsu->move(ui->label_InfoNuevoUsu->x()+ui->label_InfoNuevoUsu-
>width() +20, ui->Boton_RegUsu->y()+50);
    estado_UE=1;
    ui->Boton_RegUsu->setText("Validar Usuario");


}

////////////////////////////////////////////////////////////////////////////
/// Función on_boton_inicio_clicked():
///     La primera vez que se pulsa graba el baseline del usuario
///     La segunda, comienza el programa con normalidad
///     La tercera, finaliza el programa
////////////////////////////////////////////////////////////////////////////

void MainWindow::on_boton_inicio_clicked()
{

// La primera vez que se pulsa, graba el baseline del usuario
    if (contador_inicio==0){
            contador_inicio++;

            //Se configura la pantalla vacía y en negro
                ui->Bienvenido->hide();
                ui->InicieSesion->hide();
                ui->Boton_NuevoUsu->hide();
                ui->Boton_RegUsu->hide();
```

```
                ui->label_introducenombre->hide();
                ui->label_InfoNuevoUsu->hide();
                ui->line_NombreUsu->hide();
                ui->boton_inicio->hide();


            ////////////////////////////////////////////////////////
            /// Grabar y Guardar Baseline
            ////////////////////////////////////////////////////////

            //Llamada al programa baseline
            QProcess *procesoBaseline = new QProcess(this);
            QString programaBaseline = ARCHIVO_BASELINE;
            QStringList argumentos;
            argumentos << numero_usuario;

            procesoBaseline->start(programaBaseline, argumentos);
            procesoBaseline->waitForStarted();

            //Se espera en la interfaz "2 segundos config + 60 segundos + 2
segundos config" para que corra el baseline
            delay(64);

            //Se vuelven a mostrar opciones
            ui->boton_inicio->show();
            ui->boton_inicio->setText("Pulse aquí para comenzar el programa");
            procesoBaseline->close();

    }


//Si se vuelve a pulsar, finaliza el programa
    else if (contador_inicio==2){
            this->close();
    }
    else{

        contador_inicio++;

        ui->boton_inicio->setText("Pulse aquí para finalizar");

        // Establecer posiciones
        setPosiciones();

        //Estado 10: Menú inicial
        estado = 10;
        checkEstados(estado);


        //Se lee el fichero de mejores frecuencias del sujeto
        QFile file(RUTA_S1);

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        console->appendPlainText("El fichero no se ha abierto");
        }

        while (!file.atEnd()) {

            //Se leen las frecuencias y se guardan como QString
            frecuencias_completas=file.readLine();
```

```
            lista_frecuencias= frecuencias_completas.split(" ",
QString::SkipEmptyParts);
            frecuencia1=lista_frecuencias.at(0);
            frecuencia2=lista_frecuencias.at(1);
            frecuencia3=lista_frecuencias.at(2);
            frecuencia4=lista_frecuencias.at(3);
            frecuencia5=lista_frecuencias.at(4);
            frecuencia6=lista_frecuencias.at(5);

        }

        //Se envían las frecuencias a arduino para configurarlas. Se añade el
"99" inicial de flag para el puerto serie, y un "1" de Configuración

        frecuencias_completas =  "99 1 " + frecuencia1 + " " + frecuencia2 + " "
+ frecuencia3 + " " + frecuencia4 + " " + frecuencia5 + " " + frecuencia6;
        emit console->getData(frecuencias_completas.toLocal8Bit());


        frecuencia1_int=frecuencia1.toInt();
        frecuencia2_int=frecuencia2.toInt();
        frecuencia3_int=frecuencia3.toInt();
        frecuencia4_int=frecuencia4.toInt();
        frecuencia5_int=frecuencia5.toInt();
        frecuencia6_int=frecuencia6.toInt();

        ////////////////////////////////////////////////////////
        /// Llamar a la ejecución de Emotiv
        ////////////////////////////////////////////////////////


        QProcess *procesoejecucionEmotiv = new QProcess(this);
        QString programaejecucionEmotiv = ARCHIVO_EJECUCIONEMOTIV;
        QString idCola_string = QString::number(idCola);
        qDebug("desde el mainwindow el idcola es:", idCola);
        QStringList argumentos_ejecucionEmotiv;
        //argumentos_ejecucionEmotiv << numero_usuario << idCola_string;
        argumentos_ejecucionEmotiv << numero_usuario << idCola_string;

        procesoejecucionEmotiv->start(programaejecucionEmotiv,
argumentos_ejecucionEmotiv);
        procesoejecucionEmotiv->waitForStarted();

        console->appendPlainText("Se ha llamado a la ejecucion de emotiv");
        console->appendPlainText("Primer argumento:");
        console->appendPlainText(argumentos_ejecucionEmotiv.at(0));
        console->appendPlainText("Segundo argumento:");
        console->appendPlainText(argumentos_ejecucionEmotiv.at(1));


