UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN
MENCIÓN EN SISTEMAS DE TELECOMUNICACIÓN

Escribiendo mediante las ondas cerebrales: desarrollo y evaluación de paradigmas utilizando OpenViBE

Autor:

Dña. Cristina Cantalapiedra Cabezas

Tutores:

D. Víctor Martínez Cagigal
Dr. D. Roberto Hornero Sánchez

Valladolid, 3 de Septiembre de 2018

TÍTULO:	**Escribiendo mediante las ondas cerebrales: desarrollo y evaluación de paradigmas utilizando OpenViBE**
AUTOR:	**Dña. Cristina Cantalapiedra Cabezas**
TUTORES:	**D. Víctor Martínez Cagigal** **Dr. D. Roberto Hornero Sánchez**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Roberto Hornero Sánchez**
VOCAL:	**Dr. D. Jesús Poza Crespo**
SECRETARIO:	**Dr. D. Carlos Gómez Peña**
SUPLENTE 1:	**Dra. Dña. María García Gadañón**
SUPLENTE 2:	**Dr. D. Miguel López-Coronado Sánchez-Fortún**

FECHA:	**3 de Septiembre de 2018**
CALIFICACIÓN:	

RESUMEN

El electroencefalograma (EEG) fue desarrollado por Hans Berger en 1929 y, a partir de ello, se comenzaron a investigar las ondas cerebrales como método diagnóstico de enfermedades como la epilepsia o el trastorno del sueño. Desde su invención, se especuló que el EEG podría utilizarse para desarrollar un sistema de comunicación entre el cerebro y el medio sin que intervinieran los intermediarios normales (nervios y músculos periféricos). A este sistema se le llamó Brain Computer Interface (BCI) y fue iniciado por el Dr. Jacques Vidal en 1977. Su función principal es monitorizar la actividad cerebral y traducir determinadas características, correspondientes a las intenciones del usuario, en comandos de un dispositivo.

El objetivo de este Trabajo Fin de Grado es diseñar, desarrollar y evaluar un conjunto de aplicaciones enfocadas a la selección de caracteres con las ondas cerebrales del usuario utilizando la plataforma *OpenViBE*. Este estudio está enfocado a un uso futuro por parte de personas con grave discapacidad.

Inicialmente, se realiza un estudio sobre los sistemas BCI, los métodos de registro de la actividad cerebral, las señales de control utilizadas por estos sistemas, el procesado de la señal y diferentes paradigmas *odd-ball*. A continuación, se profundiza en el análisis de la señal y los potenciales evocados P300, señal de control utilizada en las aplicaciones a desarrollar. En este análisis, se describen los artefactos más comunes que degradan la señal y los distintos métodos de extracción y traducción de características. Entre los métodos de extracción destacan el análisis temporal, frecuencial y espacial; y con respecto a los métodos de traducción de características se emplean tanto los lineales como los no lineales.

A continuación, se muestra el diseño y el desarrollo de las aplicaciones BCI utilizadas para escribir mediante las ondas cerebrales. Estas aplicaciones utilizan los potenciales evocados P300 como señal de control gracias al paradigma *odd-ball*. Los paradigmas odd-ball presentan un estímulo deseado de forma infrecuente entre estímulos frecuentes, que generan un potencial P300 cada vez que se percibe el estímulo deseado, seleccionando un comando de la aplicación.

Para finalizar, se realiza una comparativa entre 4 paradigmas, el paradigma *Row-Column* (RCP), el paradigma *Checkerboard* (CBP), el paradigma *Honeycomb-Shaped Red-Dots* (HSRD) y el paradigma unificado *Honeycomb-Shaped Red-Dots – Checkerboard* (HSRDCBP); para buscar cuál de ellos es el más adecuado. Las aplicaciones se evaluaron por parte de cinco sujetos sanos en cuatro sesiones consecutivas utilizando la plataforma *OpenViBE*. A partir de la evaluación se puede concluir que el HSRDCBP es el paradigma desarrollado con mejores resultados, con un 100% de precisión. El paradigma tradicional obtiene casi un 93% de precisión, superado por el paradigma HSRD con un 98%. El paradigma CBP solamente obtiene un 85% de precisión.

Palabras clave

Brain Computer Interface (BCI), Checkerboard Paradigm (CBP), electroencefalograma (EEG), Honeycomb-Shaped Red-Dots Paradigm (HSRD), Honeycomb-Shaped Red-Dots – Checkerboard Paradigm (HSRDCBP), paradigma odd-ball, potenciales evocados P300.

ABSTRACT

Electroencephalogram (EEG) was developed by Hans Berger in 1929 and, as a result, brain signals were researched in order to help the diagnosis of epilepsy or sleeping disorders. Since its invention, researchers speculated that these signals might be used for developing a communication and control system between brain and environment without using the normal intermediaries (peripheral nerves and muscles). This system was called Brain Computer Interface (BCI) and it was originally developed by Dr. Jacques Vidal in 1977. Its main function is to monitor the brain activity and translate certain features, which reflect user's intentions, into device commands.

The objective of this Final Degree Project is to design, develop and evaluate a set of applications focused on the selection of characters with the user's brain waves using the OpenViBE platform. This study is aimed to be used by severely disabled people in a near feature.

Firstly, a study about BCI systems, methods of recording brain activity, controls signals used by this systems, signal processing and different *odd-ball* paradigms is carried out. Signal analysis and P300 evoked potentials, the selected control signal, are deeply studied next. In this analysis, common artifacts that degrade the EEG signal and several feature extraction and translation methods are described. Several feature extraction methods are included spatial, temporal and spectral analysis. Regarding the feature translation, lineal and non-lineal methods are discussed.

Secondly, design and development of BCI applications used to spell using only the brain waves is shown. Based on the odd-ball paradigm, both applications use the P300 evoked potentials as control signals. This paradigm is based on presenting an infrequent target stimuli among more frequent distracting stimuli. Hence, a P300 potential is generated whenever the subject detects the tarjet stimuli, selecting an application command.

Finally, a comparison is made between 4 paradigms, the *Row-Column* Paradigm (RCP), the *Checkerboard* Paradigm (CBP), the *Honeycomb-Shaped Red-Dots* Paradigm (HSRD) and the *Honeycomb-Shaped Red-Dots – Checkerboard* Paradigm (HSRDCBP); to find which one is the most appropriate. The applications were evaluated by five healthy subjects in four consecutive sessions using the OpenViBE platform. From the evaluation, it can be concluded that the HSRDCBP is the paradigm that achieved the highest accuracy (100%). The HSRD paradigm reached accuracy of 98% and the CBP paradigm almost 85% accuracy.

Keywords

Brain Computer Interface (BCI), Checkerboard Paradigm (CBP), electroencephalogram (EEG), Honeycomb-Shaped Red-Dots Paradigm (HSRD), odd-ball paradigm, P300 evoked potential

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a Roberto Hornero la oportunidad que me ha ofrecido al poder realizar este trabajo y, en general, al Grupo de Ingeniería Biomédica por descubrirme un mundo al que quiero dedicar mi vida.

A Víctor Martínez, por ayudarme a desarrollar este trabajo y por la paciencia prestada en todo momento. Ha sido un placer trabajar contigo.

A mis amigos y compañeros, por todos estos años de apoyo, por estar en los buenos y en los malos momentos y por ayudarme a desconectar cuando lo he necesitado. En especial, me gustaría agradecer a mi grupo de amigos de teleco todo lo que hemos compartido, habéis hecho que estos años hayan sido un viaje inolvidable.

A Dani, por todo lo que has hecho y haces siempre por mí. Si he llegado hasta aquí, es gracias a ti.

Para finalizar, me gustaría agradecer a mis padres el esfuerzo realizado en formarme y el apoyo prestado en esta etapa de mi vida que por fin termina; y a mi hermana, por compartir codo con codo todos estos años de caídas.

Muchas gracias a todos.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

INTRODUCCIÓN

SISTEMAS BRAIN COMPUTER INTERFACE

ÍNDICE DE TABLAS

CAPÍTULO I

INTRODUCCIÓN

1. Señales biomédicas

La Ingeniería Biomédica es un campo multidisciplinar que aplica las técnicas y métodos de la ingeniería a la comprensión, definición y resolución de problemas de las especialidades relacionadas con la medicina y la biología. Se encarga de procesar las señales producidas por los sistemas biológicos para obtener información relevante y útil para el diagnóstico. Asimismo, estudia y busca la aplicación de principios y métodos de la ingeniería para entender, modificar o controlar sistemas biológicos, así como para diseñar y fabricar productos que sirvan para monitorizar funciones fisiológicas y asistir en el diagnóstico y el tratamiento de pacientes.

Las señales biomédicas se pueden clasificar siguiendo varios criterios:

- **Según el origen de las señales y el medio utilizado para la adquisición.** Dentro de esta clasificación, las señales biomédicas pueden ser señales eléctricas, impedancias, señales acústicas, señales mecánicas, señales biomagnéticas, señales bioquímicas o imágenes biomédicas.
    - **Señales eléctricas.** Las señales eléctricas son las señales más utilizadas, ya que recogen la actividad eléctrica de las diferentes partes del cuerpo humano. Algunas de las señales eléctricas más importantes son el electroencefalograma (EEG) y el electrocorticograma (ECoG), que recogen la actividad eléctrica del cerebro; el electrocardiograma (ECG), que recoge la actividad del corazón; el electrooculograma (EOG), que recoge la actividad eléctrica de los ojos; y, por último, el electromiograma (EMG), que recoge la actividad muscular. En la Figura 1-1 se muestran las formas de 3 de estas señales. Debido a que el EEG es la señal utilizada para registrar la actividad cerebral en este trabajo, se estudiará con más detalle en el siguiente apartado. Para ello se mostrarán sus principales características, cuál es la forma en la que se registra la señal y cuáles son los artefactos que la contaminan.
    - **Señales biomagnéticas.** En cuanto a las señales biomagnéticas, destaca el magnetoencefalograma (MEG), que se encarga de registrar la actividad cerebral utilizando campos magnéticos, lo que permite estudiar las relaciones entre las estructuras cerebrales y sus funciones.
    - **Señales bioquímicas.** Las señales bioquímicas son el resultado de mediciones químicas de los tejidos vivos o de muestras analizadas en el laboratorio clínico.
    - **Imágenes biomédicas.** En cuanto a las imágenes biomédicas, se obtienen utilizando técnicas que emplean diversos principios físicos como la imagen por resonancia magnética funcional (fMRI), que permite localizar las regiones cerebrales funcionales; la tomografía por emisión de positrones (PET), que mide la actividad metabólica de los tejidos del cuerpo humano; o la retinografía, que permite visualizar el estado de la retina para diagnosticar enfermedades como la retinopatía diabética

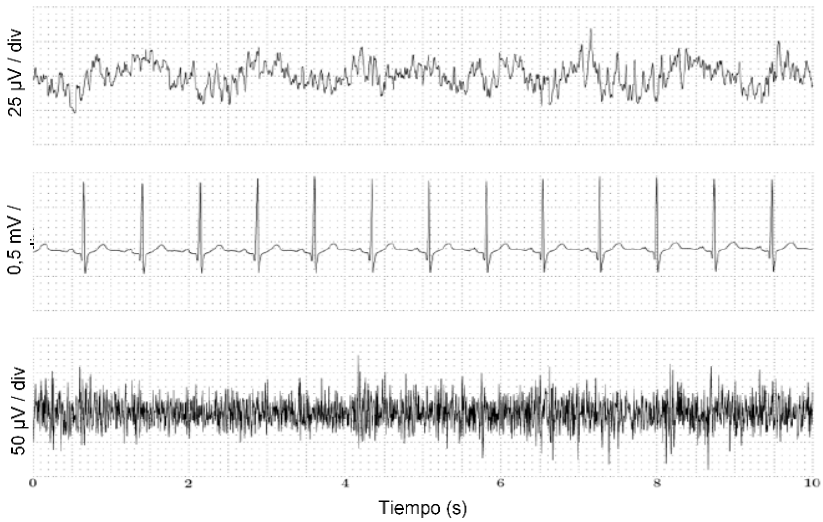


Figura 1-1. Señales eléctricas recogidas de diferentes partes del cuerpo: electroencefalograma (EEG), electrocardiograma (ECG) y electromiograma (EMG).

- **Según su descripción matemática.** Según su descripción matemática, las señales pueden ser deterministas o aleatorias [1].
    - **Señales deterministas.** Las señales deterministas son aquellas señales que pueden representarse matemáticamente de forma explícita mediante una expresión cerrada. Los valores futuros de la señal son predecibles conociendo cuáles son sus valores pasados. Dentro de la clasificación de señales deterministas, las señales biomédicas pueden ser periódicas, cuasi-periódicas o transitorias.
    - **Señales aleatorias.** Las señales aleatorias no permiten conocer cuál serán sus valores futuros con exactitud, aunque se conozcan los valores pasados. Las señales aleatorias pueden ser estacionarias o ergódicas.
        - **Señal estacionaria.** Una señal aleatoria será estacionaria cuando el comportamiento estadístico de la señal no cambie a lo largo del tiempo. Se pueden distinguir dos grados de estacionariedad, en sentido amplio y en sentido estricto.
        - **Señal ergódica.** Una señal aleatoria será ergódica si toda su aleatoriedad está presente en cualquiera de sus realizaciones y, por tanto, consideramos que una realización es representativa de todas las demás.
- **Según sus características.** Las señales biomédicas se pueden clasificar según su frecuencia o su voltaje.

Antes de poder analizar la señal, ésta debe ser tratada. El procesado de las señales biomédicas se divide en 3 etapas:

- **Adquisición de la señal**. En esta primera etapa, se realiza la detección, el muestreo, la cuantificación y digitalización de la señal. Además, se realiza un preprocesado de la señal para eliminar los artefactos que puedan estar contaminándola; y, después, se procede al almacenamiento y/o la transmisión de la señal preprocesada.

- **Procesado**. En el procesado, primero se realiza la segmentación de la señal. Después, se filtra y/o transforma y, por último, se determinan los patrones que se quieren detectar. Posteriormente, se extraen las características y se procede a la reducción y posterior selección de éstas. Finalmente, se realiza la clasificación de señal.

- **Aplicación.** La aplicación utiliza las características extraídas y clasificadas en el procesado de la señal, con el objetivo de controlar un dispositivo haciendo uso de estas.

Los objetivos del procesado de las señales biomédicas son reducir la subjetividad de las medidas manuales, mejorar la calidad de las señales registradas, mejorar la precisión de la medida, visualizar eventos complejos, conseguir que los dispositivos médicos sean más inteligentes y facilitar futuras investigaciones.

2. Electroencefalograma (EEG)

El electroencefalograma (*electroencephalogram*, EEG) es una señal biomédica que permite registrar la actividad eléctrica del cerebro de forma no invasiva, utilizando electrodos situados sobre el cuero cabelludo del usuario [2].

La señal EEG es la agrupación de la actividad de millones de neuronas registrada al mismo tiempo, ya que debido a la atenuación que sufre la señal por las diferentes capas que separan el córtex y los electrodos situados sobre el cuero cabelludo, no es posible medir la actividad de una sola neurona mediante el uso del EEG.

Esta señal se caracteriza por ser oscilatoria y repetitiva, por lo que suele denominarse ritmo y se clasifica según su rango frecuencial y su amplitud relativa. La amplitud del EEG está relacionada con el grado de sincronía con el que interactúan las neuronas y su valor oscila entre los 5 y los 200 μV [3].

El rango frecuencial está comprendido por 5 bandas de frecuencia que van desde los 0 a los 100 Hz [4]. En la Figura 1-2, se observan los diferentes ritmos dependiendo de cuál sea la banda de frecuencia analizada. Como se puede observar, el rango frecuencial de la señal EEG está relacionado con su amplitud y depende de la actividad que se esté realizando.

- **Banda delta.** La banda delta abarca las frecuencias inferiores a 4 Hz. Se trata de frecuencias que aparecen cuando el sujeto se encuentra en sueño profundo, en sujetos con la respiración forzada (hiperventilación) y, también, en sujetos con enfermedades orgánicas cerebrales graves. Estas frecuencias predominan en la zona frontal o parieto-occipital.
- **Banda theta.** La banda theta está comprendida entre 4 y 8 Hz. En esta banda se encuentra la principal componente en el EEG infantil. Se trata de frecuencias que aparecen al mantener los ojos abiertos, al dormirse y con cansancio; en sujetos con la

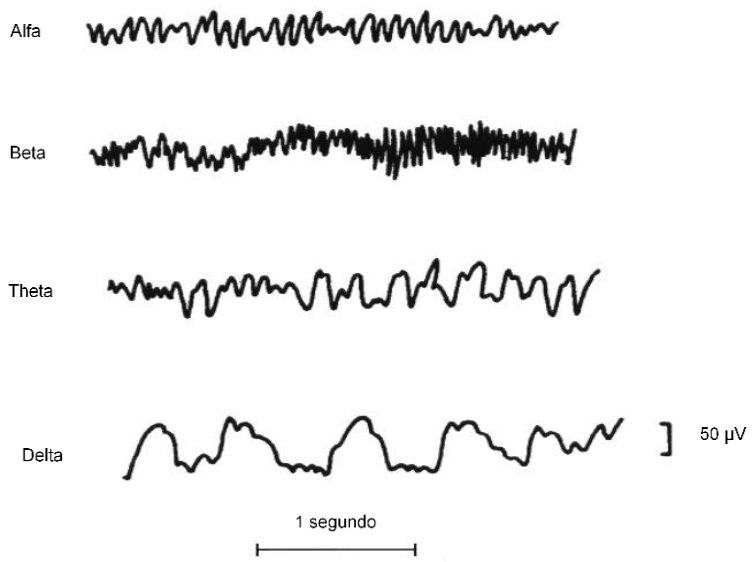


Figura 1-2. Ritmos electroencefalográficos observados en las diferentes bandas del rango frecuencial [4].

respiración forzada y, también, en sujetos con ligeras alteraciones generales. Estas frecuencias predominan en la zona occipital.

- **Banda alfa.** La banda alfa está comprendida entre 8 y 13 Hz. Se trata de frecuencias que aparecen en sujetos despiertos, sin ninguna actividad y con los ojos cerrados; en sujetos con estado de conciencia meditativa, en el sueño REM y, también, en sujetos con intoxicaciones farmacológicas y en estado comatoso. Estas frecuencias predominan en la zona occipital.
- **Banda beta.** La banda beta está comprendida entre 13 y 30 Hz. Se trata de frecuencias que aparecen en estado de vigilia con los ojos cerrados en adultos, al abrir los ojos o cuando hay mala relajación, y al dormirse. También aparecen bajo la administración de medicamentos, en ciertos estadios comatosos y en relación con algunas enfermedades internas. Esta banda está formada por las ondas β_1 (frecuencias entre 13 y 18 Hz) y β_2 (frecuencias entre 18 y 30 Hz). Las ondas β_2 aparecen cuando se activa intensamente el sistema nervioso central o cuando el sujeto está bajo tensión.
- **Banda gamma.** La banda gamma abarca las frecuencias superiores a 30 Hz. Se encuentra en fase de investigación.

Cuando el sujeto está en estado de somnolencia, se producirá una sincronía entre neuronas que producirá bajas frecuencia y elevada amplitud. Sin embargo, cuando el sujeto está concentrado en una tarea, se produce una asincronía entre neuronas que produce altas frecuencias y baja amplitud. Se aprecia un aumento en la amplitud a medida que el sujeto se duerme [4].

Los electrodos son puntos de contacto con el cuero cabelludo a través de los que se capta la señal. La colocación de dichos electrodos sobre el cuero cabelludo se realiza atendiendo al sistema estándar de registro llamado Sistema Internacional 10-20 [4], respaldado por la Federación Internacional de Sociedades de Electroencefalografía y Neurofisiología Clínica. Este estándar utiliza una serie de marcas sobre el cráneo, equivalentes a puntos de referencia para colocar los electrodos.

Los diferentes electrodos se identifican utilizando una nomenclatura en la que la letra identifica la sección del cráneo y el subíndice indica el hemisferio en el que se encuentra el electrodo. Al hemisferio izquierdo se le asignan números impares, mientras que al hemisferio derecho se le asignan números pares. Si se trata de la región en la que convergen ambos hemisferios, el subíndice utilizado es la letra 'z' (*zero*).

Las regiones en las que se divide el cráneo son: frontopolar (Fp), frontal (F), temporal (T), central (C), parietal (P) y occipital (O). Como se muestra en la Figura 1-3, se toma como medida de referencia la distancia entre el entrecejo, también llamado nasión; y la nuca, o inión, del sujeto. La sección central se encontrará a mitad de distancia y se separará de las secciones contiguas (frontal y parietal) un 20% de la distancia total. A su vez, las dos secciones estarán separadas de las seccciones frontopolar y occipital, respectivamente, un 10% de la longitud total [4].

Debido a que la señal EEG está contaminada por diversas fuentes de ruido, tanto externas como internas, es necesario aplicar un filtrado espacial antes de procesar la señal. El electrodo situado en la oreja, utilizado como referencia, elimina gran parte de estas componentes. Aun así, además de dicho filtrado, se utilizan canales bipolares, métodos de media común o filtros Laplacianos para llevar a cabo este proceso. Los canales bipolares toman la diferencia de potencial entre dos electrodos próximos. Los métodos de referencia de medida común (*Common Average Reference*, CAR) realizan la diferencia entre la señal recogida por un electrodo y la media de todo el conjunto de electrodos. Por último, el filtro Laplaciano realiza la diferencia entre la señal recogida por un electrodo y las señales adyacentes.

Además de las componentes de ruido que contaminan a la señal EEG, existen otras señales diferentes que contaminan a la señal de interés. Los artefactos más comunes son el movimiento de los ojos y el movimiento de los músculos [5].

El EEG se utiliza para el diagnóstico de enfermedades como la epilepsia; para la detección, localización y medida de la extensión de una lesión en casos como contusiones cerebrales, tumores cerebrales, hematomas subdurales crónicos y abscesos (inflamación); en estados de coma; medir el grado de anestesia durante una intervención quirúrgica; en estudios sobre trastornos del sueño y en la detección de intoxicaciones con medicamentos.

3. Brain Computer Interface (BCI)

Un sistema Interfaz Cerebro-Computadora (*Brain Computer Interface,* BCI) es un sistema que monitoriza la actividad cerebral y traduce determinadas características, correspondientes a las intenciones del usuario, en comandos de un dispositivo. Estos sistemas establecen un sistema de comunicación entre el cerebro y el medio sin la intervención de los mecanismos normales, como son los nervios y los músculos periféricos [2].

Habitualmente, los sistemas BCI procesan el EEG del usuario de forma no invasiva, pero también existen sistemas BCI que utilizan métodos invasivos para el registro de la actividad cerebral. Algunos métodos invasivos son los ECoG (*electrocorticography*, ECoG) o los electrodos intracorticales, que consiguen mayor resolución a cambio de suponer un riesgo para el paciente.

Para interpretar y traducir las intenciones del usuario en comandos de un dispositivo, los sistemas BCI utilizan señales de control. Las señales de control pueden ser potenciales evocados

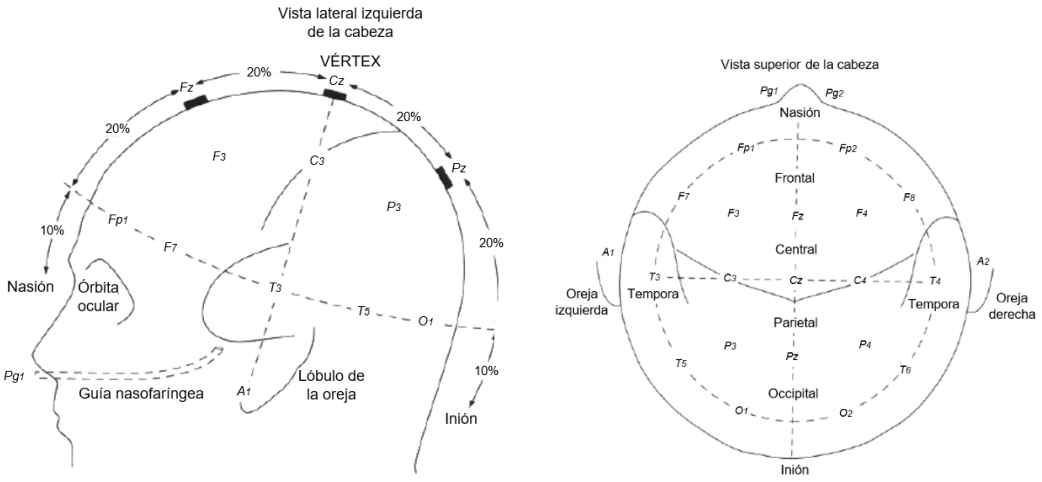


Figura 1-3. Distribución de los electrodos según el Sistema Internacional 10-20 [4].

visuales (*Visual Evoked Potentials*, VEP), potenciales corticales lentos (*Slow Cortical Potentials*, SCP), ritmos sensoriomotores (*ritmos μ y β*) o potenciales evocados P300 [5].

Los sistemas BCI pueden ser de dos tipos: endógenos o exógenos. Los sistemas endógenos son aquellos sistemas que no necesitan de ningún tipo de estimulación externa para generar actividad cerebral necesaria para clasificar las intenciones del usuario. Estos sistemas requieren entrenamiento para poder controlar la aplicación BCI. En estos sistemas se suelen utilizar señales de control como los SCP y los ritmos sensoriomotores.

Por otro lado, los sistemas BCI exógenos son aquellos sistemas que requieren una estimulación externa para producir actividad cerebral necesaria para clasificar las intenciones del usuario. Estos sistemas no requieren entrenamiento, ya que utilizan la respuesta natural del cerebro ante un estímulo externo. En estos sistemas se suelen utilizar señales de control como los VEP y los potenciales evocados P300.

Antes de poder traducir cuáles son las intenciones del usuario, es necesario tratar la señal. El tratamiento de la señal está compuesto por 3 etapas: la adquisición de la señal, el procesado y la aplicación. En la primera etapa se obtiene la señal, se amplifica y se digitaliza. Además, se eliminan los artefactos que puedan estar contaminándola [5].

En la etapa de procesado, la señal se hace pasar por dos nuevas etapas, la extracción y clasificación de características. En la extracción de características, la señal se somete a una serie de procedimientos con el objetivo de extraer aquellas características que codifican las intenciones del usuario. En la etapa de clasificación, el algoritmo de traducción transforma estas características en comandos que identifican la intención del usuario. El tratamiento de la señal se tratará con más detalle en el segundo, tercer y cuarto capítulo.

La motivación de los sistemas BCI es aumentar la independencia y la capacidad de comunicación de usuarios con grave discapacidad o con enfermedades neurodegenerativas. Estos sistemas pueden servir de ayuda a personas con enfermedades como la esclerosis amiotrófica lateral (*Amyotrophic Lateral Sclerosis*, ELA) o la distrofia muscular; o con parálisis, lesiones medulares,

lesiones cerebrales o amputaciones. Por lo tanto, las aplicaciones de los sistemas BCI abarcan diferentes ámbitos, como el control de prótesis, el control domótico o videojuegos.

4. Objetivos del Trabajo Fin de Grado

El objetivo de este Trabajo Fin de Grado es diseñar, desarrollar y evaluar paradigmas *odd-ball*, capaces de evocar potenciales P300 a través de una estimulación visual. Esto va a permitir escribir mediante las ondas cerebrales a personas con algún tipo de discapacidad o enfermedad neurodegenerativa. Para ello, se va a utilizar el EEG, un método no invasivo de registro de la actividad cerebral y de bajo coste, que traduce las intenciones el usuario en comandos. Para el desarrollo de los paradigmas se ha utilizado la plataforma software *OpenViBE* (*Open Virtual Brain Environment*).

Para alcanzar el objetivo general, se deben cumplir los siguientes objetivos específicos:

1. Estudiar las distintas técnicas utilizadas en el registro de la actividad cerebral, los sistemas BCI y las señales de control que se utilizan para traducir las intenciones del usuario en comandos de un dispositivo.
2. Realizar un estudio de los posibles paradigmas *odd-ball* a implementar, así como una comparativa que refleje las principales características de cada uno de ellos.
3. Diseñar las matrices que generan los potenciales evocados P300 y desarrollar los paradigmas utilizando lenguaje C++ y la plataforma *OpenViBE*.
4. Evaluar las aplicaciones de selección de caracteres implementadas mediante nuevos paradigmas *odd-ball* utilizando cinco sujetos sanos en cuatro sesiones distintas, una sesión por cada paradigma.
5. Analizar y discutir los resultados obtenidos y realizar una comparativa con otros estudios.
6. Extraer conclusiones a partir de los resultados obtenidos.
7. Plantear las posibles líneas futuras de investigación dentro del ámbito de estudio.

5. Estructura del Trabajo Fin de Grado

El Trabajo Fin de Grado está formado por siete capítulos: introducción, sistemas Brain Computer Interface, análisis de los potenciales P300, desarrollo de paradigmas *odd-ball* con la plataforma *OpenViBE*, resultados, discusión y conclusiones.

En el primer capítulo, introducción, se realiza una descripción de los tipos de señales biomédicas y cómo se clasifican dependiendo de sus características. A continuación, se expone el concepto de señal EEG y se detallan cuáles son sus características. Además, se realiza una introducción a los sistemas BCI, se describen las señales de control utilizadas y cuál es la estructura que se lleva a cabo para realizar el procesado de la señal. Por último, se explican cuáles son los objetivos y la estructura del Trabajo Fin de Grado.

En el segundo capítulo, sistemas Brain Computer Interface, se detallan las principales características de los sistemas BCI que se han introducido en el capítulo anterior. Se describen los métodos utilizados para registrar la actividad cerebral, así como las distintas señales de control. A continuación, se explica el procesado de la señal EEG. Para finalizar, se enumeran las diversas aplicaciones en las que se utilizan los sistemas BCI y se realiza un estudio de los posibles paradigmas *odd-ball* a implementar en las aplicaciones de selección de caracteres propuestas en este trabajo.

En el tercer capítulo, análisis de los potenciales evocados P300, se detallan las características y propiedades de la señal de control utilizada en las aplicaciones de selección de caracteres, los potenciales evocados P300. A continuación, se realiza una introducción de los métodos más utilizados para procesar dicha señal de control, tanto en la extracción de características como en su posterior clasificación.

En el cuarto capítulo, desarrollo de paradigmas *odd-ball* con la plataforma *OpenViBE*, se describen los objetivos, la estructura, el diseño y el desarrollo de las aplicaciones de selección de caracteres desarrolladas. A continuación, se describe la plataforma *OpenViBE* y se muestra la guía de usuario de las aplicaciones. Además, se incluye el cuestionario de satisfacción entregado a los usuarios al finalizar las sesiones de evaluación. Por último, se detalla el procedimiento de evaluación utilizado, describiendo las diferentes tareas llevadas a cabo por los usuarios.

En el quinto capítulo, resultados, se muestran los resultados obtenidos en las aplicaciones de selección de caracteres. Además, se detallan las precisiones obtenidas por cada uno de los usuarios en las sesiones de evaluación. Por último, se incluyen las opiniones y sugerencias de mejora de las aplicaciones, recogidas en el cuestionario de satisfacción.

En el sexto capítulo, discusión, se explican y justifican los resultados obtenidos. Además, los resultados se comparan con los obtenidos a partir de otros paradigmas *odd-ball* y se exponen las limitaciones de las aplicaciones, junto con las sugerencias de mejora.

En el séptimo y último capítulo, conclusiones y líneas futuras, se resumen las conclusiones extraídas a partir del desarrollo del Trabajo Fin de Grado. Para finalizar, se establecen las posibles líneas futuras de investigación que se proponen a partir del mismo.

CAPÍTULO II

SISTEMAS BRAIN COMPUTER INTERFACE

1. Introducción

El desarrollo de los sistemas BCI se ha basado en la utilización de los EEG. El EEG fue desarrollado por Hans Berger en 1929 y, a partir de esto, se comenzaron a investigar las ondas cerebrales como método diagnóstico de enfermedades como la epilepsia o el trastorno del sueño. Desde su invención, se especuló que el EEG podría utilizarse para desarrollar un sistema de comunicación entre el cerebro y el medio sin que intervinieran los intermediarios normales (nervios y músculos periféricos) [2].

El primer sistema BCI realizado con éxito fue dirigido por el Dr. Jacques Vidal. En este proyecto, el usuario podía controlar el movimiento de un cursor en dos dimensiones utilizando los potenciales VEP [2]. Como se muestra en la Figura 2-1, el sistema estaba formado por 4 puntos (arriba, abajo, derecha e izquierda) que se iluminaban de forma intermitente a diferentes frecuencias. Cuando el usuario se fijaba en uno de los 4 puntos, en el EEG se reflejaban los VEP que aparecen de forma natural como respuesta a un estímulo visual. A partir de este esquema, el usuario conseguía controlar el cursor [3]. Sin embargo, aunque este sistema fue un proyecto innovador en el ámbito de los sistemas BCI, hasta los años 90 estos sistemas no se desarrollaron completamente debido al coste computacional que exigían las aplicaciones prácticas.

Un sistema BCI puede clasificarse según diferentes criterios. Dependiendo de los caminos de salida que se utilicen para llevar el mensaje, los sistemas BCI se pueden clasificar como sistemas dependientes o independientes. Un sistema BCI dependiente es aquel sistema que no utiliza los caminos de salida normales del cerebro, pero la actividad en estos caminos (músculos y nervios periféricos) es necesaria para que se genere la actividad cerebral que consigue llevar el mensaje. Por otro lado, los sistemas BCI independientes son aquellos sistemas que no dependen de los caminos de salida del cerebro. El mensaje no viaja por los músculos y nervios periféricos y, además, la actividad en estos caminos no es necesaria para generar la actividad cerebral [5].

Dependiendo del método que se utilice para registrar la actividad cerebral, se puede distinguir entre métodos invasivos o no invasivos. Los sistemas BCI invasivos son aquellos sistemas que necesitan una intervención quirúrgica para implantar los electrodos que registran la actividad cerebral. Estos sistemas no suelen utilizarse en humanos, puesto que implican un riesgo para el paciente, a cambio de obtener una mayor calidad de las señales registradas. Por otro lado, los sistemas BCI no invasivos son sistemas que no requieren la implantación de electrodos. Habitualmente, la actividad cerebral se registra mediante el EEG, debido a su sencillez y bajo coste, utilizando electrodos colocados en el cuero cabelludo del usuario. Estos sistemas son los más utilizados en la práctica [5].

Dependiendo de si necesitan estimulación o no, se pueden distinguir sistemas BCI endógenos o exógenos. Los sistemas BCI endógenos no necesitan ningún tipo de estimulación externa para generar la actividad cerebral necesaria para interpretar las intenciones del usuario, pero necesitan entrenamiento. Por otro lado, los sistemas BCI exógenos requieren una estimulación externa para producir la actividad cerebral necesaria. Como se trata de una respuesta natural del cerebro, este tipo de sistemas no necesitan entrenamiento o el tiempo de entrenamiento es muy bajo [5].

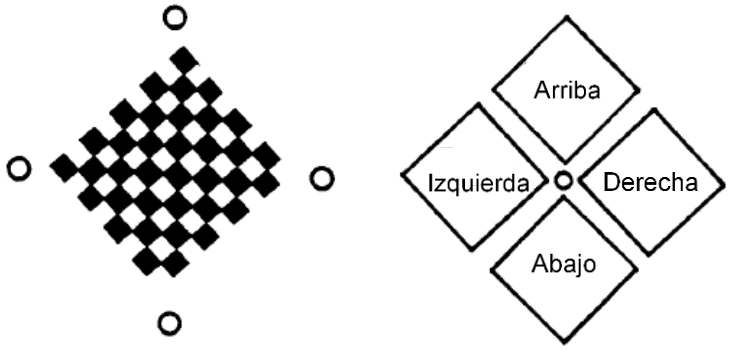


Figura 2-1. Esquema del primer sistema BCI controlado por VEP, desarrollado por el Dr. Jaques Vidal [3].

El objetivo de los sistemas BCI es mejorar la calidad de vida de personas con grave discapacidad, debida a una enfermedad neurodegenerativa (esclerosis lateral amiotrófica o distrofia neuromuscular) o debida a una parálisis o amputación (lesión cerebral o lesión medular). Estos sistemas ayudan a la comunicación o el control de dispositivos domóticos.

Un sistema BCI, como cualquier sistema de comunicación, está formado por una entrada, una salida, componentes que traducen la señal de entrada en la señal de salida y un protocolo que determina el comienzo, el offset y el tiempo de operación [5]. Como se puede observar en la Figura 2-2, la estructura de un sistema BCI estará comprendida por tres etapas, la adquisición de la señal, el procesado de ésta y, posteriormente, la aplicación.

En este capítulo se van a estudiar los métodos más utilizados para la adquisición de la señal y las diferentes señales que utilizan los sistemas BCI, profundizando en cuáles son sus principales características, así como sus ventajas e inconvenientes.

Las señales registradas poseen otro tipo de señales que contaminan a la señal de interés. A estas señales que contaminan se les denomina artefactos. Los artefactos más comunes que afectan a las señales son el movimiento de los ojos y el parpadeo, los latidos del corazón y la actividad eléctrica de los músculos.

Una vez que la señal se ha digitalizado y está libre de artefactos, se realiza el procesado. A su vez, el procesado se divide en dos etapas: la extracción de características y el algoritmo de traducción. En la primera parte del procesamiento, las señales se someten a varios procedimientos con el objetivo de extraer las características de la señal que codifican las intenciones del usuario. A continuación, el algoritmo de traducción transforma estas características en comandos que llevan a cabo la intención del usuario [5].

Por último, se explican las principales aplicaciones de los sistemas BCI como son la ayuda a la comunicación, mediante la selección de comandos, letras o números; el movimiento de un cursor o el desplazamiento sobre un mapa. En particular, analizaremos el estado del arte de los paradigmas *odd-ball* utilizados en los sistemas BCI.