    }

}


////////////////////////////////////////////////////////////////////////////////
/// Función setPosiciones()
///     Establece las posiciones de los estímulos, después de grabar el Baseline
////////////////////////////////////////////////////////////////////////////////
void MainWindow::setPosiciones()
{
```

```
//Mostrar el menú y los estímulos

    console->show();
    ui->estimulo1->show();
    ui->estimulo2->show();
    ui->estimulo3->show();
    ui->estimulo4->show();
    ui->estimulo5->show();
    ui->estimulo6->show();

// Centrar estímulos

    ui->estimulo1->setFixedSize(ancho_estimulo, alto_estimulo);
    ui->estimulo2->setFixedSize(ancho_estimulo, alto_estimulo);
    ui->estimulo3->setFixedSize(ancho_estimulo, alto_estimulo);
    ui->estimulo4->setFixedSize(ancho_estimulo, alto_estimulo);
    ui->estimulo5->setFixedSize(ancho_estimulo, alto_estimulo);
    ui->estimulo6->setFixedSize(ancho_estimulo, alto_estimulo);

    ui->estimulo1->move(margen_lateral,margen_superior);
    ui->estimulo2->move(ancho_pantalla/3 + margen_lateral, margen_superior);
    ui->estimulo3->move(2*ancho_pantalla/3 + margen_lateral, margen_superior);
    ui->estimulo4->move(margen_lateral,alto_pantalla/2 + margen_superior/2);
    ui->estimulo5->move(ancho_pantalla/3 + margen_lateral,alto_pantalla/2 +
margen_superior/2);
    ui->estimulo6->move(2*ancho_pantalla/3 + margen_lateral,alto_pantalla/2 +
margen_superior/2);

//Boton centrado en la esquina superior izquierda de cada estimulo
      ui->pushButton_1->move(0,0);
      ui->pushButton_2->move(0,0);
      ui->pushButton_3->move(0,0);
      ui->pushButton_4->move(0,0);
      ui->pushButton_5->move(0,0);
      ui->pushButton_6->move(0,0);


//Tamaño de texto grande
      QFont f( "DejaVu Serif", 44, QFont::Bold);
      ui->label_1->setFont(f);
      ui->label_2->setFont(f);
      ui->label_3->setFont(f);
      ui->label_4->setFont(f);
      ui->label_5->setFont(f);
      ui->label_6->setFont(f);

//Texto centrado en cada estímulo
      ui->label_1->move(0,0);
      ui->label_2->move(0,0);
      ui->label_3->move(0,0);
      ui->label_4->move(0,0);
      ui->label_5->move(0,0);
      ui->label_6->move(0,0);
      ui->label_1->setFixedSize(ancho_estimulo, alto_estimulo);
      ui->label_2->setFixedSize(ancho_estimulo, alto_estimulo);
      ui->label_3->setFixedSize(ancho_estimulo, alto_estimulo);
      ui->label_4->setFixedSize(ancho_estimulo, alto_estimulo);
      ui->label_5->setFixedSize(ancho_estimulo, alto_estimulo);
      ui->label_6->setFixedSize(ancho_estimulo, alto_estimulo);
```

```
}

////////////////////////////////////////////////////////////////////////////////
////////////
/// AQUI COMIENZA LA MÁQUINA DE ESTADOS
////////////////////////////////////////////////////////////////////////////////
////////////


////////////////////////////////////////////////////////////////////////////////
/// Función setEstado0
///     Estado 0 corresponde con el Menú Inicial de la GUI
////////////////////////////////////////////////////////////////////////////////
void MainWindow::setEstado0()
{
    cadena_speller="";
    //Fijar contenido de los cubos
          ui->label_1->setText("Juegos");
          ui->label_2->setText("Speller");
          ui->label_3->setText("Sí / No");
          ui->label_4->setText("Easy \n Life");
          ui->label_5->setText("Re- \n calibrar");
          ui->label_6->setText("Cerrar \n Sesión");

}

////////////////////////////////////////////////////////////////////////////////
/// Función setEstado1
///     Estado 1 se ejecuta cuando desde el Estado 0 se detecta
///     el estímulo 1 (Juegos)
////////////////////////////////////////////////////////////////////////////////
void MainWindow::setEstado1()
{
    //Fijar contenido de los cubos
      ui->label_1->setText("12 Luces");
      ui->label_2->setText("Contrarreloj \n 30s");
      ui->label_3->setText("Contrarreloj \n 1 minuto");
      ui->label_4->setText("Contrarreloj \n 2 minutos");
      ui->label_5->setText("Dectector \n Estimulos ");
      ui->label_6->setText("Volver \n al menú");
}

////////////////////////////////////////////////////////////////////////////////
/// Función setEstado2
///     Estado 2 se ejecuta cuando desde el Estado 0 se detecta
///     el estímulo 2 (SPELLER)
////////////////////////////////////////////////////////////////////////////////

void MainWindow::setEstado2()
{
    //Fijar contenido de los cubos
      ui->label_1->setText("A E I \n O U T");
      ui->label_2->setText("S N R \n D L C");
      ui->label_3->setText("M P B \n G V Y");
      ui->label_4->setText("Q H F \n Z J Ñ");
      ui->label_5->setText("X W K \n espacio . borrar");
      ui->label_6->setText("Volver \n al menú");
}
```

```
////////////////////////////////////////////////////////////////////////////
/// Función setEstado3
///     Estado 3 se ejecuta cuando desde el Estado 0 se detecta
///     el estímulo 3 (Sí / No)
////////////////////////////////////////////////////////////////////////////
void MainWindow::setEstado3()
{
    //Fijar contenido de los cubos
      ui->label_1->setText("Sí");
      ui->label_2->setText("No lo Sé");
      ui->label_3->setText("No");
      ui->label_4->setText("Tal Vez");
      ui->label_5->setText("Siguiente \n Pregunta");
      ui->label_6->setText("Volver \n al menú");
}

////////////////////////////////////////////////////////////////////////////
/// Función setEstado4
///     Estado 4 se ejecuta cuando desde el Estado 0 se detecta
///     el estímulo 4 (Easy Life)
////////////////////////////////////////////////////////////////////////////

void MainWindow::setEstado4()
{
    //Fijar contenido de los cubos

      ui->label_1->setText("Sensaciones");
      ui->label_2->setText("Sentimientos");
      ui->label_3->setText("Acciones");
      ui->label_4->setText("Actividades");
      ui->label_5->setText("-");
      ui->label_6->setText("Volver \n al menú");
}

////////////////////////////////////////////////////////////////////////////
/// Función setEstado5
///     Estado 5 se ejecuta cuando desde el Estado 0 se detecta
///     el estímulo 5 (Recalibrar)
////////////////////////////////////////////////////////////////////////////
void MainWindow::setEstado5()
{
    setInicio();
    estado_NU=0;
    estado_UE=0;
    contador_inicio=0;


}
////////////////////////////////////////////////////////////////////////////
/// Función setEstado6
///     Estado 6 se ejecuta cuando desde el Estado 0 se detecta
///     el estímulo 6 (Volver al Menú)
////////////////////////////////////////////////////////////////////////////
void MainWindow::setEstado6()
{
    //Estado 6: Volver al menú
    estado = 10;
    setEstado0();

}
```

```
////////////////////////////////////////////////////////////////////////////
/// Función setEstadoJuegos()
///     Para todos los juegos, cada estímulo corresponderá a un número.
///     La GUI indicará en cada momento al usuario a qué luz (número) debe
///     mirar.
////////////////////////////////////////////////////////////////////////////
void MainWindow::setEstadoJuegos()
{
    ui->label_1->setText("1");
    ui->label_2->setText("2");
    ui->label_3->setText("3");
    ui->label_4->setText("4");
    ui->label_5->setText("5");
    ui->label_6->setText("6");
}

////////////////////////////////////////////////////////////////////////////
/// ESTADOS CORRESPONDIENTES AL SPELLER (Estado 2)
////////////////////////////////////////////////////////////////////////////

//Cuando desde el Estado 2 (SPELLER) se detecta el estímulo 1 (AEIOU)
void MainWindow::setEstado21()
{
      ui->label_1->setText("A");
      ui->label_2->setText("E");
      ui->label_3->setText("I");
      ui->label_4->setText("O");
      ui->label_5->setText("U");
      ui->label_6->setText("T");
}

//Cuando desde el Estado 2 (SPELLER) se detecta el estímulo 2 (SNRDLC)
void MainWindow::setEstado22()
{
      ui->label_1->setText("S");
      ui->label_2->setText("N");
      ui->label_3->setText("R");
      ui->label_4->setText("D");
      ui->label_5->setText("L");
      ui->label_6->setText("C");
}

//Cuando desde el Estado 2 (SPELLER) se detecta el estímulo 3 (MPBGVY)
void MainWindow::setEstado23()
{
      ui->label_1->setText("M");
      ui->label_2->setText("P");
      ui->label_3->setText("B");
      ui->label_4->setText("G");
      ui->label_5->setText("V");
      ui->label_6->setText("Y");
}

//Cuando desde el Estado 2 (SPELLER) se detecta el estímulo 4 (QHFZJÑ)
void MainWindow::setEstado24()
{

      ui->label_1->setText("Q");
      ui->label_2->setText("H");
      ui->label_3->setText("F");
      ui->label_4->setText("Z");
```