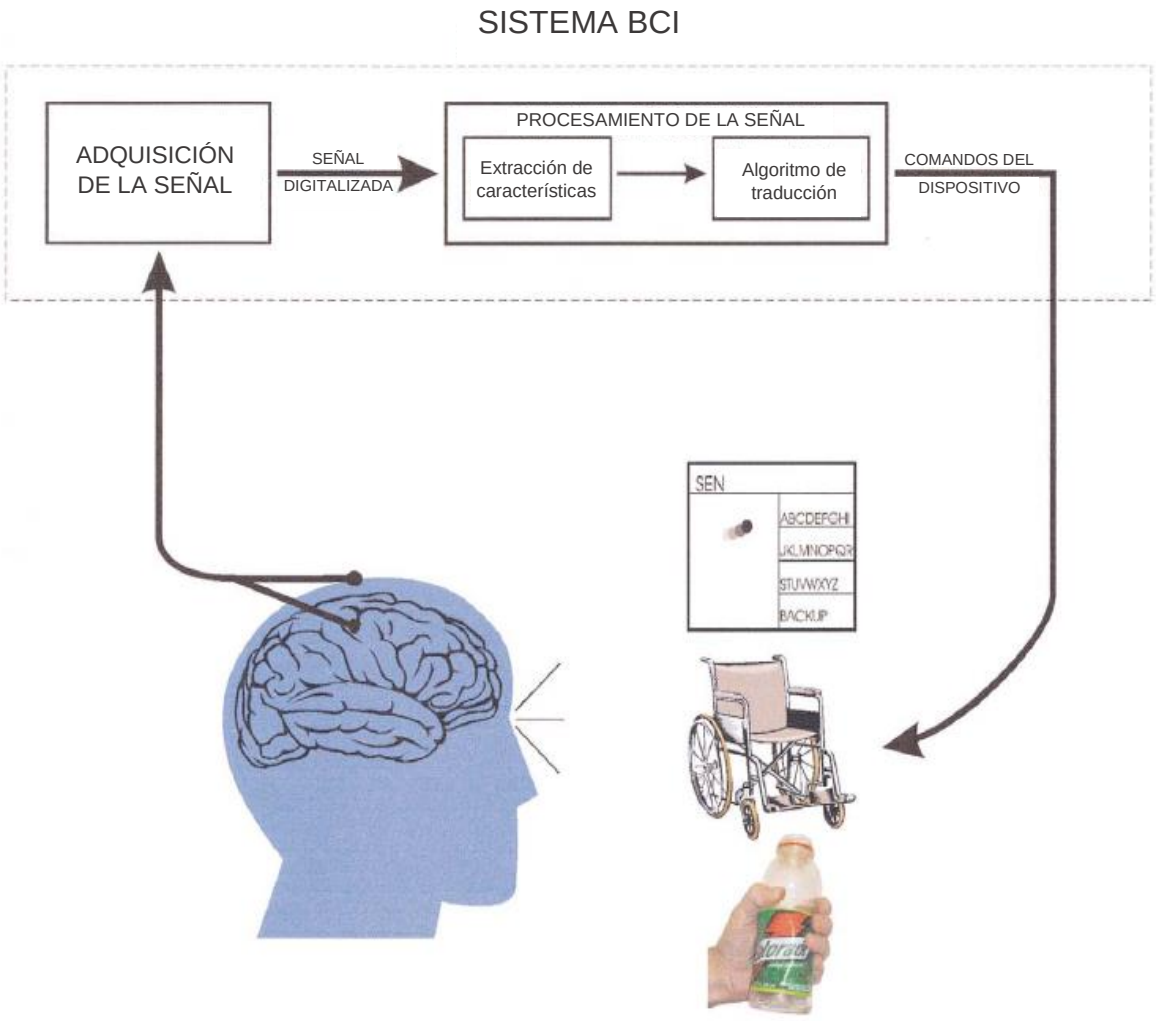


Figura 2-2. Estructura de los sistemas BCI [5].

2. *Métodos para registrar la actividad cerebral*

La actividad cerebral utilizada en los sistemas BCI se puede adquirir a través de varios métodos. Estos métodos se pueden clasificar dependiendo de si se trata de la adquisición de señales eléctricas, más utilizados; o señales no eléctricas, como los que se muestran a continuación.

- **Magnetoencefalografía (*Magnetoencephalography*, MEG).** La magnetoencefalografía es una técnica utilizada para el registro de la actividad magnética del cerebro, caracterizada por ser un método no invasivo. En esta técnica, la actividad cerebral se registra mediante campos magnéticos, lo que permite investigar las relaciones entre las estructuras cerebrales y sus funciones.

- **Espectroscopia de infrarrojo cercano (*Near Infrared Spectroscopy*, NIRS).** La espectroscopia estudia la interacción entre la radiación y la materia utilizando la región del espectro electromagnético llamada región del infrarrojo cercano. Esta técnica consiste en la emisión de fotones de luz con longitudes de onda entre 780 nm y 2500 nm, sobre la frente

del paciente. Los fotones se dispersan y se introducen a través del cráneo y del cerebro y algunos vuelven a la parte superior de la cabeza, debido a la reflectancia espectral. A partir de este proceso, se obtienen medidas sobre el nivel de saturación de oxígeno y el nivel de azúcar en sangre [6].

- **Imagen de resonancia magnética funcional (*functional Magnetic Resonance Imaging*, fMRI).** La fMRI es una técnica utilizada para medir la respuesta hemodinámica del cerebro, que proporciona imágenes de las regiones cerebrales activas, es decir, obtiene información sobre la localización de las funciones cerebrales. Se trata de un procedimiento clínico y de investigación.

- **Tomografía por emisión de positrones (*Positron Emission Tomography*, PET).** La PET es una técnica no invasiva que se basa en la detección y el análisis de la distribución tridimensional que adopta en el interior del cuerpo un radiofármaco, administrado a través de una inyección intravenosa. Dependiendo del estudio, se utilizan diferentes radiofármacos. Con esta técnica, se mide la actividad metabólica de los tejidos que forman el cuerpo humano, entre ellos, el sistema nervioso central.

Técnicas como la fMRI o la NIRS proporcionan una buena resolución espacial, pero poseen muy poca resolución temporal, lo que supone una desventaja para su utilización en tiempo real. Además, la fMRI y la PET requieren de un equipo costoso y de gran volumen.

El MEG, sin embargo, puede utilizarse en tiempo real ya que posee una resolución espacial y temporal excelentes. La principal desventaja de esta técnica, al igual que ocurre con fMRI, es que requiere un superconductor, lo que hace que su uso sea impracticable, debido al elevado coste y su gran volumen. Otra desventaja importante de esta técnica es que necesita utilizar aislamiento magnético para el registro de la actividad cerebral [6].

Las principales características que se buscan a la hora de utilizar un sistema BCI es que pueda utilizarse en tiempo real, que sea portátil, eficiente y cuyo coste no sea muy elevado. Por lo tanto, las técnicas anteriormente estudiadas no cumplen con estos requisitos.

Las señales eléctricas, por tanto, son las más utilizadas ya que cumplen con los requisitos impuestos por los sistemas BCI. A continuación, se describen las señales eléctricas más utilizadas y en la Figura 2-3 se muestra la localización de los electrodos en función de la técnica utilizada.

- **Electroencefalografía (*Electroencephalography*, EEG).** El EEG es una técnica no invasiva en la que se registra la actividad eléctrica del cerebro mediante la utilización de electrodos situados sobre el cuero cabelludo. Habitualmente se utilizan 64 electrodos para cubrir la totalidad del cuero cabelludo, pero en aplicaciones concretas suele emplearse un número de electrodos menor [5]. La desventaja de esta técnica es que suele estar contaminada con artefactos como la actividad electromiografía (EMG) de los músculos del cráneo o la actividad electrooculográfica (EOG).

- **Electrocorticografía (Electrocorticography, ECoG).** La ECoG es una técnica invasiva en la que la actividad eléctrica del cerebro se registra a través de electrodos situados sobre la superficie del córtex. Debido a esto, el ECoG tiene mayor resolución topográfica y no

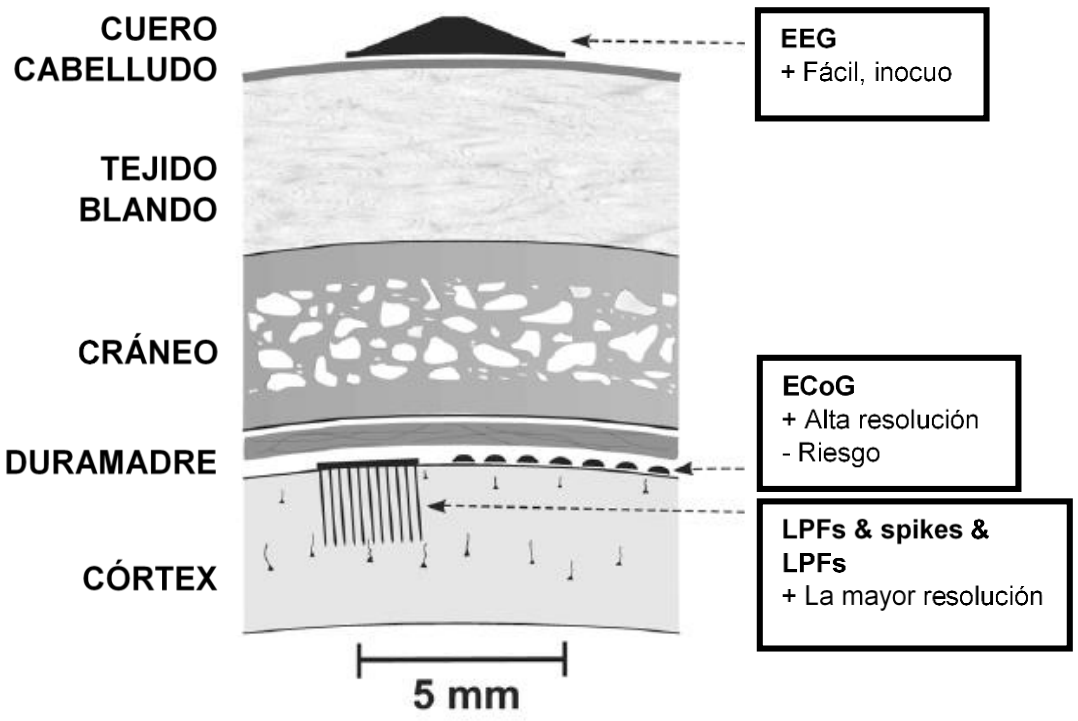


Figura 2-3. Localización de los electrodos según las diferentes técnicas estudiadas para registrar la actividad eléctrica del cerebro [6].

está libre de artefactos como el EMG o el EOG, pero supone un riesgo para el paciente [6].

- **Electrodos epidurales o intracorticales.** Los electrodos intracorticales son electrodos de cristal con forma de cono hueco que sirven para registrar la actividad de neuronas aisladas. Estos electrodos se implantan dentro del córtex, por lo que ofrecen mayor resolución espacial y temporal que las técnicas anteriores. La principal desventaja es que los electrodos pueden permanecer implantados durante un tiempo limitado. Además, la neurona en la que se ha implantado el electrodo puede morir o, en otro caso, puede que otra neurona asuma sus funciones haciendo que la obtención de la señal a través de esta técnica sea inservible.

Como se ha mostrado, las diferentes técnicas estudiadas poseen tanto ventajas como inconvenientes. El EEG es la técnica más utilizada ya que se trata de un método no invasivo, portátil y cuyo coste no es elevado. Sin embargo, la resolución espacial que ofrece es limitada y está contaminado por artefactos como el movimiento de músculos cercanos o el movimiento de los ojos. Además, los electrodos del EEG deben tener un mantenimiento continuo para poder asegurar que su impedancia sea baja y poder obtener una señal con calidad.

El ECoG ofrece mayor resolución espacial y está libre de artefactos, pero se trata de un método invasivo que requiere de una intervención quirúrgica para poder implantar los electrodos en el córtex. En la práctica, esta técnica solo se utiliza en personas que sufren alguna enfermedad, como la epilepsia.

Por último, los electrodos intracorticales ofrecen una resolución excelente, la mejor de entre todas las técnicas analizadas. Por otro lado, debido a las desventajas mencionadas anteriormente, solo se utilizan en experimentos [6].

Por lo tanto, se va a priorizar el uso de técnicas no invasivas, como es el EEG, ya que aunque tienen peores resultados en cuanto a la resolución, suponen un menor riesgo para el paciente. Tras obtener la señal a través de los electrodos situados sobre el cuero cabelludo, se va a digitalizar y se van a eliminan los artefactos presentes.

3. Tipos de señales de control en BCI

Una vez se ha seleccionado el método para el registro de la actividad cerebral que se va a utilizar en el desarrollo de este trabajo, es necesario elegir una señal de control adecuada. Estas señales se pueden generar de manera natural, como respuesta a un estímulo externo; o de manera voluntaria, con entrenamiento. A las primeras se las denomina señales de control exógenas, mientras que las segundas son señales de control endógenas [5].

Para registrar la actividad cerebral mediante un EEG, habitualmente se utilizan 64 electrodos. Sin embargo, en aplicaciones que utilizan señales de control concretas, se puede reducir el número de electrodos si se conoce la localización específica donde se generan estas señales. En la Figura 2-4, se muestran las distintas regiones en las que se divide la corteza cerebral.

Las señales más utilizadas en los sistemas BCI son los potenciales evocados visuales (VEP), los potenciales corticales lentos (SCP), los potenciales evocados P300, los ritmos sensoriomotores y los potenciales de neuronas corticales [5].

3.1. Potenciales evocados visuales

Los potenciales evocados visuales (*Visual Evoked Potentials*, VEP) son cambios de potencial que se producen en la corteza visual tras una estimulación luminosa [5]. Se producen cuando una persona enfoca la atención visual en una fuente de luz que parpadea con una frecuencia superior a 4 Hz y se utilizan para determinar la dirección de la mirada y, así, conocer cuál es la intención del usuario [7]. Los VEP surgen en la parte del córtex visual, localizada en la parte posterior del cerebro, en torno al lóbulo occipital y parietal [5].

Para determinar la dirección de la mirada se utiliza una matriz de fuentes de luz que parpadean a diferentes frecuencias. Cuando el usuario fija su mirada en una fuente concreta, el sistema mide la frecuencia con la que aparecen los VEP y se genera un fenómeno de resonancia.

Este fenómeno provoca que se produzca un pico en la frecuencia correspondiente a la fuente de luz en la que el usuario ha fijado su mirada y en sus armónicos. Como se puede observar en la Figura 2-5, la respuesta de mayor amplitud se obtiene para frecuencias entre 5 y 20 Hz [7]. Sabiendo cuál es la frecuencia correspondiente al pico, se puede estimar la dirección de la mirada del usuario y, por tanto, sus intenciones.

Los VEP provocan una respuesta natural del cerebro y no necesitan que el usuario realice un entrenamiento, por lo que se clasifican como señales exógenas [8]. Debido a esto, los sistemas BCI

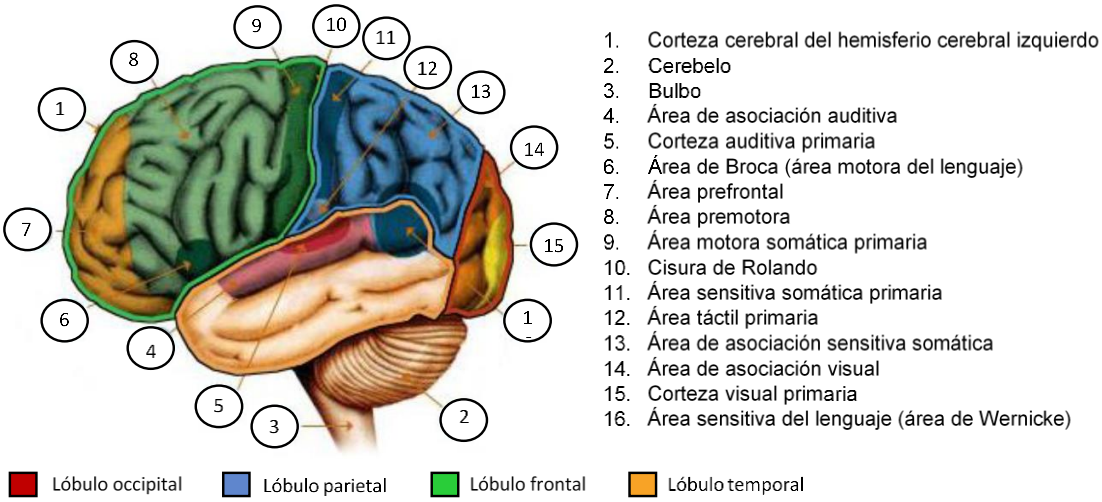


Figura 2-4. Regiones en las que se divide la corteza cerebral.

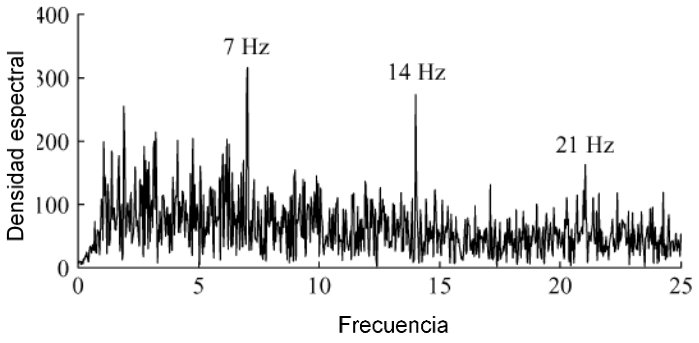


Figura 2-5. Espectro frecuencial de una señal EEG registrada durante una estimulación visual a una frecuencia de parpadeo de 7 Hz. El fenómeno de resonancia de los VEP tiene picos tanto en 7 Hz como en sus armónicos en 14 y 21 Hz [7].

que utilizan estas señales son sistemas dependientes, dependen de la capacidad del usuario de poder mantener la mirada fija sobre la fuente luminosa. Los VEP se pueden utilizar en diversas aplicaciones, desde seleccionar comandos a controlar un cursor.

3.2. Potenciales corticales lentos

Los potenciales corticales lentos (*Slow Cortical Potentials*, SCP) son cambios de voltaje lentos, con una duración entre 0,5 y 10 segundos, generados en el córtex a bajas frecuencias. Los SCP se generan en la parte superior de la cabeza, en torno a la unión del lóbulo parietal y el lóbulo frontal, también llamado vértex; y son referenciados a los mastoides.

Los valores negativos en los SCP se asocian con el movimiento y otras funciones que involucran un aumento en la activación cortical. Por otra parte, los valores positivos en los SCP se asocian con una disminución en la activación cortical. En la Figura 2-6, se muestra una comparativa entre el aumento y la disminución en la activación cortical.

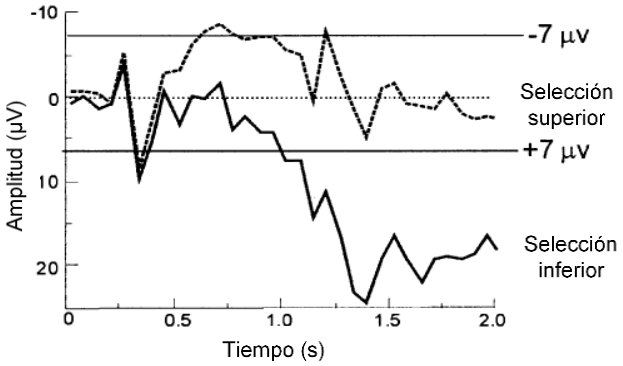


Figura 2-6. SCP registrados durante una prueba en la que los usuarios tenían que aprender a controlar los SCP para desplazar un cursor hacia un objetivo en parte superior de una pantalla (SCP más negativo) o en la parte inferior (SCP más positivo) [5].

La selección del objetivo se realiza mediante la medición de voltaje. Durante los 2 primeros segundos, el sistema mide el nivel de voltaje inicial del usuario y los 2 segundos siguientes se determina si se ha producido o no activación cortical. En anteriores estudios, se ha demostrado que las personas pueden aprender a controlar los SCP y, con esto, aprender a controlar el movimiento de un objeto en una pantalla [5].

Como se trata de una señal obtenida a partir del EEG va a estar contaminada con artefactos. Los artefactos que más afectan a los SCP son la respiración y el parpadeo. A pesar de esto, las tasas binarias obtenidas con los SCP son de aproximadamente 15 bits por minuto [9].