```
	ui->label_5->setText("J");
	ui->label_6->setText("Ñ");
}

//Cuando desde el Estado 2 (SPELLER) se detecta el estímulo 5 (XWK ESPACIO .
BORRAR)
void MainWindow::setEstado25()
{

	ui->label_1->setText("X");
	ui->label_2->setText("W");
	ui->label_3->setText("K");
	ui->label_4->setText("ESPACIO");
	ui->label_5->setText(".");
	ui->label_6->setText("BORRAR");
}
// Los siguientes estados (211 a 256) se controlarán desde la máquina de estados
//  Máquina de estados--> funcion checkEstados(int estados)


////////////////////////////////////////////////////////////////////////////////
/// ESTADOS CORRESPONDIENTES A EASY LIFE (Estado 4)
////////////////////////////////////////////////////////////////////////////////


//Cuando desde el Estado 4 (EASY LIFE) se detecta el estímulo 1 (Sensaciones)
void MainWindow::setEstado41()
{

	ui->label_1->setText("Calor");
	ui->label_2->setText("Frío");
	ui->label_3->setText("Hambre");
	ui->label_4->setText("Sueño");
	ui->label_5->setText("Cansancio");
	ui->label_6->setText("Volver");
}

//Cuando desde el Estado 4 (EASY LIFE) se detecta el estímulo 2 (Sentimientos)
void MainWindow::setEstado42()
{

	ui->label_1->setText("Triste");
	ui->label_2->setText("Alegre");
	ui->label_3->setText("Soledad");
	ui->label_4->setText("Amor");
	ui->label_5->setText("Aburrimiento");
	ui->label_6->setText("Volver");
}

//Cuando desde el Estado 4 (EASY LIFE) se detecta el estímulo 3 (Acciones)
void MainWindow::setEstado43()
{

	ui->label_1->setText("Ir al \n baño");
	ui->label_2->setText("Comer");
	ui->label_3->setText("Salir fuera");
	ui->label_4->setText("Ir a la \n habitación");
	ui->label_5->setText("Dar un \n paseo");
	ui->label_6->setText("Volver");
}
```

```
//Cuando desde el Estado 4 (EASY LIFE) se detecta el estímulo 4 (Actividades)
void MainWindow::setEstado44()
{

      ui->label_1->setText("Encender TV");
      ui->label_2->setText("Apagar TV");
      ui->label_3->setText("Escuchar música");
      ui->label_4->setText("Leer");
      ui->label_5->setText("Llamar a \n alguien");
      ui->label_6->setText("Volver");
}

//Cuando desde el Estado 4 (EASY LIFE) se detecta el estímulo 5 (-)
// PENDIENTE CONFIGURAR PARA USUARIO
void MainWindow::setEstado45()
{

      ui->label_1->setText("-");
      ui->label_2->setText("-");
      ui->label_3->setText("-");
      ui->label_4->setText("-");
      ui->label_5->setText("-");
      ui->label_6->setText("Volver");
}



////////////////////////////////////////////////////////////////////////////////
///
/// Función checkEstados(int estados)
///     Controlador de Estados. Dependiendo del estado que recibe como parámetro
///     llama a la función correspondiente
////////////////////////////////////////////////////////////////////////////////

void MainWindow::checkEstados(int estados)
{
    switch (estados)
    {
        //Si el estado es 1, entra a JUEGOS
        case 1:
            setEstado1();
            break;
        //Si el estado es 2, entra a SPELLER
        case 2:
            setEstado2();
            break;
        //Si el estado es 3, entra a SI/NO
        case 3:
            setEstado3();
            break;
        //Si el estado es 4, entra a RECALIBRAR
        case 4:
            setEstado4();
            break;
        //Si el estado es 5, entra a EASY LIFE
        case 5:
            setEstado5();
            break;
        //Si el estado es 6, entra a SALIR
        case 6:
            setEstado6();
```

```
            break;
        // Estado 0, menú principal
        case 10:
            setEstado0();
            break;
        // Si desde el estado 1 se emite 1, el estado es 12 LUCES
        case 11:
            setEstadoJuegos();

            //12 Luces
            if (contador12luces==0){
                console->appendPlainText("Comenzando Juego: 12 Luces. Intenta
superar el récord de detección de luces. Buena suerte!");
                temporizador12l.start();
                contador12luces=contador12luces+1;
                mirarluz = qrand() % 6;
                if (mirarluz==0)
                    mirarluz=1;

                console->appendPlainText("Mira a la luz...");
                console->appendPlainText(QString::number(mirarluz));
            }

            else if (contador12luces==MAX_LUCES){
                //Se captura el tiempo

                contador12luces=0;
                console->appendPlainText("Has tardado " +
QString::number(temporizador12l.elapsed()/1000) + " segundos y has tenido " +
QString::number(numero_fallos) + " fallos");
                console->appendPlainText("Juego terminado.");
                estado=10;
                setEstado0();

                // Se guarda el tiempo en un fichero

                QFile file_12luces(RUTA_12LUCES);

                 if (!file_12luces.open(QIODevice::Append | QIODevice::Text))
                            return;

                QTextStream out(&file_12luces);
                out << "Usuario: " << nuevo_usuario  << "\n";
                out << "Tiempo: " <<
QString::number(temporizador12l.elapsed()/1000) << "\n";
                out << "Número de fallos: " << QString::number(numero_fallos) <<
"\n";
                out << "\n";
                file_12luces.close();

            } else{

                console->appendPlainText("LUZ DETECTADA");
                console->appendPlainText("Numero de luces realizadas: " +
QString::number(contador12luces));
                contador12luces=contador12luces+1;
                mirarluz = qrand() % 6;
                if (mirarluz==0)
                    mirarluz=1;

                console->appendPlainText("Mira a la luz...");
```

```
            console->appendPlainText(QString::number(mirarluz));

        }
        break;
    case 12:
        setEstadoJuegos();
        break;
    case 13:
        setEstadoJuegos();
        break;
    case 14:
        setEstadoJuegos();
        break;
    case 15:
        setEstadoJuegos();
        break;
    case 16:
        setEstadoJuegos();
        break;
    case 21:
        setEstado21();
        break;
    case 22:
        setEstado22();
        break;
    case 23:
        setEstado23();
        break;
    case 24:
        setEstado24();
        break;
    case 25:
        setEstado25();
        break;

    case 211:
        cadena_speller=cadena_speller + "A";
        //console->appendPlainText(QString::number(estado));
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 212:
        cadena_speller=cadena_speller + "E";
        //console->appendPlainText(QString::number(estado));
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 213:
        cadena_speller=cadena_speller + "I";
        //console->appendPlainText(QString::number(estado));
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 214:
        cadena_speller=cadena_speller + "O";
        //console->appendPlainText(QString::number(estado));
        console->appendPlainText(cadena_speller);
        estado = 2;
```

```
        setEstado2();
        break;
    case 215:
        cadena_speller=cadena_speller + "U";
        //console->appendPlainText(QString::number(estado));
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 216:
        cadena_speller=cadena_speller + "T";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;

    //S N R D L C
    case 221:
        cadena_speller=cadena_speller + "S";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 222:
        cadena_speller=cadena_speller + "N";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 223:
        cadena_speller=cadena_speller + "R";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 224:
        cadena_speller=cadena_speller + "D";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 225:
        cadena_speller=cadena_speller + "L";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 226:
        cadena_speller=cadena_speller + "C";
        estado = 2;
        setEstado2();
        break;
    case 231:
        cadena_speller=cadena_speller + "M";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 232:
        cadena_speller=cadena_speller + "P";
        console->appendPlainText(cadena_speller);
```

```
            estado = 2;
            setEstado2();
            break;
        case 233:
            cadena_speller=cadena_speller + "B";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 234:
            cadena_speller=cadena_speller + "G";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 235:
            cadena_speller=cadena_speller + "V";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 236:
            cadena_speller=cadena_speller + "Y";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 241:
            cadena_speller=cadena_speller + "Q";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 242:
            cadena_speller=cadena_speller + "H";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 243:
            cadena_speller=cadena_speller + "F";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 244:
            cadena_speller=cadena_speller + "Z";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 245:
            cadena_speller=cadena_speller + "J";
            console->appendPlainText(cadena_speller);
            estado = 2;
            setEstado2();
            break;
        case 246:
            cadena_speller=cadena_speller + "Ñ";
            console->appendPlainText(cadena_speller);
            estado = 2;
```

```
        setEstado2();
        break;
    case 251:
        cadena_speller=cadena_speller + "X";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 252:
        cadena_speller=cadena_speller + "W";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 253:
        cadena_speller=cadena_speller + "K";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 254:
        cadena_speller=cadena_speller + " ";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 255:
        cadena_speller=cadena_speller + ".";
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;
    case 256:
        cadena_speller.chop(1);
        console->appendPlainText(cadena_speller);
        estado = 2;
        setEstado2();
        break;

    // Estados Si / No

    case 31:
        console->appendPlainText("Sí");
        estado = 3;
        break;

    case 32:
        console->appendPlainText("No lo sé");
        estado = 3;
        break;

    case 33:
        console->appendPlainText("No");
        estado = 3;
        break;

    case 34:
        console->appendPlainText("Tal Vez");
        estado = 3;
        setEstado3();
        break;
```

```
 case 35:
     console->appendPlainText("Siguiente Pregunta");
     estado = 3;
     break;


// Estados Easy Life

 // Sensaciones
 case 41:
     setEstado41();
     break;
 // Sentimientos
 case 42:
     setEstado42();
     break;
 // Acciones
 case 43:
     setEstado43();
     break;
 // Actividades
 case 44:
     setEstado44();
     break;