Los SCP requieren de entrenamiento para poder controlar la señal, por lo que se clasifican como señales endógenas. Además, dado que solo permiten discernir entre dos opciones concretas, las aplicaciones que los utilizan se basan en decisiones binarias.

3.3. Potenciales evocados P300

Los potenciales evocados P300 son picos de voltaje positivos que se producen, como respuesta a un estímulo, alrededor de 300 ms después de que éste se produzca. Estos potenciales se generan sobre la zona parietal del córtex. En la Figura 2-7 se muestra la forma de un potencial evocado P300 [5]. Para registrar la señal se utilizan electrodos situados sobre la región central del lóbulo parietal. Estos electrodos reciben el nombre de Cz, Fz y Pz.

Para provocar la aparición de los potenciales evocados P300 se utiliza el paradigma *odd-ball*. Este paradigma consiste en presentar el estímulo deseado de forma infrecuente entre estímulos más frecuentes. El potencial evocado P300 se producirá cuando la opción deseada se encuentre entre otras opciones que no se desean. Cuanto más improbable sea el estímulo deseado, más probable será que aparezca el potencial evocado P300 y que su amplitud sea mayor. De esta forma, se puede determinar la intención del usuario [5].

Los potenciales evocados P300 provocan una respuesta natural del cerebro ante un estímulo y no requieren entrenamiento, por lo que se pueden clasificar como señales exógenas. Debido a esto, se pueden alcanzar precisiones altas en un periodo de tiempo corto.

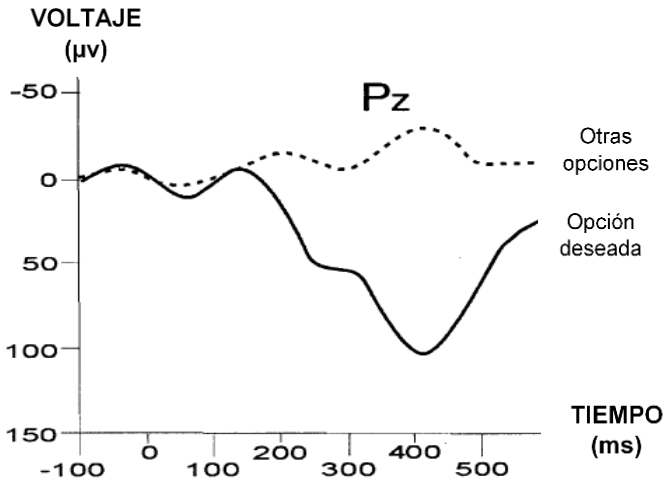


Figura 2-7. Potencial evocado P300. Solo la opción deseada muestra el pico aproximadamente 300 ms después de que se produzca el estímulo [5].

Este tipo de señal se utiliza en aplicaciones de selección, en las que se presenta una matriz con celdas formadas por letras, números o comandos que el usuario debe seleccionar y que se iluminan de forma aleatoria. En un intento se iluminan aleatoriamente todas las filas y columnas de la matriz, de forma que cada celda se ilumina 2 veces. El usuario debe fijarse en una de las celdas, por lo que al iluminarse 2 veces se producirán 2 potenciales evocados P300. Para determinar la celda que el usuario desea seleccionar se realiza un promedio de las respuestas de cada celda de la matriz y se elige la celda cuyo potencial sea el de mayor amplitud.

La desventaja de esta señal es que, a largo plazo, los usuarios pueden acostumbrarse a las iluminaciones haciendo que los potenciales evocados P300 tengan menor amplitud y, por tanto, el rendimiento del sistema BCI se deteriore o no se obtengan los resultados deseados [5].

3.4. Ritmos sensoriomotores (ritmos μ y β)

En personas despiertas, las áreas sensoriales o corticales motoras primarias muestran actividad en el EEG en la banda de 8 a 12 Hz cuando no se está realizando ninguna acción motora o procesando estímulos externos. En la Figura 2-8, se muestra la actividad producida en la banda de 8 a 12 Hz.

Esta actividad se denomina *ritmo μ* cuando se enfoca sobre la corteza somatosensorial o motora del córtex; o *ritmo α* cuando la actividad se enfoca sobre la corteza visual. Para registrar esta actividad, se utilizan los electrodos C3, C4 y Cz y, en otras ocasiones, se añaden F3, F4 O1 y O2.

Los *ritmos μ* están comprendidos en frecuencias entre 8 y 12 Hz y, además, están asociados con los *ritmos β,* que son ritmos de mayor frecuencia, situados en la banda comprendida entre 18 y 26 Hz. Algunos *ritmos β* son armónicos de los *ritmos μ*, pero otros son independientes, de manera que se pueden utilizar como dos características diferentes [5].

El movimiento o la preparación del movimiento conlleva una disminución en los *ritmos μ* y *β*, particularmente contralateral al movimiento. A esta disminución se le conoce como desincronización relacionada con eventos (*Event-Related Desynchronization,* ERD) y ocurre pocos segundos antes de realizar el movimiento. Se conoce como desincronización ya que la señal pierde su carácter oscilatorio, haciendo que se produzca una disminución en sus picos frecuenciales.

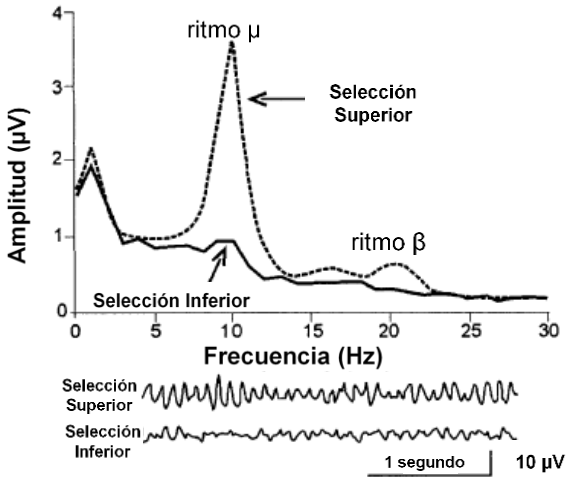


Figura 2-8. Ritmos μ y β [5].

Por otro lado, después del movimiento y con la relajación, se produce un aumento en los *ritmos μ* y *β*. A este aumento se le denomina sincronización relacionada con eventos (*Event-Related Synchronization*, ERS) y determina el punto en el cuál la señal recupera su carácter oscilatorio. Una característica a destacar es que, tanto los ERD como los ERS, se localizan en el lado opuesto del cerebro al movimiento que se desea realizar [5].

El panel superior de la Figura 2-9 muestra tres tipos de señales diferentes que ocurren sobre la zona sensoriomotora del electrodo C3 cuando el usuario levanta un dedo de forma voluntaria. En el *ritmo μ*, la ERD comienza aproximadamente 2,5 segundos antes del inicio del movimiento, alcanza la desincronización máxima poco después del inicio del movimiento y recupera el nivel inicial en pocos segundos. La actividad del *ritmo β* muestra una ERD de corta duración durante el inicio del movimiento, seguido de una ERS con un máximo después de la ejecución del movimiento. En cuanto a las oscilaciones de la *banda γ* (36 – 40 Hz), poseen un aumento de potencia antes del inicio del movimiento, lo que se corresponde con una ERS. Sin embargo, estas oscilaciones no suelen encontrarse en el EEG humano [10].

En el panel inferior, se muestra el EEG en los diferentes electrodos cuando el usuario está realizando el mismo movimiento. Se pueden observar las ERD en los electrodos que se situan en el centro de la cabeza (C3, C4 y Cz), antes de realizar el movimiento; y las ERS en los electrodos situados sobre la zona visual del córtex (P3, P4 y Oz), después del movimiento [10].

Se ha comprobado que, además de la realización del movimiento, la observación o la imaginación de éste, produce cambios en los *ritmos μ* y *β*. Por lo tanto, a partir de los ritmos sensoriomotores, pueden conseguirse sistemas BCI independientes, ya que no es necesario que el usuario realice le movimiento [11].

Los ritmos sensoriomotores también requieren de entrenamiento para poder controlar la señal, por lo que se clasifican como señales endógenas. Las tasas binarias alcanzadas con este tipo de señales son de entre 20 y 25 bits por minuto y la precisión en selecciones binarias es del 95% [5].

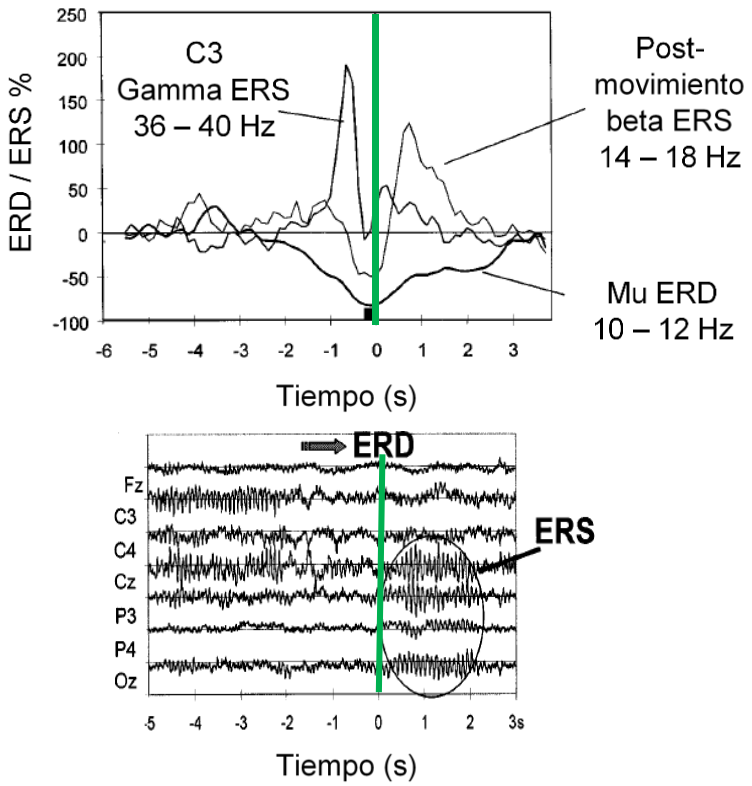


Figura 2-9. Señal EEG antes y después de realizar un movimiento. El gráfico superior se corresponde con una ampliación de la actividad del electrodo C3 [10].

3.5. Potenciales de neuronas corticales

Los potenciales de neuronas corticales se obtienen a partir de la actividad eléctrica de neuronas asiladas situadas sobre el córtex motor. Estos potenciales se recogen utilizando electrodos epidurales o intracorticales implantados en las neuronas que se van a monitorizar. En la Figura 2-10 se muestra un esquema de la implantación de un electrodo epidural o intracortical.

Estos electrodos implantados en cortezas motoras de monos se han utilizado para aprender a controlar las tasas de activación neuronal. Más tarde se intentó llevar a cabo este trabajo en humanos, pero la evaluación se retrasó por la falta de electrodos intracorticales adecuados para el uso en humanos y por la imposibilidad de realizar un registro estable de la actividad neuronal a largo plazo. El problema que surge es que la neurona que se está monitorizando puede morir o puede que otra neurona asuma sus funciones, haciendo que el electrodo implantado ya no sea útil [5].

Utilizando este tipo de señales, el usuario puede controlar un cursor en una dimensión para seleccionar los comandos que aparezcan en una pantalla. Además, se podría controlar un cursor en dos dimensiones si el usuario consigue realizar un control sobre la actividad EMG residual. En este caso, se han logrado velocidades de comunicación de hasta 15 bits por minuto [5].

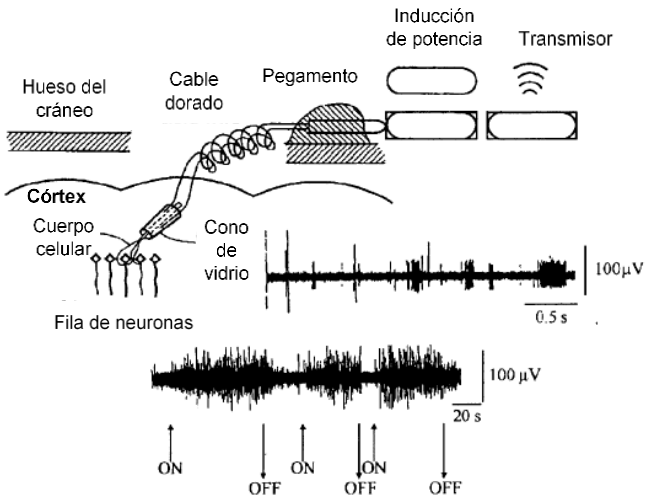


Figura 2-10. Esquema de la implantación de un electrodo epidural o intracortical [5].

Para llevar a cabo esta tarea, se utilizaron microelectrodos implantados en la corteza cerebral de animales despiertos para registrar los potenciales de las neuronas individuales durante el movimiento. Debido a esto, no está claro si los mismos patrones de actividad neuronal estarán presentes cuando los movimientos no se realicen o cuando el animal ya no sea capaz de realizar movimientos.

En cuanto a la utilización de estas señales en humanos, no se puede asegurar un control de la actividad neuronal ya que no se han realizado experimentos [5].

4. Tratamiento de la señal EEG

El tratamiento de la señal EEG se realiza en 3 etapas. En la primera etapa se realiza la adquisición de la señal, después se realiza el procesado y, por último, la aplicación. El procesado de la señal se puede dividir en 2 etapas, la extracción y traducción de características. En la Figura 2-11, se muestran las etapas del procesado de forma esquemática. A continuación, se detalla cada una de las etapas.

4.1. Adquisición de la señal

La primera etapa del tratamiento de la señal es la adquisición. En esta etapa se obtiene la señal, se amplifica, se digitaliza, se eliminan los artefactos que puedan contaminarla y se almacena para realizar un análisis *offline*.

Como ya se explicó en el capítulo de introducción, para obtener la señal EEG se utilizan electrodos situados sobre el cuero cabelludo atendiendo al sistema internacional 10/20. Es necesario reducir la impedancia que posee el cráneo, por lo que se utilizará un gel conductor. Una vez que se ha obtenido la señal, ésta se amplifica y se digitaliza utilizando una frecuencia de muestreo adecuada. Según el teorema de Nyquist la frecuencia de muestreo debe ser, al menos, 2 veces la frecuencia máxima de la banda que se desea analizar [12].

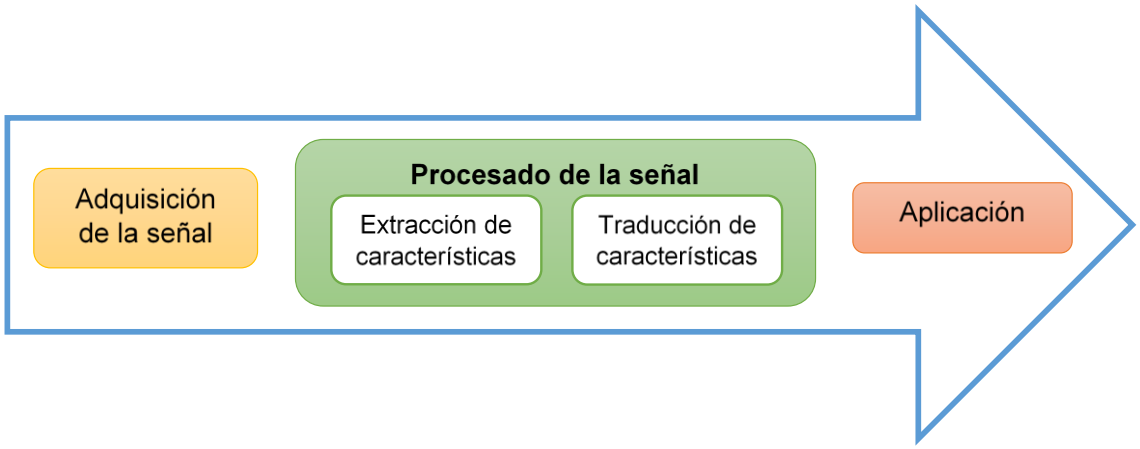


Figura 2-11. Etapas en las que se divide el tratamiento de la señal.

Posteriormente, la señal es filtrada para eliminar los posibles artefactos que puedan estar contaminándola. Los artefactos más comunes presentes en la señal EEG son el movimiento de los ojos o electrooculografía (*Electrooculography*, EOG) y los pestañeos, la actividad de los músculos (EMG) y la actividad del corazón (ECG).

- **Movimiento ocular y pestañeos.** La actividad eléctrica producida por el movimiento de los ojos es visible en la señal EEG, debido a su intensidad y cercanía al cerebro. La EOG mide la diferencia de potencial existente entre la córnea y la retina, cuya amplitud será proporcional al ángulo de visión. Al igual que el movimiento de los ojos, el pestañeo también contamina la señal de interés. Este tipo de artefactos se pueden eliminar obteniendo una señal de referencia a partir de electrodos situados cerca de los ojos. Para ello, se podría utilizar un filtrado adaptativo.

- **Actividad de los músculos.** La actividad eléctrica de los músculos también puede afectar cuando se realiza un registro de la señal EEG. Este tipo de señal suele contaminar a la señal de interés cuando se trata de músculos cercanos, como la mandíbula o los músculos faciales. El efecto producido en el EEG depende de la fuerza con la que el sujeto esté contrayendo los músculos. Este artefacto es especialmente molesto debido a que no es posible adquirir una señal de referencia que solamente contenga la actividad muscular interferente.

- **Actividad del corazón.** La actividad eléctrica producida por el corazón puede interferir en la señal EEG. A pesar de que la amplitud del ECG habitualmente es pequeña en el cráneo en comparación con la amplitud del EEG, cuando se mide la actividad del cerebro, en determinados electrodos y dependiendo del cuerpo del sujeto, dicha actividad puede contaminar la señal EEG de interés. Debido a que el ECG es una señal repetitiva y regular, la eliminación de este artefacto suele ser sencilla.

Por último, a pesar de que este trabajo se basa en el desarrollo de paradigmas en tiempo real, en esta etapa se realiza un registro para poder hacer un análisis *offline.*

4.2. Procesado de la señal

La segunda etapa del tratamiento de la señal es el procesado de ésta. Esta etapa, a su vez, está formada por dos etapas: extracción y traducción de características.

a) Extracción de características

Una vez se ha obtenido la señal digitalizada, ésta se somete a procedimientos de extracción de características. La etapa de extracción de características se basa en obtener información capaz de identificar la señal EEG. Esta información se denomina característica y tiene como objetivo diferenciar las intenciones del usuario.

Como ya hemos visto, existen numerosos artefactos que contaminan la señal de interés, pero también existen factores que pueden afectar a las características de la señal proveniente del usuario. Algunos de estos factores son la fatiga, la motivación, la frustración, etcétera. Para que un sistema BCI funcione de forma correcta, será necesario que se adapte al usuario y que utilice técnicas de procesado de la señal adecuadas [5].

En la Tabla 2-1 se muestra una clasificación de los principales métodos de extracción de características empleados en los sistemas BCI [13]. De entre estos métodos de extracción de características, los más utilizados son los que trabajan en el dominio tiempo-frecuencia y aquellos que trabajan en el dominio espacial. Habitualmente, para realizar la extracción de características se utiliza un método de cada tipo, independientemente de cuál sea el orden de aplicación, tratando cada dimensión por separado; o un método que trabaje en el dominio espacio-tiempo. La elección del método de extracción de características depende principalmente del tipo de señal de control que utilice el sistema y de los artefactos que estén contaminando la señal de interés.