 //Por determinar
 case 45:
     setEstado45();
     break;

 case 411:
     console->appendPlainText("Calor");
     estado = 4;
     setEstado4();
     break;
 case 412:
     console->appendPlainText("Frío");
     estado = 4;
     setEstado4();
     break;
 case 413:
     console->appendPlainText("Hambre");
     estado = 4;
     setEstado4();
     break;
 case 414:
     console->appendPlainText("Sueño");
     estado = 4;
     setEstado4();
     break;
 case 415:
     console->appendPlainText("Cansancio");
     estado = 4;
     setEstado4();
     break;
 case 421:
     console->appendPlainText("Triste");
     estado = 4;
     setEstado4();
     break;
```

```
case 422:
    console->appendPlainText("Alegre");
    estado = 4;
    setEstado4();
    break;
case 423:
    console->appendPlainText("Soledad");
    estado = 4;
    setEstado4();
    break;
case 424:
    console->appendPlainText("Amor");
    estado = 4;
    setEstado4();
    break;
case 425:
    console->appendPlainText("Aburrimiento");
    estado = 4;
    setEstado4();
    break;
case 431:
    console->appendPlainText("Ir al baño");
    estado = 4;
    setEstado4();
    break;
case 432:
    console->appendPlainText("Comer");
    estado = 4;
    setEstado4();
    break;
case 433:
    console->appendPlainText("Salir fuera");
    estado = 4;
    setEstado4();
    break;
case 434:
    console->appendPlainText("Ir a la habitación");
    estado = 4;
    setEstado4();
    break;
case 435:
    console->appendPlainText("Dar un paseo");
    estado = 4;
    setEstado4();
    break;
case 441:
    console->appendPlainText("Encender TV");
    estado = 4;
    setEstado4();
    break;
case 442:
    console->appendPlainText("Apagar TV");
    estado = 4;
    setEstado4();
    break;
case 443:
    console->appendPlainText("Escuchar música");
    estado = 4;
    setEstado4();
    break;
case 444:
```

```
            console->appendPlainText("Leer");
            estado = 4;
            setEstado4();
            break;
        case 445:
            console->appendPlainText("Llamar a alguien");
            estado = 4;
            setEstado4();
            break;
        case 451:
            console->appendPlainText("Por determinar");
            estado = 4;
            setEstado4();
            break;
        case 452:
            console->appendPlainText("Por determinar");
            estado = 4;
            setEstado4();
            break;
        case 453:
            console->appendPlainText("Por determinar");
            estado = 4;
            setEstado4();
            break;
        case 454:
            console->appendPlainText("Por determinar");
            estado = 4;
            setEstado4();
            break;
        case 455:
            console->appendPlainText("Por determinar");
            estado = 4;
            setEstado4();
            break;

    }
}


////////////////////////////////////////////////////////////////////////////
/// SLOTS: ESTÍMULOS DETECTADOS
////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////
/// Función Estimulo1Detectado()
///     En función del estado en el que se encuentre cuando se detecta el
///     estímulo se invocará al estado correspondiente de la máquina de estados.
///     En esta función se selecciona la opción contenida en el estímulo
///     superior izquierdo.
////////////////////////////////////////////////////////////////////////////
void MainWindow::Estimulo1Detectado()
{

    switch (estado)
    {
        case 1:
            estado=11;
            break;
        case 2:
            estado=21;
```

```
            break;
        case 3:
            estado=31;
            break;
        case 4:
            estado=41;
            break;
        case 10:
            estado=1;
            break;
        case 21:
            estado=211;
            break;
        case 22:
            estado=221;
            break;
        case 23:
            estado=231;
            break;
        case 24:
            estado=241;
            break;
        case 25:
            estado=251;
            break;
        case 41:
            estado=411;
            break;
        case 42:
            estado=421;
            break;
        case 43:
            estado=431;
            break;
        case 44:
            estado=441;
            break;
        case 45:
            estado=451;
            break;
    }

    if (estado==11 && contador12luces!=0 && mirarluz!=1){
        console->appendPlainText("Luz incorrecta");
        numero_fallos++;

    }  else
        checkEstados(estado);
}


////////////////////////////////////////////////////////////////////////////
/// Función Estimulo2Detectado()
///     En función del estado en el que se encuentre cuando se detecta el
///     estímulo se invocará al estado correspondiente de la máquina de estados.
///     En esta función se selecciona la opción contenida en el estímulo
///     superior central.
////////////////////////////////////////////////////////////////////////////
void MainWindow::Estimulo2Detectado(){

    switch (estado)
```

```
    {
        case 1:
            estado=12;
            break;
        case 2:
            estado=22;
            break;
        case 3:
            estado=32;
            break;
        case 4:
            estado=42;
            break;
        case 10:
            estado=2;
            break;
        case 21:
            estado=212;
            break;
        case 22:
            estado=222;
            break;
        case 23:
            estado=232;
            break;
        case 24:
            estado=242;
            break;
        case 25:
            estado=252;
            break;
        case 41:
            estado=412;
            break;
        case 42:
            estado=422;
            break;
        case 43:
            estado=432;
            break;
        case 44:
            estado=442;
            break;
        case 45:
            estado=452;
            break;