En cuanto a los métodos que trabajan en el dominio tiempo-frecuencia, destacan los métodos temporales por su reducido coste computacional y por su rapidez a la hora de proporcionar retroalimentación al usuario. Debido a esto, estos métodos son los más utilizados en aplicaciones de BCI en tiempo real. Por lo general, los métodos en el dominio frecuencial requieren una transformación de la señal, haciendo que aumente el coste computacional. Por lo tanto, para aplicaciones en tiempo real se van a utilizar métodos en el dominio frecuencial que operen con pequeños segmentos de la señal [5].

Por otro lado, los métodos que trabajan en el dominio espacial también dependen de la señal de control que vaya a utilizar el sistema ya que depende de la ubicación en la que se encuentre la fuente en la corteza cerebral. Debido a esto, el método espacial más adecuado para detectar los ritmos sensoriomotores (ritmos μ y β), relativamente localizados, no será la mejor opción para medir los SCP o los P300, distribuidos a lo largo del cuero cabelludo.

Cuando las señales están contaminadas y no reflejan un evento cerebral específico, se realiza un análisis offline utilizando r^2. El coeficiente de determinación o r^2 es una medida útil que calcula la correlación de los datos elevada al cuadrado. Se define como la varianza de la característica de la señal que es contabilizada por la intención del usuario [5].

TABLA 2-1. Clasificación de los principales métodos de extracción de características.

Tiempo – Frecuencia	Espacio	Tiempo – Espacio	Modelos inversos
▪ Transformada de Fourier (TF) ▪ Transformada Wavelet (WT) ▪ Modelos Autorregresivos (AR) ▪ Filtrado Paso-Banda ▪ Filtrado Adaptado ▪ Filtro de Kalman ▪ Detección de Picos	▪ Filtro Laplaciano ▪ Método de Referencia de Media Común (CAR) ▪ Análisis de Componentes Independientes (ICA) ▪ Patrones Espaciales Comunes (CSP) ▪ Amplitudes ▪ Proporciones y Diferencias	▪ Análisis de Componentes en Tiempo y Espacio ▪ Modelos Autorregresivos Multivariables ▪ Coherencia	▪ EEG a ECoG ▪ EEG a Dipolos Fuente

En la Figura 2-12, se muestran los resultados del coeficiente r^2 en la detección de los ritmos sensoriomotores (ritmos μ y β) sobre el electrodo C3, utilizando 4 métodos espaciales diferentes aplicados a los mismos datos. Los métodos utilizados son el método de referencia de media común (CAR), el Laplaciano largo, con separación entre electrodos de 6 cm; el Laplaciano corto, con separación entre electrodos de 3 cm; y la referencia en la oreja.

El valor de r^2 más alto se obtiene para CAR, seguido del Laplaciano largo. Los resultados más bajos se obtienen para la referencia en la oreja. El Laplaciano largo obtiene mejores resultados que el Laplaciano corto, ya que cuanto mayor es la distancia entre electrodos circundantes, más sensible es el Laplaciano a las fuentes de voltaje con frecuencias espaciales más altas, es decir, a las fuentes más localizadas; y menos lo es a las frecuencias espaciales más bajas, es decir, a las fuentes más distribuidas [5].

Otras alternativas a estos métodos espaciales comúnmente utilizadas en los sistemas BCI son el análisis de componentes principales (*Principal Component Analysis*, PCA), el análisis de componentes independientes (*Independent Component Analysis*, ICA) o los patrones espaciales comunes (*Common Spatial Pattern*, CSP).

b) Traducción de características

La etapa de traducción de características, también llamada algoritmo de clasificación, se encarga de traducir las características obtenidas en la etapa de extracción y convertirlas en comandos del dispositivo, que lleva a cabo la intención del usuario [5]. Para ello, se realiza una clasificación de las características obtenidas, en función de diferentes parámetros, y se asocian con un determinado comando. El objetivo es que exista una correspondencia entre los comandos del dispositivo y la selección que el usuario ha realizado.

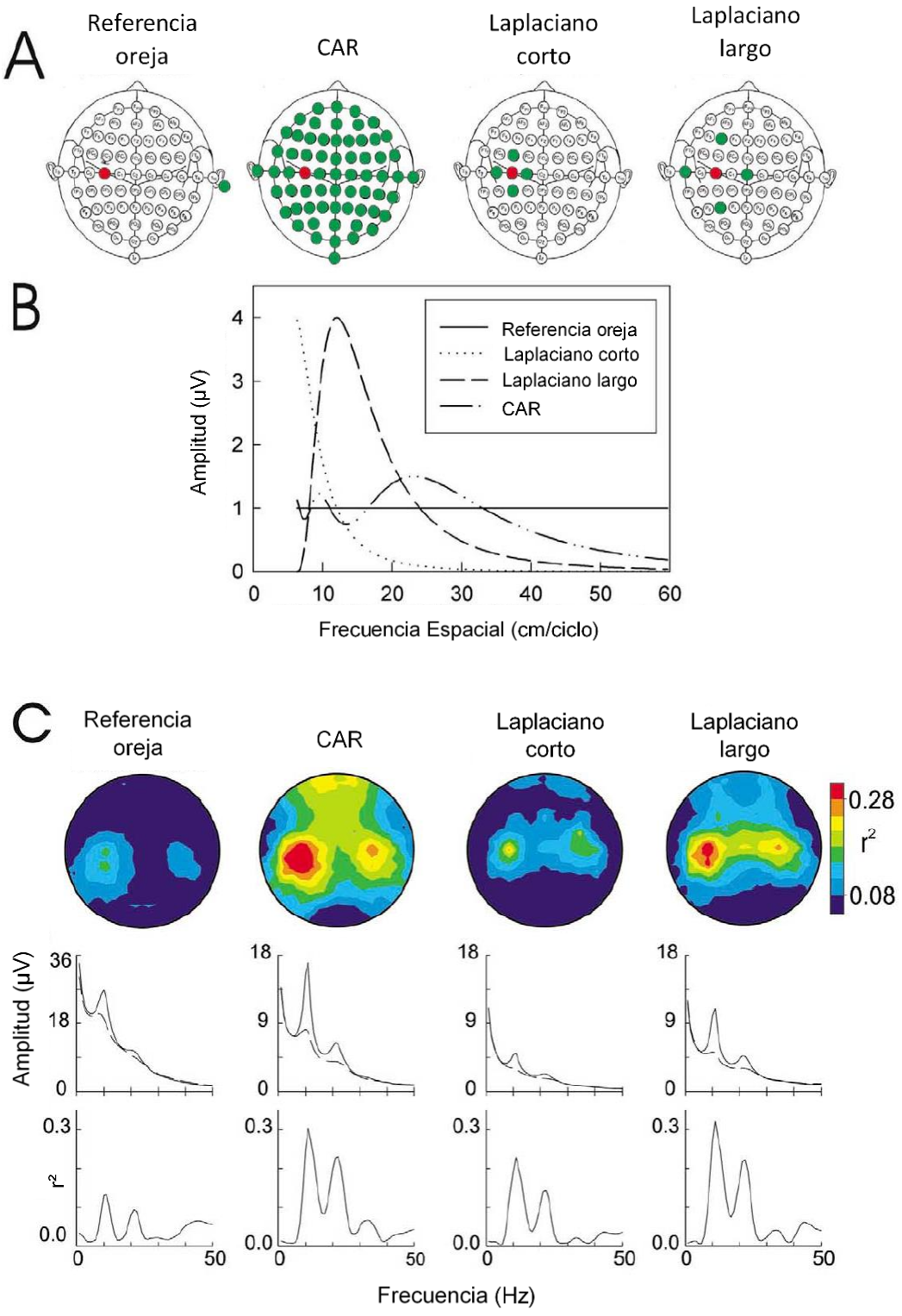


Figura 2-12. Comparación de 4 métodos espaciales de extracción de características. A) Localización de los electrodos utilizados para registrar la señal EEG objetivo C3 (rojo). B) Banda de paso utilizada para cada método. Muestra la raíz cuadrada de los valores de la raíz cuadrática media de la señal objetivo C3. C) Topografía de r^2 medio y amplitud espectral para cada método espacial [5].

Los algoritmos de clasificación se pueden dividir en dos grandes grupos, clasificadores lineales y clasificadores no lineales. En la Tabla 2-2 se muestran los clasificadores más empleados en los sistemas BCI.

Los métodos lineales utilizan un hiperplano de separación fruto de la combinación lineal de las características que divide el espacio de datos en regiones, cada una de las cuales se corresponde con una selección diferente. El hiperplano de separación óptimo es aquel que maximiza el margen mínimo. El margen de un clasificador lineal se define como la distancia mínima de cualquier punto al hiperplano [14]. En la Figura 2-13 (A) se muestra un clasificador lineal que maximiza el margen mínimo.

Los clasificadores lineales son, generalmente, más robustos y requieren menor carga computacional. A pesar de esto, estos clasificadores pueden fallar en presencia de ruido u *outliers*. Los *outliers* son valores atípicos que distan del resto y que suelen ser valores incorrectos. En la Figura 2-13 (B) se puede observar que los *outliers* pueden cambiar de forma drástica el hiperplano de decisión si la influencia de los valores atípicos no está limitada. Para controlar el nivel de desconfianza de los datos se utiliza la regularización, que ayuda a limitar la influencia de los datos atípicos y permite obtener un hiperplano más fiable [14].

Los métodos no lineales también utilizan un hiperplano de separación, pero en este caso, para obtener el hiperplano se emplea una función cuadrática que proyecta los datos a una dimensión mayor y calcula un hiperplano lineal en esa dimensión. Estos métodos suelen obtener mayor precisión, ya que permiten configurar un mayor número de parámetros, siendo más flexibles. A cambio, estos métodos requieren mayor coste computacional.

En general, los métodos lineales suelen ser más utilizados debido a que conllevan menor carga computacional y son métodos más simples. Suelen utilizarse cuando el número de muestras es pequeño y la información sobre los datos es limitada. Los métodos no lineales son más adecuados si el número de datos es elevado. Estos métodos suelen utilizarse para encontrar una estructura más compleja en los datos [14].

Además de traducir las características, en esta etapa también se debe considerar la adaptabilidad al usuario. El algoritmo de traducción debe ajustarse a las variaciones producidas por diversos factores como son el entorno, la hora del día, la fatiga, etcétera. Estas variaciones deben tenerse en cuenta para que el rango de valores de las características obtenidas de la señal del usuario coincida con el rango de valores disponible de los comandos del dispositivo. Los requisitos de adaptabilidad se clasifican en 3 niveles [5].

El primer nivel requiere adaptar las características de las señales a cada usuario, ya que existen diferencias significativas entre usuarios. Por ejemplo, la amplitud de los potenciales SCP o P300 es diferente en cada usuario.

El segundo nivel requiere adaptar el clasificador a las variaciones producidas en las características obtenidas tanto a largo como a corto plazo. Estas variaciones están relacionadas con la hora del día, los niveles hormonales, el entorno, la fatiga o las enfermedades. Para corregir el impacto de estas variaciones, se realizan ajustes periódicos en tiempo real [5].

TABLA 2-2. Clasificación de los principales métodos de traducción de características.

Lineal	▪ Análisis Discriminante Lineal (LDA) ▪ Perceptrón Multicapa (MLP) ▪ Regresión, regularización y adaptación		
No Lineal	***Estructura Fija***	▪ Análisis Cuadrático Discriminante (QDA)	
	Estructura Modificable	***Basados en Memoria***	▪ Algoritmo de las k vecinas más cercanas (KNN)
			▪ Máquinas de Soporte Vectorial (SVM)
			▪ Mínimos Cuadrados Parciales (PLS)
		Combinaciones de No Linealidades Simples	▪ Redes Neuronales Artificiales (ANN)
			▪ Árbol de decisión
			▪ Cuantificación del Vector de Aprendizaje
		Modelos Generativos	▪ Modelo de mezclas gaussianas
			▪ Modelos ocultos de Markov (HMM)

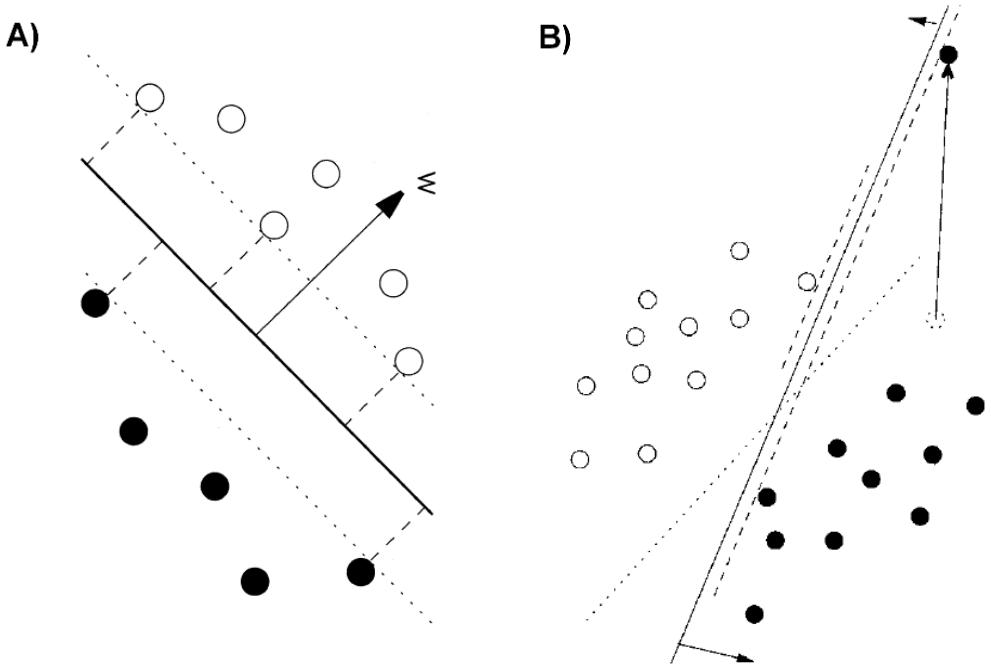


Figura 2-13. Clasificador lineal que maximiza el margen mínimo. A) Comportamiento óptimo del clasificador. B) Comportamiento en presencia de un *outlier* [14].

El tercer nivel requiere mantener la precisión del clasificador a largo plazo, siendo capaz de detectar una disminución en el nivel de atención, responder ante esta disminución y ofrecer realimentación al usuario.

Este tercer nivel de adaptación está relacionado con el aprendizaje de los clasificadores. Existen clasificadores no lineales capaces de "adaptarse" a las características de la señal de cada usuario y adaptarse a ellas. Para ello, es necesario realizar un entrenamiento que permita obtener los datos que vaya a utilizar el clasificador. Algunos de los métodos lineales que permiten realizar este aprendizaje son las máquinas de soporte vectorial (*Support Vector Machines*, SVM) o las redes neuronales artificiales (*Artificial Neural Networks*, ANN). Ambos métodos se caracterizan por ser algoritmos rápidos y, por lo tanto, adecuados para aplicaciones en tiempo real [5].

4.3. Aplicación

La tercera etapa del tratamiento de la señal es la etapa de aplicación. Una vez se han traducido las características de la señal, se obtiene la señal de control. En esta última etapa, será necesario implementar una aplicación que reciba la señal de control y la traduzca a comandos que realicen acciones. En la siguiente sección, se detallan las aplicaciones más utilizadas de los sistemas BCI.

5. Aplicaciones de los sistemas BCI

Los sistemas BCI destacan por las numerosas aplicaciones en las que se han utilizado. A continuación, se exponen algunas de las aplicaciones más básicas que utilizan las señales de control que se han mencionado en apartados anteriores. Las principales aplicaciones de los sistemas BCI son la selección de caracteres, el movimiento de un cursor o el desplazamiento sobre un mapa.

5.1. Selección de caracteres

La aplicación de selección de caracteres es la más utilizada en el ámbito de los sistemas BCI. A partir de la aplicación más básica, se han desarrollado numerosas aplicaciones más complejas que utilizan la selección de caracteres, número o comandos para implementar otras funcionalidades. En este tipo de aplicaciones se utilizan como señales de control los potenciales VEP o los potenciales SCP.

a) Potenciales Evocados Visuales

Otra señal de control utilizada en la selección de caracteres son los potenciales VEP. Esta señal suele utilizarse cuando los sujetos tienes alguna discapacidad o enfermedades neurodegenerativas.

Las aplicaciones que utilizan los potenciales VEP como señal de control muestran una matriz de caracteres, cuyas celdas se iluminan a diferencias frecuencias. Cuando el usuario se fija en el carácter que desea seleccionar y éste se ilumina, se produce un fenómeno de resonancia a la misma frecuencia a la que se ha iluminado el carácter. El sistema identifica los picos producidos en la señal en dicha frecuencia y sus armónicos, y determina la localización en la que el usuario se está fijando.

En la Figura 2-14, se muestra un ejemplo de una aplicación que utiliza los potenciales VEP. En el tablero de ajedrez, cada celda se ilumina a una frecuencia diferente. Para mover la figura a través del tablero, el usuario debe fijarse en la celda a la que quiere mover dicha figura [15].

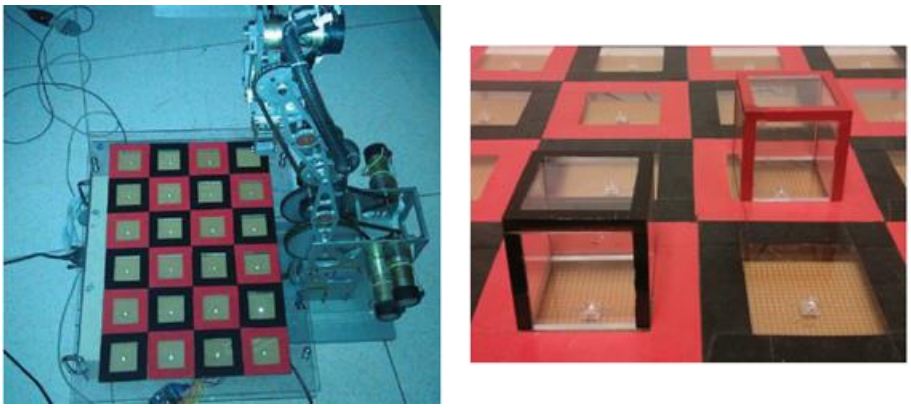


Figura 2-14. Aplicación que utiliza los potenciales evocados visuales para jugar al ajedrez [15].

Los potenciales VEP tienen la ventaja de ser una señal de control exógena, por lo que no necesitan realizar un entrenamiento previo a la utilización de la aplicación.

b) Potenciales Corticales Lentos

Los potenciales SCP solamente permiten realizar selecciones binarias. Debido a esto, la selección de caracteres se realiza dividiendo sucesivamente el alfabeto en dos partes: primero se divide en dos, después en cuatro, etcétera. El procedimiento llevado a cabo en aplicaciones que utilizan esta señal de control se muestra en la Figura 2-15 [16].

Al contrario que los potenciales VEP, los potenciales SCP son una señal de control endógena, por lo que necesitan realizar un entrenamiento previo para poder controlar la aplicación.

5.2. Movimiento de un cursor

El objetivo de las aplicaciones en las que hay que desplazar un cursor es aprovechar el control de movimiento del cursor para seleccionar botones, navegar por menús o moverse en un videojuego.

a) Potenciales Corticales Lentos

Dado que los potenciales SCP solo permiten realizar selecciones binarias, las aplicaciones que utilizan esta señal de control para mover un cursor suelen ser aplicaciones en las que el cursor se desplaza horizontalmente a una velocidad constante y el usuario debe controlar el movimiento vertical para seleccionar una de las opciones posibles.

b) Ritmos Sensoriomotores

Al contrario que los potenciales SCP, los ritmos sensoriomotores μ y β permiten controlar un cursor en dos dimensiones habiendo realizado un entrenamiento previamente. De esta forma, esta señal de control se puede utilizar para desplazarse a través de menús complejos, controlar una prótesis o una silla de ruedas o incluso pueden aplicarse en videojuegos, como se muestra en la Figura 2-16 [17].

Figura 2-15 Aplicación de selección de letras utilizando potenciales corticales lentos (SCP) con selecciones binarias. La mitad de los caracteres se muestran en morado y la otra mitad se muestran en azul [16].

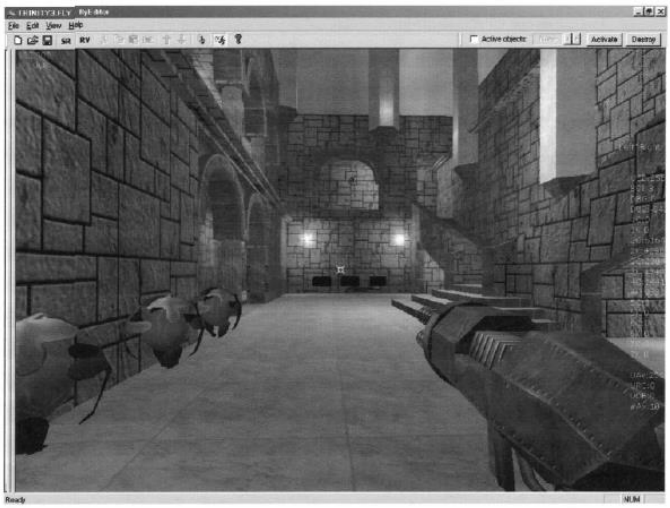


Figura 2-16. Aplicación de movimiento de un cursor para un videojuego [17].

5.3. Desplazamiento sobre un mapa

Las aplicaciones de desplazamiento sobre un mapa son una extensión de las aplicaciones en las que hay que desplazar un cursor. Normalmente, en este tipo de aplicaciones, el usuario debe situarse frente a una pantalla en la que se muestra un mapa por satélite y 4 botones que controlan el movimiento a través de la aplicación.

Las señales de control más utilizadas en este tipo de aplicaciones son los potenciales VEP y los ritmos sensoriomotores.

6. Paradigmas odd-ball

Como hemos visto en el capítulo anterior, una de las aplicaciones más utilizadas de los sistemas BCI es la selección de caracteres. A través de este tipo de aplicaciones, personas con alguna discapacidad o enfermedades neurodegenerativas pueden comunicarse.

En este capítulo, se va a realizar un estudio de los diferentes paradigmas desarrollados para escribir mediante las ondas cerebrales utilizados en los sistemas BCI. Después, a partir de los resultados obtenidos, se va a realizar una comparativa para decidir cuáles serán los paradigmas implementados en las aplicaciones desarrolladas en este Trabajo Fin de Grado.

6.1. Row-Column Paradigm

El paradigma tradicional RCP consta de una matriz de tamaño 6x6 formada por caracteres y comandos, como se muestra en la Figura 2-17, que se iluminan de forma aleatoria agrupados en filas o columnas. El usuario debe prestar atención al carácter que desea seleccionar [18].

Este paradigma está sujeto a errores que ralentizan la comunicación, producen la frustración del sujeto y disminuyen su atención. Los errores más habituales suelen producirse en ubicaciones adyacentes a la del carácter que se desea seleccionar y casi siempre en la misma fila o columna. Este error inherente al paradigma RCP se produce debido a que, cada vez que el carácter que se desea seleccionar parpadea (iluminación), se genera un P300 para cada carácter de la fila o columna. Sin embargo, solo la intersección de la fila y columna del carácter deseado es válida. Los errores surgen cuando se iluminan las filas o columnas a las que no pertenece el car'acter deseado y son adyacentes a este, atrayendo la atención del sujeto y produciendo una respuesta P300 equívoca. A estos errores se les conoce como errores debidos a distracciones adyacentes [19].

Otro error frecuente producido en este paradigma es la iluminación consecutiva del carácter objetivo. Si la fila del carácter que se desea seleccionar se ilumina y, a continuación, se ilumina la columna, la segunda iluminación puede pasar desapercibida y no generar una respuesta P300 lo suficientemente potente como se desea. A este problema se le conoce como problema del *doble-flash* [19].

A consecuencia de esta problemática, se han investigado diversas modificaciones. En los siguientes apartados se describen algunas de las alternativas al paradigma tradicional que proponen eliminar los principales problemas que afectan a dicho paradigma.

6.2. Checkerboard Paradigm

El paradigma del tablero de ajedrez (*CheckerBoard Paradigm*, CBP) se propone como solución a los problemas de precisión y velocidad del paradigma tradicional (RCP). La hipótesis en la que se basa este paradigma es que puede obtener mayor precisión en comparación con el RCP porque evita los errores debidos a distracciones adyacentes y el problema del *doble-flash*, a los que RCP es propenso.

En este estudio, el CBP utiliza una matriz de tamaño 8x9 que contiene 72 caracteres. Los caracteres que conforman la matriz se superponen virtualmente con un tablero de ajedrez. A partir de esto, los caracteres pertenecientes a las celdas blancas se aíslan en una matriz blanca de tamaño 6x6 y los de las celdas negras en una matriz negra de tamaño 6x6, como se muestra en la Figura 2-

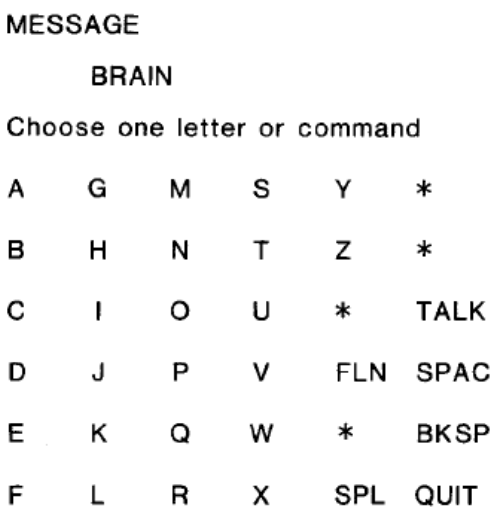


Figura 2-17. Interfaz del paradigma RCP para la selección de caracteres y comandos [18].

18 (A). Antes de que comience la secuencia de iluminación, los caracteres se distribuyen de forma aleatoria en las matrices blanca y negra. En la Figura 2-18(B), se muestra la interfaz que visualiza el sujeto con un grupo de 6 caracteres iluminándose [19].

Durante la selección de un carácter, la secuencia de iluminaciones comienza iluminando las 6 filas virtuales de la matriz blanca, seguidas por las 6 filas virtuales de la matriz negra. Después, pasan a iluminarse las 6 columnas virtuales de la matriz blanca, seguidas por las 6 columnas virtuales de la matriz negra. De esta forma, ningún carácter se puede iluminar consecutivamente durante un mínimo de 6 iluminaciones y un máximo de 18, eliminando el problema del *doble-flash*. Además, esta configuración evita el solapamiento de las épocas objetivo ya que 6 iluminaciones intermedias equivalen a 750 ms, mientras que, en este caso se están utilizando épocas de clasificación de 800 ms.

Otro fenómeno destacado que ocurre en este paradigma es la reducción producida en el número de caracteres adyacentes seleccionados, es decir, los errores por distracciones adyacentes. Como se muestra en Figura 2-19, existen tres tipos de errores identificados por celdas de color gris o de color blanco. Los números en las celdas negras representan el número de selecciones correctas [19].

En el paradigma tradicional, los errores de primer grado (gris oscuro) son los que ocurren en las celdas adyacentes al carácter objetivo y los errores de segundo grado (gris claro) son los que ocurren en cualquier otra localización dentro de la misma fila o columna. En el paradigma CBP, los errores de primer grado son los que ocurren en las celdas diagonales al carácter objetivo, ya que esas celdas se iluminan al mismo tiempo que dicha celda; y los errores de segundo grado son aquellos que ocurren en cualquier otra ubicación de la matriz.

La aplicación se evaluó por parte de 18 sujetos sanos y 3 sujetos con la enfermedad de ELA. Dos de los sujetos con ELA estaban totalmente paralizadas y otro conservaba una leve contracción de la ceja. En todas las sesiones de evaluación, los sujetos debían contar el número de iluminaciones del carácter que debía seleccionar [19].

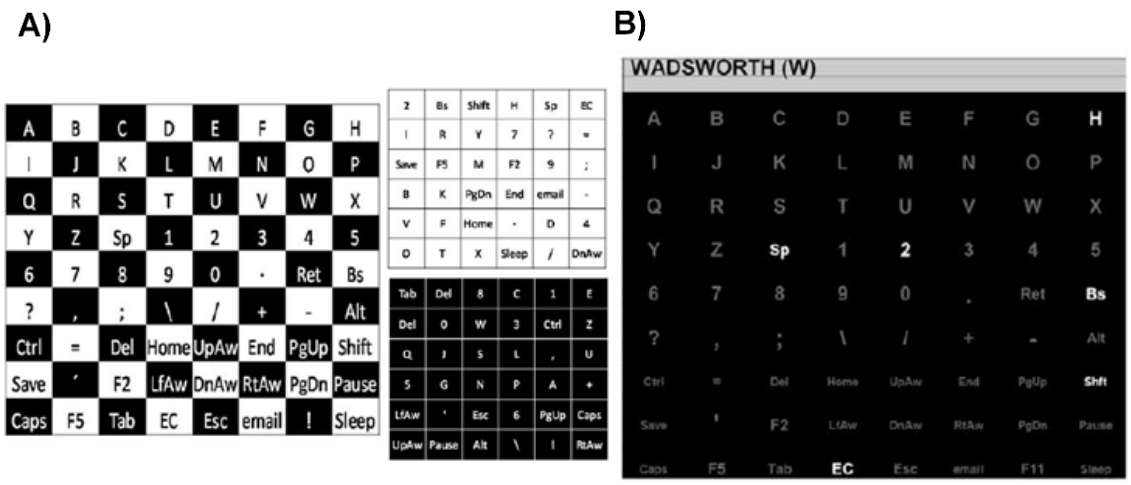


Figura 2-18. A) Matriz virtual generada al superponer la matriz original con el tablero de ajedrez. B) Matriz final que visualizan los usuarios con los caracteres de la primera columna de la matriz blanca iluminándose [19].

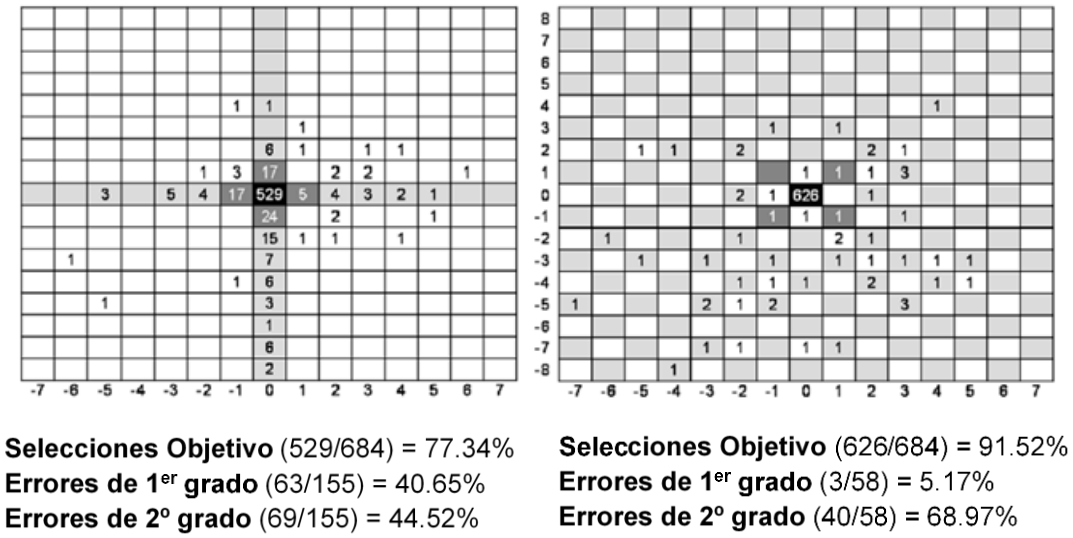


Figura 2-19. Disposición de los errores en el paradigma tradicional (izquierda) y en el paradigma CBP (derecha) [19].

Se realizaron dos sesiones de evaluación, una por paradigma, comenzando con la fase de calibración y, posteriormente, pasando a la fase de test. En la fase de calibración, los sujetos debían escribir 5 cadenas de caracteres, cuatro palabras y una cadena numérica, con una pausa entre caracteres de 3.5 segundos. A partir de la fase de calibración se realizó el entrenamiento del clasificador. En la fase de test se llevó a cabo el mismo procedimiento, excepto por dos diferencias. El número de palabras que se debían escribir fue un número diferente para cada participante. Además, las selecciones se clasificaron utilizando el sistema de pesos del SWLDA (*StepWise Linear Discriminant Analysis*, SWLDA) y, al finalizar cada selección, se proporcionó al usuario la retroalimentación visual de las selecciones.

Una vez se hubo probado el funcionamiento del paradigma CBP en sujetos sanos, se probó en los sujetos con ELA. Para ello, se realizó una única sesión con una persona gravemente discapacitada

por ELA en la que se obtuvo una precisión promedio del 89%, mejorando la precisión en un 27% con respecto al RCP. Después de esto, no se realizaron sesiones RCP adicionales [19].

El procedimiento de evaluación llevado a cabo con los sujetos con ELA fue diferente al de los sujetos sanos. En cada sesión, los sujetos utilizaron un número diferente de selecciones. Al usuario 1 se le proporcionó una cantidad variable de caracteres, mientras que a los usuarios 2 y 3 se les proporcionaron 19 caracteres, aunque no fueron los mismos para cada usuario.

Los resultados obtenidos para los sujetos sanos fueron significativamente mejores para el paradigma CBP, cuya precisión es del 91.52 %; que para el paradigma tradicional, cuya precisión es del 77.34%. Además de la precisión, también se calculó la tasa de transferencia práctica, en la que se incluye la pausa de 3.5 segundos entre selecciones. Los resultados de la tasa de transferencia práctica obtenidos en el test *online* fueron de 22.59 bits/min, para el CBP; y 16.61 bits/min para el RCP [19].

En cuanto a la tasa de transferencia teórica, el CBP logró obtener una tasa de 23.17 bits/min, siendo la tasa del RCP de 19.85 bits/min. Como se puede observar, la diferencia entre paradigmas no es tan significativa, lo que podría ser debido al hecho de que el CBP tarda aproximadamente un 30% más en presentar una secuencia de iluminaciones[19].

Al disociar las filas y columnas, el CBP reduce las tasas de error, presumiblemente al eliminar los errores debidos a distracciones adyacentes. Los resultados obtenidos muestran que la mayoría de los errores del paradigma CBP fueron errores de segundo grado, dentro de los cuales el 74.14% se produce en la misma matriz y el 25.85% restante en la matriz opuesta. Este resultado sugiere que en el paradigma CBP, la proximidad temporal al carácter objetivo es mucho más importante que la proximidad espacial. Por el contrario, en el RCP, la proximidad espacial y la proximidad temporal ocurren juntas.

En cuanto a los resultados obtenidos por los sujetos con ELA, los datos preliminares sugirieron que se obtienen mejores resultados con el paradigma CBP que con el paradigma tradicional. La precisión promediada en sujetos con ELA aumenta un 24.60% para el paradigma CBP en relación con el paradigma RCP, mientras que, en los sujetos sanos, la precisión aumenta solamente un 14.18%. La tasa de transferencia no se calculó para las sesiones del RCP porque los parámetros se manipularon continuamente para optimizar el rendimiento [19].

6.3. Hex-o-Spell Paradigm

El paradigma del hexágono (*Hex-o-Spell*, HoS) se propone para estudiar la dependencia de los sistemas BCI con la forma de atención visual. La hipótesis de este paradigma es que puede reducir el amontonamiento, debido a la presentación de pocos símbolos y de grandes dimensiones, y previene la disminución de la atención [20].

Para analizar cómo dependen los sistemas BCI de la atención visual, se estudiaron dos modos de atención. En la atención abierta, también llamada *overt attention*, se analiza la dependencia con los movimientos oculares; mientras que, en la atención cubierta, o c*overt attention*, se analiza la dependencia con respecto a la visión periférica. Este último modo de atención tiene dos peculiaridades. La primera peculiaridad es la disminución de la agudeza espacial producida por el aumento de excentricidad visual o visión en detalle. La segunda peculiaridad es el efecto de

"*crowding*", que es un fenómeno en el que la identificación de objetos en la periferia visual se ve obstaculizada si se estos se encuentran rodeados por objetos similares.

El paradigma HoS consta de 6 discos dispuestos en forma circular y un punto situado en el centro del círculo que forman los 6 discos. Como se muestra en la Figura 2-20, el proceso de selección se divide en dos etapas sucesivas. En la primera etapa, cada uno de los 6 discos contiene un quinto de los caracteres. Una vez se ha elegido uno de los 6 discos mostrados en la primera etapa, los caracteres del disco elegido se expanden ocupando los 6 discos para poder seleccionar el carácter deseado. En la segunda etapa, uno de los discos pasa a estar vacío, con el fin de poder regresar a la etapa anterior en caso de que se seleccione un grupo de caracteres incorrecto [20].

En el modo *overt attention*, los sujetos deben fijarse en el disco que desean seleccionar. Sin embargo, en el modo *covert attention*, deben fijarse durante toda la prueba en el punto mostrado en el centro de la pantalla.

La aplicación se evaluó por parte de trece sujetos sanos. Se realizó una única sesión que consistía en una fase de calibración y dos fases de test, *online* y *offline*. En cada una de las fases, los usuarios debían escribir 9 palabras de entre 5 y 6 letras. Cada prueba se dividió en bloques de tres palabras y cada palabra tuvo que escribirse dos veces, una para cada subcondición (*covert attention* y *overt attention*). El orden de los bloques fue aleatorio y entre bloques se realizó una pequeña pausa. Antes de comenzar la prueba, la palabra que se debía escribir se mostró en el centro de la pantalla. Después, apareció una cuenta atrás auditiva de 4 segundos, en los que el sujeto pudo identificar la localización del disco que debía seleccionar. Una vez iniciadas las iluminaciones, la palabra pasaba a mostrarse en la parte superior de la pantalla con la letra que se debía seleccionar resaltada [20].

En todas las sesiones, cada disco fue iluminado 10 veces, tanto en la primera etapa como en la segunda. Para fijar la atención, se aconsejó al usuario que contara el número de iluminaciones del disco que debía seleccionar.

Los resultados obtenidos fueron mejores para el modo *overt attention* que para el *covert attention*, cuya precisión es de alrededor del 60%. Desafortunadamente, la precisión obtenida en la sesión *offline* para el modo de *covert attention* es demasiado baja como para ser un medio de comunicación viable. La precisión aumenta cuanto mayor es el número de iluminaciones.

Para asegurar que los sujetos fijaban la vista de forma adecuada, se monitorizaron los movimientos oculares. Si se detectaba que el usuario no fijaba su atención en el centro de la pantalla durante la secuencia de iluminaciones, se produjo un sonido de advertencia y la prueba fue anulada. Para volver a la prueba, el usuario tenía que presionar un botón cualquiera del teclado [20].