    }

    if (estado==11 && contador12luces!=0 && mirarluz!=2){
        console->appendPlainText("Luz incorrecta");
        numero_fallos++;

    }  else
        checkEstados(estado);
}
```

```
////////////////////////////////////////////////////////////////////////////
/// Función Estimulo3Detectado()
///     En función del estado en el que se encuentre cuando se detecta el
///     estímulo se invocará al estado correspondiente de la máquina de estados.
///     En esta función se selecciona la opción contenida en el estímulo
///     superior derecho.
////////////////////////////////////////////////////////////////////////////
void MainWindow::Estimulo3Detectado(){

    switch (estado)
    {
        case 1:
            estado=13;
            break;
        case 2:
            estado=23;
            break;
        case 3:
            estado=33;
            break;
        case 4:
            estado=43;
            break;
        case 10:
            estado=3;
            break;
        case 21:
            estado=213;
            break;
        case 22:
            estado=223;
            break;
        case 23:
            estado=233;
            break;
        case 24:
            estado=243;
            break;
        case 25:
            estado=253;
            break;
        case 41:
            estado=413;
            break;
        case 42:
            estado=423;
            break;
        case 43:
            estado=433;
            break;
        case 44:
            estado=443;
            break;
        case 45:
            estado=453;
            break;
    }

    if (estado==11 && contador12luces!=0 && mirarluz!=3){
        console->appendPlainText("Luz incorrecta");
        numero_fallos++;
```

```
    }  else
        checkEstados(estado);


}

////////////////////////////////////////////////////////////////////////////
/// Función Estimulo4Detectado()
///     En función del estado en el que se encuentre cuando se detecta el
///     estímulo
///     se invocará al estado correspondiente de la máquina de estados.
///     En esta función se selecciona la opción contenida en el estímulo
///     inferior izquierdo.
////////////////////////////////////////////////////////////////////////////
void MainWindow::Estimulo4Detectado(){

    switch (estado)
    {
        case 1:
            estado=14;
            break;
        case 2:
            estado=24;
            break;
        case 3:
            estado=34;
            break;
        case 4:
            estado=44;
            break;
        case 10:
            estado=4;
            break;
        case 21:
            estado=214;
            break;
        case 22:
            estado=224;
            break;
        case 23:
            estado=234;
            break;
        case 24:
            estado=244;
            break;
        case 25:
            estado=254;
            break;
        case 41:
            estado=414;
            break;
        case 42:
            estado=424;
            break;
        case 43:
            estado=434;
            break;
        case 44:
            estado=444;
            break;
```

```
        case 45:
            estado=454;
            break;
    }

    if (estado==11 && contador12luces!=0 && mirarluz!=4){ //Si al mirar las
luces has mirado a la que no es
        console->appendPlainText("Luz incorrecta");
        numero_fallos++;

    }  else
        checkEstados(estado);
}

////////////////////////////////////////////////////////////////////////////
/// Función Estimulo5Detectado()
///     En función del estado en el que se encuentre cuando se detecta el
///     estímulo
///     se invocará al estado correspondiente de la máquina de estados.
///     En esta función se selecciona la opción contenida en el estímulo
///     inferior central.
////////////////////////////////////////////////////////////////////////////
void MainWindow::Estimulo5Detectado(){
    switch (estado)
    {
        case 1:
            estado=15;
            break;
        case 2:
            estado=25;
            break;
        case 3:
            estado=35;
            break;
        case 4:
            estado=45;
            break;
        case 10:
            estado=5;
            break;
        case 21:
            estado=215;
            break;
        case 22:
            estado=225;
            break;
        case 23:
            estado=235;
            break;
        case 24:
            estado=245;
            break;
        case 25:
            estado=255;
            break;
        case 41:
            estado=415;
            break;
        case 42:
            estado=425;
            break;
```

```
        case 43:
            estado=435;
            break;
        case 44:
            estado=445;
            break;
        case 45:
            estado=455;
            break;
    }

    if (estado==11 && contador12luces!=0 && mirarluz!=5){
        console->appendPlainText("Luz incorrecta");
        numero_fallos++;