6.4. Variantes de Hex-o-Spell

Las variantes del paradigma HoS se proponen como nuevos métodos para obtener sistemas BCI más rápidos y precisos. Como ya se ha visto, la hipótesis de este tipo de paradigmas es reducir el amontonamiento y prevenir la disminución de la atención [21].

La primera variante estudiada es una extensión directa del HoS, compuesta por 6 discos de diferentes colores. Al mostrar cada disco en un color, el usuario puede atender a la ubicación espacial del disco o atender al color asociado a dicho disco.

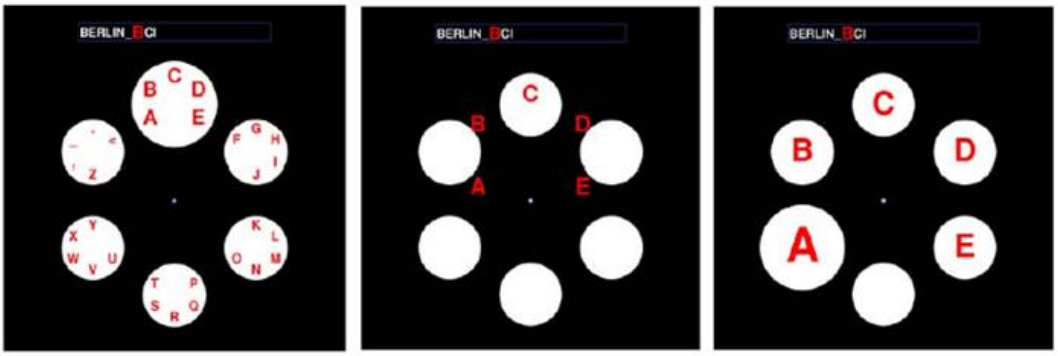


Figura 2-20. Disposición de los discos en el paradigma HoS [20].

La segunda variante, llamada *Cake Speller*, tiene un diseño similar al HoS. En lugar de 6 discos, este paradigma consta de 6 triángulos de diferentes colores que se unen formando un hexágono. Como cada triángulo se extiende hasta el punto central de fijación, se facilita el despliegue de la atención espacial [21].

En la tercera y última variante, llamada *Center Speller*, los discos se reemplazan por formas geométricas, cada una de ellas con un color diferente. A diferencia de la primera y segunda variantes, las formas se presentan de forma secuencial en el centro de la pantalla.

Como se muestra en la Figura 2-21, en todas las variantes se utilizó el mismo conjunto de 6 colores. Los sujetos debían fijar su atención en el centro de la pantalla o en los símbolos presentados en esa localización, y contar el número de veces que se iluminaba el objetivo [21].

La aplicación se evaluó por parte de trece sujetos sanos. Se realizaron tres sesiones de evaluación, una primera sesión de calibración, posteriormente, una sesión de test *online* y, para finalizar, una sesión libre. Antes del comienzo de cada selección, se realizó una cuenta atrás de 5 segundos. En la sesión de calibración, los usuarios debían escribir 3 palabras para realizar el entrenamiento del clasificador. A continuación, se realiza la sesión de prueba *online*. En esta sesión, los sujetos debían escribir una frase incluyendo espacios. Si se seleccionaba un grupo de letras incorrecto en la primera etapa, los sujetos debían utilizar el retroceso para volver a la etapa anterior, al igual que si seleccionaban un carácter incorrecto. Si al realizar una selección incorrecta no se obtuvo éxito en la corrección después de dos intentos, los sujetos debían saltar ese carácter y pasar al siguiente. Para finalizar, en la sesión libre, debían escribir una frase con aproximadamente 20 símbolos.

En todas las sesiones, cada símbolo fue iluminado 10 veces, tanto en la primera etapa como en la segunda. El orden de las iluminaciones fue aleatorio, pero se realizaban 2 iluminaciones intermedias antes de volver a iluminar el mismo símbolo [21].

La precisión promedio obtenida en la sesión de test *online* fue de 91.3% para la variante del HoS, 88.2% para el *Cake Speller* y 97.1% para el *Center Speller*. Como se puede observar, la diferencia de precisión entre el *Cake Speller* y el *Center Speller* es bastante elevada. Esto puede deberse a que en el paradigma *Center Speller*, los símbolos se muestran en el centro de la pantalla, donde el usuario debe fijar su vista. Sin embargo, entre la variante del HoS y el *Cake Speller* la diferencia es bastante menor, a pesar de que el *Cake Speller* debería obtener mejores resultados, puesto que los triángulos

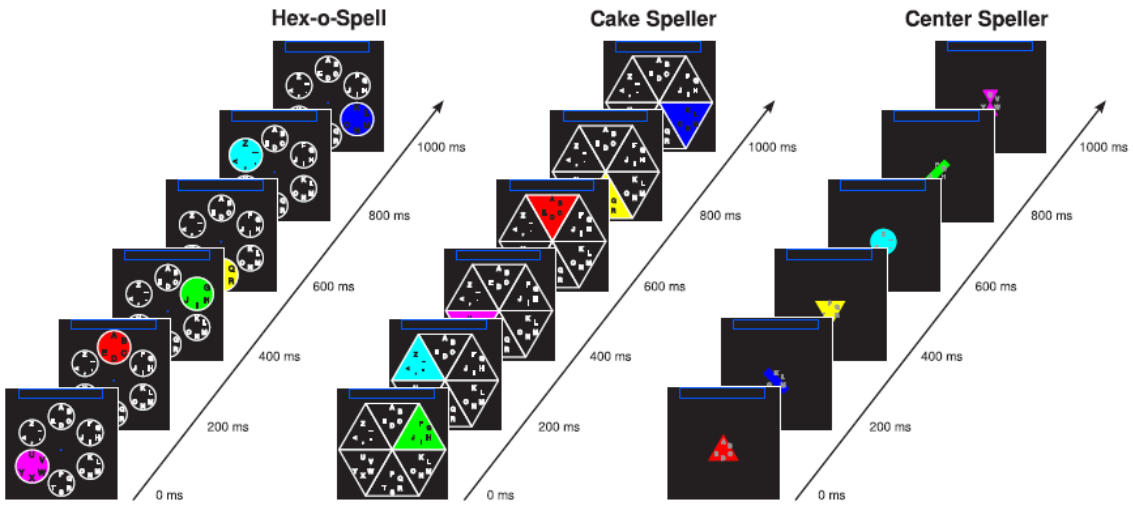


Figura 2-21. Interfaces de selección de caracteres de la variante del HoS, el *Cake Speller* y el *Center Speller*, respectivamente [21].

se extienden hasta el centro de la pantalla, donde el usuario debe tener fija la vista. Además, la precisión aumenta cuanto mayor es el número de iluminaciones [21].

Para asegurar que los sujetos fijaban la vista de forma adecuada, se monitorizaron los movimientos oculares. Si se detectaba que el usuario no fijaba su atención en el centro de la pantalla durante la secuencia de iluminaciones, se produjo un sonido de advertencia y la prueba fue anulada. En este caso, para volver a la prueba, se mostró una cuenta atrás de 5 segundos.

En la variante del HoS y en el *Cake Speller*, los símbolos más seleccionados fueron aquellos cuyo color era rojo, verde o amarillo, lo que sugiere que el contraste podría ser una característica relevante. En el *Center Speller*, el símbolo más seleccionado fue el reloj de arena. Esto puede deberse a la presencia de un mayor número de bordes en el punto en el que los usuarios debían fijarse [21].

6.5. Rapid Serial Visual Paradigm

El paradigma visual en serie rápida (*Rapid Serial Visual Paradigm*, RSVP) se propone como alternativa al método de selección de caracteres tradicional, ya que este método está basado en la atención no espacial, independiente de los movimientos oculares [22].

Como se muestra en la Figura 2-22, en el paradigma RSVP los caracteres se presentan aleatoriamente de uno en uno situados en el centro de la pantalla. Se analizaron tres variantes que diferían con respecto a la asincronía de comienzo de estímulo (*Stimulus Onset Asynchrony*, SOA) y al color de las letras. En la condición de Color se utilizaron dos SOA diferentes, 83 ms y 116 ms, para estudiar cómo influye esta característica a la hora de seleccionar un carácter. Los caracteres se mostraban de 5 colores diferentes, rojo, blanco, azul, verde y negro. En la condición de NoColor todos los caracteres eran de color negro. En todas las variantes, los usuarios debían concentrarse en la letra objetivo y contar el número de veces que se iluminaba.

La aplicación se evaluó por parte de doce sujetos sanos en tres fases diferentes, la fase de calibración, la fase de test *online* y, para finalizar, una fase libre [22].

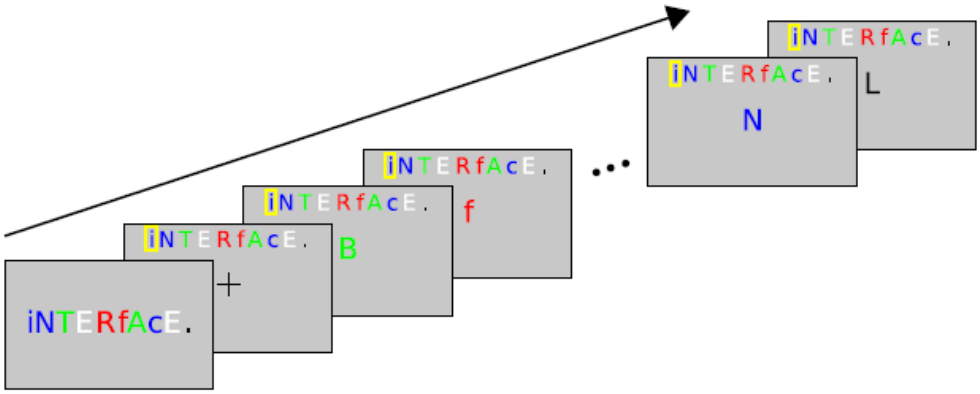


Figura 2-22. Interfaz para la selección de caracteres en el paradigma RSVP [22].

En la fase de calibración, los usuarios debían escribir una frase. Antes de comenzar la prueba, la frase se mostró en el centro de la pantalla durante dos segundos. Al comenzar la secuencia de iluminaciones, la frase pasó a mostrarse en la parte superior de la pantalla con la letra que se debe seleccionar resaltada. En ese momento, el sujeto dispone de 4 segundos para identificar la letra que debe seleccionar, ya que a continuación, aparece una cruz de fijación en el centro de la pantalla.

En la fase de test *online*, el sujeto debía escribir una frase diferente para cada paradigma. El resto del procedimiento fue igual que en la fase anterior. Al finalizar cada secuencia de iluminaciones, en la pantalla se mostró el carácter que el sujeto había seleccionado [22].

En la fase libre, se debía escribir una frase por cada paradigma con al menos 15 caracteres. Como en la sesión de test *online*, al finalizar la secuencia de iluminaciones, se mostró el carácter que el sujeto había seleccionado. Si el clasificador seleccionaba una letra incorrecta, el usuario debía usar la siguiente prueba para seleccionar el símbolo de retroceso y borrarla.

En todas las sesiones, cada símbolo fue iluminado 10 veces. Entre cada secuencia de 10 iluminaciones se realiza una breve pausa de 0.3 segundos [22].

Los resultados obtenidos en la fase de test *online* demuestran que la precisión aumenta cuanto mayor es el número de iluminaciones. Las precisiones logradas con cada uno de los paradigmas fueron 94.8% en NoColor 116 ms, 94.7 % en Color 116 ms y 93.6% en Color 83 ms. La tasa de bit promedio es de 5.65 bits/min en el paradigma NoColor 116 ms, 5.66 bits/min en el paradigma Color 116 ms y 7 bits/min en el paradigma Color 83 ms.

A través de estos resultados se demuestra que la utilización de mayúsculas y minúsculas y colores facilitan la discriminación de los caracteres. A partir de estos resultados, se puede concluir que SOA más rápidos logran una mayor transferencia de información (bits/min), pero reducen su precisión [22].

6.6. Honeycomb-Shaped Red Dots Paradigm

El paradigma de los puntos rojos con forma de panal (*Honeycomb-Shaped Red Dots*, HSRD) se propone como método alternativo para ayudar a los usuarios a enfocar su atención en el estímulo visual [23].

Como se muestra en la Figura 2-23, en el paradigma HSRD se sustituye la iluminación de un carácter de la matriz 6x6 por la aparición de un panal de color verde, en cuyo interior se encuentran distribuidos puntos de color rojo. El número de puntos varía de 1 a 3 de forma aleatoria, lo que permite que el usuario pueda contarlos de una forma rápida y sencilla. El patrón de puntos generado es el mismo para todos los caracteres en cada iluminación.

Debido a que la posición y el número de puntos rojos cambia de forma aleatoria en cada iluminación, los usuarios no pueden predecir el siguiente estímulo y se reducen las distracciones producidas por caracteres adyacentes [23].

La aplicación se evaluó por parte de 10 sujetos sanos. Se realizaron dos sesiones, la sesión de entrenamiento *offline* y la sesión de test *online*.

En la sesión *offline*, los sujetos debían escribir 3 palabras de 5 letras cada palabra, utilizando una secuencia de 16 iluminaciones. Después de esto, el sistema registró y extrajo las características de la señal EEG de los usuarios. Para realizar la clasificación se utilizó el análisis discriminante bayesiano (*Bayesian Linear Discriminant Analysis*, BLDA), una extensión del discriminante lineal de Fisher (FLD), puesto que posee una buen rendimiento en sistemas BCI basados en potenciales P300 [23].

En la sesión *online*, los sujetos debían escribir una frase formada por 36 letras. La secuencia de iluminaciones era de un número variable, por lo que fue necesario utilizar un sistema de adaptación, el cual seleccionaba de forma automática el número de iluminaciones en función de la salida del clasificador. El número máximo de iluminaciones fue 16, aunque el promedio fueron 6 o 7.

Los resultados *online* y *offline* muestran que la precisión de clasificación y la tasa de transferencia de información del paradigma HSRD pueden superar fácilmente los resultados obtenidos con el paradigma tradicional. Los resultados obtenidos en cuanto a la precisión, la tasa de bit y la tasa de bit práctica son 89.7%, 38.0 bits/min y 20.9 bits/min, respectivamente. Como ocurre en otros estudios, la precisión aumenta cuanto mayor es el número de iluminaciones.

Dado que en otros estudios se utilizaron diversos parámetros para el filtrado paso-banda, se probaron diferentes bandas de paso para encontrar la banda óptima a utilizar con los potenciales evocados P300. Finalmente, la banda que obtuvo la mayor precisión de clasificación fue 1-12 Hz, seguida por la banda 1-30 Hz, utilizada en este estudio [23].

6.7. Comparativa de los paradigmas odd-ball

Analizando los resultados obtenidos en cada uno de los estudios anteriores, se ha realizado una tabla comparativa (Tabla 2-3) en la que se muestran los diferentes valores de precisión y la tasa de transferencia conseguida por cada paradigma.

A la hora de comparar, hay que tener en cuenta que cada uno de los estudios utilizó un número diferente de sujetos, por lo que las precisiones podrían variar si todos utilizaran el mismo número de sujetos.

A pesar de que los tres paradigmas que mayor precisión consiguen son los del paradigma RSVP con un 94.8%, 94.7% y 93.6%, su tasa de transferencia es muy pequeña en comparación con el resto de paradigmas. En cuanto a las variantes del HoS, no se pueden comparar con el resto de paradigmas puesto que las tasas de transferencia obtenidas no son realistas para uso práctico.

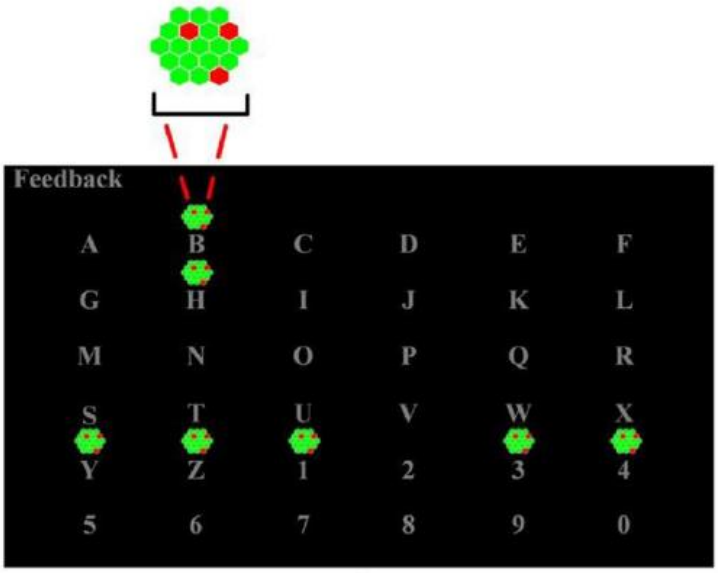


Figura 2-23. Interfaz utilizada en el paradigma HSRD. Los panales con puntos rojos aparecen cuando el carácter se ilumina [23].

TABLA 2-3. Comparativa de paradigmas *odd-ball*.

Paradigma		Propiedades	
		Precisión (%)	**Tasa de transferencia (bits/min)**
RCP		77.32	19.85
CBP		91.52	23.17
HoS		69.00	-
Variantes HoS	**Variante HoS**	91.30	-
	Cake Speller	88.20	-
	Center Speller	91.70	-
RSVP	**NoColor 116 ms**	94.80	5.65
	Color 116 ms	94.70	5.66
	Color 83 ms	93.60	7.00
HSRD		89.70	20.90

ANÁLISIS DE POTENCIALES EVOCADOS P300

1. Potenciales evocados P300

En el capítulo anterior, se ha introducido el concepto de potenciales evocados P300, así como sus características principales. Los potenciales evocados P300 son picos de voltaje positivos que se producen en el EEG del usuario, como respuesta a un estímulo, alrededor de 300 ms después de que éste se produzca.

Estos potenciales suelen tener una amplitud en torno a los 10 µV y se generan sobre la zona parietal del córtex, por lo que para registrar la señal se utilizan los electrodos situados sobre esta región, llamados Cz, Fz y Pz [5]. Analizando el tiempo de reacción (*Reaction Time*, RT) del sujeto, definido como el tiempo requerido para que los impulsos viajen desde el cerebro hacia los músculos, se ha demostrado que se produce aproximadamente 50 ms antes del potencial evocado P300. Además, también se le conoce como potencial P3 ya que es el pico con la tercera amplitud más alta entre los potenciales evocados sensoriales [24].

Para provocar la aparición de los potenciales evocados P300 se utiliza el paradigma *odd-ball*. Este paradigma consiste en presentar el estímulo deseado de forma infrecuente entre estímulos frecuentes no deseados. Cuanto más improbable sea el estímulo deseado, más probable será la aparición del P300 y que su amplitud sea mayor [5].

A lo largo de los años se ha tratado de caracterizar el potencial evocado P300, así como la forma de obtenerlo. Finalmente, se ha concluido que la mejor forma de eliminar las variaciones lentas de la señal es utilizando un filtro paso-alto con una frecuencia de corte de 0.1 Hz. Filtros con frecuencias de corte superiores a 0.5 Hz distorsionan el P300, disminuyendo su amplitud y haciendo que la parte posterior de la señal se vuelva negativa, debido a la distorsión de fase [24].

Los potenciales P300 suelen tener una respuesta de pequeña amplitud y suelen aparecer superpuestos con ruido y actividad EEG de fondo. Debido a esto, una respuesta individual puede no suele ser reconocible. Sin embargo, es necesario poder distinguir la respuesta evocada por el estímulo de la actividad EEG de fondo [25].