    }  else
        checkEstados(estado);
}

////////////////////////////////////////////////////////////////////////////
/// Función Estimulo6Detectado()
///     En función del estado en el que se encuentre cuando se detecta el
///     estímulo se invocará al estado correspondiente de la máquina de estados.
///     En esta función se selecciona la opción contenida en el estímulo
///     inferior derecho.
////////////////////////////////////////////////////////////////////////////
void MainWindow::Estimulo6Detectado(){

    console->appendPlainText(QString::number(estado));
        switch (estado)
        {
            case 1:
                estado=16;
                break;
            case 10:
                setInicio();
                break;

            case 11:
                if (contador12luces!=0 && mirarluz!=6){
                    console->appendPlainText("Luz incorrecta");
                    numero_fallos++;
                }
                else
                    checkEstados(estado);
                break;
            case 21:
                estado=216;
                checkEstados(estado);
                break;
            case 22:
                estado=226;
                checkEstados(estado);
                break;
            case 23:
                estado=236;
                checkEstados(estado);
                break;
            case 24:
                estado=246;
                checkEstados(estado);
```

```
                break;
            case 25:
                estado=256;
                checkEstados(estado);
                break;
            default:
                setEstado6();
                break;
        }

}
```

6.1.2.2 MyThread.cpp

```
#include "mythread.h"
#include "mainwindow.h"

//La variable idCola se recibe desde el Main
extern int idCola;

//Los valores de frecuencias configuradas se reciben desde MainWindow
extern int frecuencia1_int;
extern int frecuencia2_int;
extern int frecuencia3_int;
extern int frecuencia4_int;
extern int frecuencia5_int;
extern int frecuencia6_int;

void MyThread::run(){

    //Declaración de variables
    MainWindow * w = MainWindow::getInstance();
    MensajeRobot mensaje;
    int errorTransmision;
    int contador[MAX_ESTIMULOS]; contador[0] = 0; contador[1] = 0; contador[2] =
0; contador[3] = 0; contador[4] = 0; contador[5] = 0;
    int ultima_freq = 0;
    QString idCola_string = QString::number(idCola);
    //Se verifica que el id de Cola es el mismo en Main y en MyThread
    qDebug("El id de cola en el hilo es" + idCola_string.toLatin1());

    //El hilo lee de la cola indefinidamente e interpreta los resultados
    while(1){

        w->delay(2);
        // Lectura del mensaje de la cola
        errorTransmision = msgrcv(idCola, &mensaje, sizeof(mensaje), 0,
MSG_NOERROR);
        if (errorTransmision < 0) {
            qDebug("Recepcion de datos fallida ");
        }

        // Si viene una frecuencia diferente a la anterior, se resetean los
contadores
        if(mensaje.ComandoRobot != ultima_freq){
            contador[0] = 0;
            contador[1] = 0;
```

```
    contador[2] = 0;
    contador[3] = 0;
    contador[4] = 0;
    contador[5] = 0;
}

ultima_freq=mensaje.ComandoRobot;

// Se analiza la información según el tipo de dato almacenado en la cola
switch(mensaje.tipoComunicacion){

case IPC_DATOS: // Mensaje de datos
{

    //Recepción del mensaje
    int dato = mensaje.ComandoRobot;
    QString dato_string = QString::number(dato);
    qDebug("La frecuencia recibida es" + dato_string.toLatin1());

    //Se realiza la comparación de frecuencia
    if (frecuencia1_int==dato){
        contador[0]++;
        if (contador[0]==LIMITE_CONTADOR){
            qDebug("SEÑAL EMITIDA!!!");
            emit Signal_Thread_Estimulo1();
            contador[0]=0;
        }

    }

    else if (frecuencia2_int==dato){
        contador[1]++;
        if (contador[1]==LIMITE_CONTADOR){
            qDebug("SEÑAL EMITIDA!!!");
            emit Signal_Thread_Estimulo2();
            contador[1]=0;
        }
    }

    else if (frecuencia3_int==dato){
        contador[2]++;
        if (contador[2]==LIMITE_CONTADOR){
            qDebug("SEÑAL EMITIDA!!!");
            emit Signal_Thread_Estimulo3();
            contador[2]=0;
        }
    }

    else if (frecuencia4_int==dato){
        contador[3]++;
        if (contador[3]==LIMITE_CONTADOR){
            qDebug("SEÑAL EMITIDA!!!");
            emit Signal_Thread_Estimulo4();
            contador[3]=0;
        }
    }

    else if (frecuencia5_int==dato){
        contador[4]++;
        if (contador[4]==LIMITE_CONTADOR){
            qDebug("SEÑAL EMITIDA!!!");
```

```
                    emit Signal_Thread_Estimulo5();
                    contador[4]=0;
                }
            }

            else if (frecuencia6_int==dato){
                contador[5]++;
                if (contador[5]==LIMITE_CONTADOR){
                    qDebug("SEÑAL EMITIDA!!!");
                    emit Signal_Thread_Estimulo6();
                    contador[5]=0;
                }
            }
        }
    }
  }
}
```