Una vez que se han analizado los artefactos más importantes que degradan la señal, se pueden analizar técnicas de filtrado que los eliminen. Sabiendo que la señal P300 está solapada en frecuencia con el contenido espectral del EEG de fondo, no va a ser posible aplicar una técnica de filtrado en frecuencia eficaz para obtener la respuesta evocada.

Una alternativa a este filtrado frecuencial es el filtrado en el dominio del tiempo, ya que existe una relación temporal repetitiva a lo largo de las diferentes épocas de la señal obtenida, entre la aparición del estímulo y la respuesta evocada. Típicamente, la respuesta evocada aparece 300 ms después de la presentación del estímulo. Este filtrado temporal se conoce como promediado sincronizado y consiste en promediar todas las épocas de la señal estableciendo como referencia el punto en el que se ha generado el estímulo que produce el potencial P300 [25].

Dentro de la adquisición de la señal, es importante eliminar los nuevos artefactos que puedan estar contaminándola. Para eliminar el movimiento de los ojos y los pestañeos producidos por el usuario, que degradan los potenciales evocados P300, se pueden utilizar dos enfoques [24].

El primer enfoque se basa en rechazar las épocas en las que se hayan producido movimientos oculares o pestañeos. Para ello, se monitoriza el EOG utilizando electrodos cerca de los ojos, localizados encima y debajo de estos, y se rechazan las épocas si el EOG supera un umbral, por ejemplo, de 100 μV [24]. Este enfoque tiene dos inconvenientes principales. El primer inconveniente se debe a que rechazar épocas de la señal reduce la eficiencia del registro (en el peor de los casos, todas las épocas serían rechazadas), por lo que se recomienda a los sujetos que no muevan los ojos mientras se está realizando la prueba. Esto conduce al segundo inconveniente, ya que al recomendar al sujeto que limite los movimientos oculares, cambia la naturaleza de la prueba, haciendo que el sujeto divida su atención entre los ojos y el estímulo, y produciendo un cambio en la forma del P300.

El segundo enfoque consiste en restar la señal EOG del EEG. Para ello se suelen utilizar filtros adaptativos, que requieren una señal de referencia fuertemente correlada con el artefacto que se desea eliminar. En este caso, se quiere eliminar la señal EOG registrada a través de los electrodos situados alrededor del ojo. El problema de este enfoque es que la eliminación del EOG no es perfecta sino que, eliminará ciertas partes de la señal EEG recogidas en los electrodos frontales, los electrodos más cercanos a los ojos [24].

La onda P300 se mide en términos de una amplitud máxima, relativa a una línea base llamada actividad basal. El pico de amplitud se identifica como el punto más positivo en la forma de onda retrasado 200 ms o más después de la presentación del estímulo. La detección se realiza utilizando un solo electrodo, típicamente Cz o Pz. La latencia del P300 varía de un electrodo a otro, alcanzando su valor mínimo en los electrodos situados en la zona frontal del córtex. Desafortunadamente, esta forma de tratar los potenciales P300 no tiene en cuenta la posibilidad de que éstos estén conformados por múltiples procesos generados en diferentes ubicaciones del córtex. Un estudio más detallado revela que la onda P300 está compuesta por la superposición de 3 ondas independientes y positivas: la onda P3a con un máximo cerca de 250 ms, la onda P3b con un máximo cerca de 350 ms y la onda lenta positiva (*Slow Wave*, SW). En la Figura 3-1 se muestra el potencial P300 con sus respectivas componentes.

Estas componentes se consideran independientes porque tienen distintas relaciones con las variables experimentales. La onda P3a se distribuye en la zona frontal del cuero cabelludo, mientras que la onda lenta lo hace en la zona parietal. Aunque las tres componentes varían de la misma manera con la probabilidad de estímulo, aumentando su amplitud a medida que el objetivo se hace más improbable, son sensibles a la información proporcionada por dicho estímulo. La componente P3a no se ve afectada si el sujeto está atendiendo o no al estímulo. Sin embargo, la componente P3b y la onda lenta aumentan su amplitud si el sujeto presta atención, como se puede observar en la Figura 3-1. Las componentes se pueden distinguir por separado utilizando el análisis de componentes principales (*Principal Component Analysis*, PCA) [24].

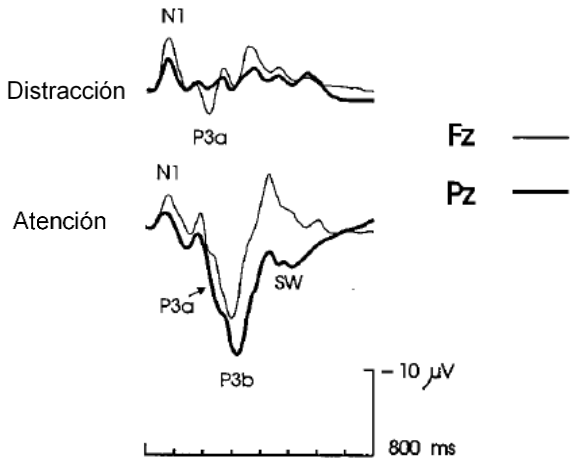


Figura 3-1. Componentes del potencial evocado P300 en los electros Fz y Pz. En la gráfica superior aparece la respuesta cuando el sujeto ignora el estímulo, donde se genera la componente P3a. En la gráfica inferior aparece la respuesta cuando el sujeto atiende el estímulo. En este caso, se observa que se generan las 3 componentes de la onda P300, P3a, P3b y la onda lenta positiva [24].

A continuación, se muestra cómo varía la forma del potencial evocado P300 dependiendo de cuáles sean las características del sujeto, atendiendo a los aspectos psicológicos, las enfermedades neuronales, los factores farmacológicos y las diferencias entre sujetos.

1.1. Aspectos psicológicos

Los aspectos psicológicos del sujeto pueden afectar a la forma de la onda P300. Es importante destacar los efectos que producen la atención del sujeto y la probabilidad de que ocurra el estímulo que genera el P300. Varios estudios han demostrado que la amplitud del potencial P300 aumenta cuanto mayor es el nivel de atención que presta el sujeto al estímulo y cuanto más improbable es la aparición de dicho estímulo objetivo. En la Figura 3-2 (A), se muestra el efecto de la probabilidad del estímulo en la onda P300 [24].

Los efectos que produce la probabilidad del estímulo ocurren independientemente de que el sujeto los conozca, es decir, si se modifica la probabilidad de estímulo en una prueba, aunque el usuario no tenga conocimiento del cambio, la amplitud del P300 se verá modificada.

Otro aspecto relacionado con la probabilidad del estímulo que influye en el P300 es la probabilidad temporal, el tiempo entre la aparición de dos estímulos objetivo. Como se muestra en la Figura 3-2 (B), cuando mayor es la probabilidad temporal, mayor es la amplitud de la onda P300 [24].

Además de estos aspectos, cabe destacar el efecto que produce el nivel de dificultad de la tarea que debe realizar el sujeto. Cuando discriminar el estímulo objetivo se vuelve una tarea complicada, la amplitud del potencial P300 disminuye, mientras que la latencia aumenta. Este efecto se puede observar en la Figura 3-2 (C).

La disminución en la amplitud del P300 se debe, principalmente, a la disminución en la confianza del usuario. Cuanto más confiando se encuentre el sujeto a la hora de discriminar el estímulo objetivo, mayor será la amplitud de la onda P300 generada. Sin embargo, si la prueba pasa a ser

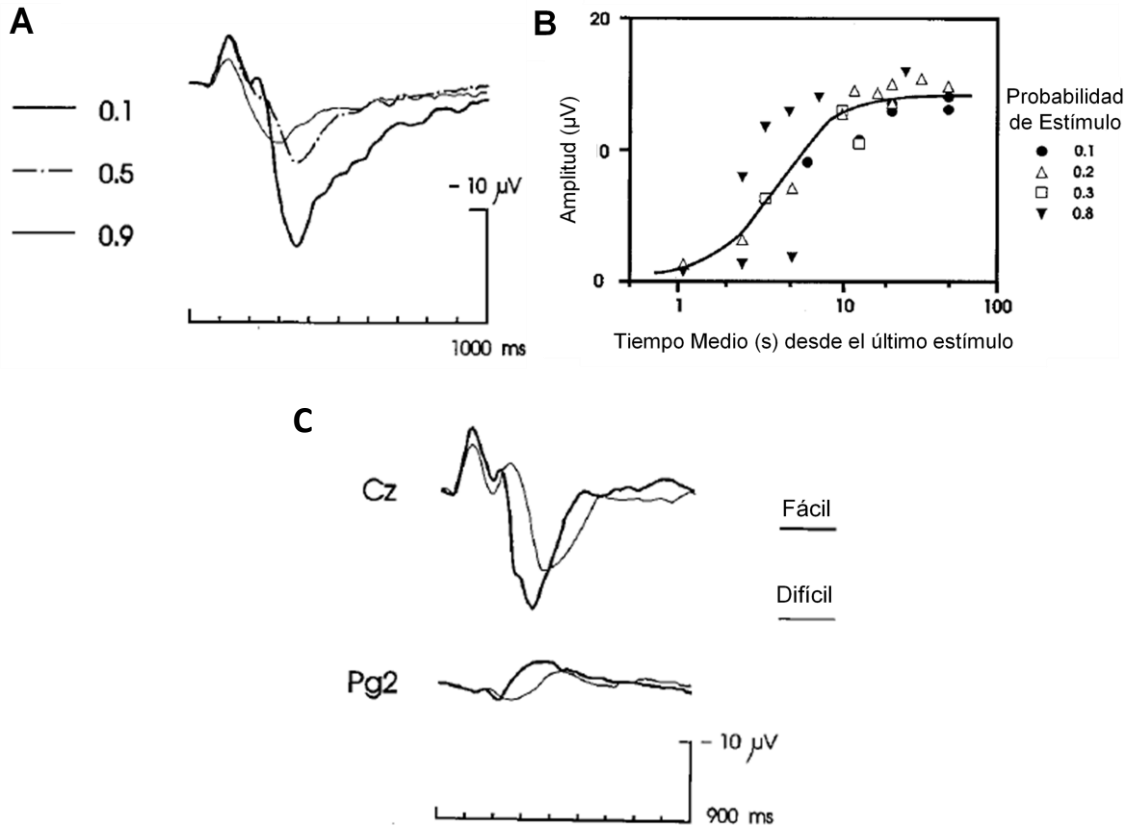


Figura 3-2. Variación en la forma del potencial evocado P300 producida en el electrodo Pz, dependiendo de la probabilidad de estímulo (A) y de la probabilidad temporal (B), cuando se utiliza un paradigma *odd-ball* auditivo. (C) Potencial evocado P300 registrado en los electrodos Cz y Pg2, electrodo nasofaríngeo. Forma de onda dependiendo de la dificultad que tenga la discriminación de los estímulos [25].

demasiado fácil, el usuario desviará su atención del objetivo haciendo que la amplitud disminuya [24].

En la Tabla 3-1, se muestra un resumen de los aspectos psicológicos que modifican la forma del potencial evocado P300.

1.2. Enfermedades neuronales

Las enfermedades neuronales que más afectan a la forma del potencial evocado P300 son la esquizofrenia, el trastorno obsesivo-compulsivo, la depresión, el autismo, la psicopatía o la demencia. A continuación, se detallan los efectos que producen.

Varios estudios han demostrado que, en enfermos de esquizofrenia, la amplitud del potencial P300 se reduce notablemente. Esta reducción en la amplitud no parece estar relacionada con la falta de atención o la medicación, sino con la gravedad de los síntomas del paciente. Además, en los sujetos con esquizofrenia, el P300 no suele estar concentrado en la zona parietal del córtex, como ocurre con los sujetos control. Esto sugiere que los pacientes esquizofrénicos suelen tener anormalidades en el lóbulo temporal [24].

TABLA 3-1. Aspectos psicológicos que modifican la forma del potencial evocado P300.

Aspectos Psicológicos		
	Amplitud del P300	**Latencia del P300**
↑ Atención	Aumenta	
↓ Probabilidad de Estímulo	Aumenta	
↑ Probabilidad Temporal	Aumenta	
↑ Dificultad de la Tarea	Disminuye	Aumenta

En cuanto a los trastornos psiquiátricos, el autismo y la depresión severa producen una reducción en la amplitud de la onda P300. Sin embargo, en pacientes con trastorno obsesivo-compulsivo la reducción se produce en la latencia del P300. Esto puede ser debido a alguna hiperactividad en sus sistemas perceptivos.

Por otro lado, tanto los pacientes psicopáticos como los enfermos de ELA poseen un P300 mucho más prolongado que los sujetos control [24]. Al igual que ocurre en los enfermos con trastornos psiquiátricos, los pacientes con demencia suelen tener una onda P300 con menor amplitud y con mayor latencia.

Por último, se ha demostrado que aquellos niños que padecen trastorno por déficit de atención o dislexia poseen una onda P300 de menor amplitud. En el caso de niños con dislexia, el P300 suele ser estar más retrasado [24].

En la Tabla 3-2, se muestra un resumen de las enfermedades neuronales que modifican la forma del potencial evocado P300.

1.3. Factores farmacológicos

Entre los factores farmacológicos que más afectan a la forma del potencial P300 destacan el alcohol, las drogas y los medicamentos.

Muchos estudios han demostrado que el alcohol produce una disminución en la amplitud del P300 y un aumento en la latencia. Sin embargo, cuando la dificultad de la tarea de discriminación entre estímulos es elevada, no se produce este efecto, ya que el sujeto intente superar los efectos del alcohol cuando la tarea lo requiere [24].

En cuanto a los efectos producidos por los medicamentos destaca el efecto de la escopolamina, también conocida como burundanga, que produce una disminución significativa en la amplitud de la onda P300. Este medicamento puede llegar a producir una disminución tan importante que puede hacerlo desaparecer cuando el usuario está realizando una tarea auditiva. También, se ha demostrado que reduce la capacidad de recordar durante cortos periodos de tiempo.

TABLA 3-2. Enfermedades neuronales que modifican la forma del potencial evocado P300.

Enfermedades Neuronales			
	Amplitud del P300	**Latencia del P300**	**Otros efectos**
Esquizofrenia	Disminuye (puede desaparecer)		
Autismo	Disminuye		
Depresión Severa	Disminuye		
Trastorno Obsesivo – Compulsivo		Disminuye	
Psicopatía	Disminuye		P300 más prologado
ELA	Disminuye		P300 más prologado
Demencia	Disminuye	Aumenta	
Trastorno por Déficit de Atención	Disminuye		
Dislexia	Disminuye		P300 más retrasado

Otros medicamentos que también destacan por sus efectos son el metilfenidato, un psicoestimulante utilizado en el tratamiento de la hiperactividad, que acelera el RT del sujeto; la clonidina, utilizado como fármaco para personas con hipertensión, las benzodiacepinas, medicamentos psicotrópicos, y los antihistamínicos, que producen una reducción en la amplitud del P300; y la dopamina, que disminuye su latencia [24].

En la Tabla 3-3, se muestra un resumen de los factores farmacológicos que modifican la forma del potencial evocado P300.

1.4. Diferencias individuales entre sujetos

Las diferencias individuales entre sujetos también pueden afectar al potencial evocado P300. Entre estas diferencias destacan la edad, la inteligencia, la personalidad o el oído absoluto.

Como se muestra en la Figura 3-3 (A), el comportamiento del potencial P300 varía de forma distinta en los niños y en los adultos, siendo complicado determinar cuál es su comportamiento en los niños debido a que estos poseen un gran número de artefactos y, además, no son capaces de prestar atención de forma constante [24].

A medida que los niños (edades entre 5 y 12 años) se desarrollan, la latencia de la onda positiva producida en la zona parietal del córtex disminuye. Esta disminución en la latencia es más importante entre los 5 y los 12 años, cuando la latencia del P300 disminuye a 25 ms por año.

TABLA 3-3. Factores farmacológicos que modifican la forma del potencial evocado P300.

Factores Farmacológicos			
	Amplitud del P300	**Latencia del P300**	**Otros efectos**
Alcohol	Disminuye	Aumenta	No afecta en tareas complicadas
Escopolamina	Disminuye (puede desaparecer)		Disminuye la memoria a corto plazo
Metilfenidato			Acelera el RT del sujeto
Clonidina	Disminuye		
Benzodiacepinas	Disminuye		
Medicamentos Psicotrópicos	Disminuye		
Antihistamínicos	Disminuye		
Dopamina		Disminuye	

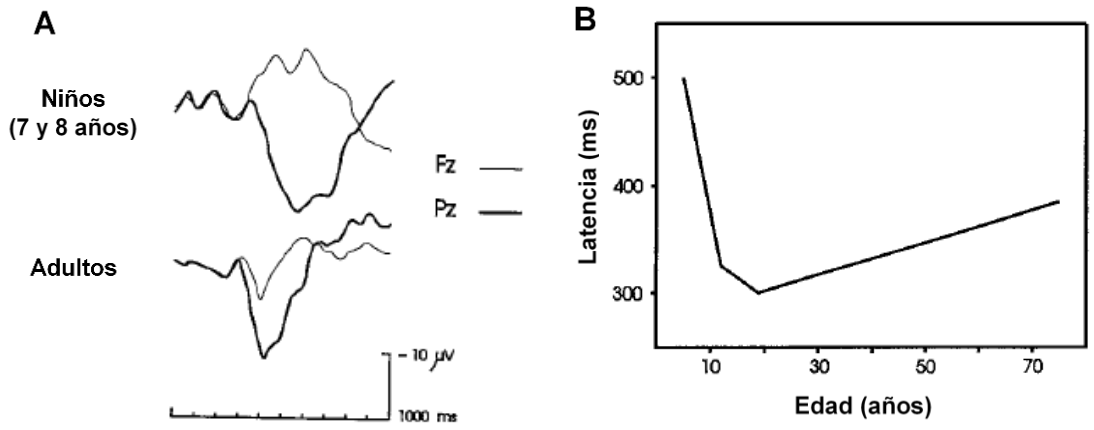


Figura 3-3. Variación del potencial P300 dependiendo de la edad del sujeto. En (A) se muestran los efectos que produce la edad en la latencia de la onda P300. En (B) se muestra el P300 de adultos y niños recogido en los electros Fz y Pz [24].

Una vez que los niños pasan a tener entre 12 y 20 años, se produce una disminución adicional de entre 1 y 5 ms por año. Como se puede observar en la Figura 3-3 (B), los valores mínimos de la latencia ocurren entre los 15 y los 20 años. Para sujetos adultos mayores de 20 años, la latencia del P300 es una regresión lineal significativa, cuya pendiente es de 1.3 ms por año, con un error estándar de 31 ms. Por lo tanto, los cambios en la latencia de la onda P300 son más importantes en niños. Esto puede estar relacionado con la confusión producida entre la componente P3b y la SW [24].

En cuanto a la amplitud del P300, en general, se puede afirmar que la amplitud de la onda P300 disminuye con la edad en sujetos adultos. Esto puede deberse a la variabilidad producida en la latencia. Sin embargo, en los niños, la amplitud del P300 aumenta hasta llegar a una edad de 13 años. A partir de esa edad, el P300 suele disminuir ligeramente hasta alcanzar los valores normales de un adulto [24].

Analizando el RT del sujeto, se ha demostrado que puede cambiar con la edad dependiendo de la dificultad de la tarea a la que se enfrente el sujeto. En tareas simples, la edad aumenta significativamente la latencia del P300 y disminuye débilmente el RT. Por el contrario, en tareas más complejas, el RT se ve más afectado por la edad. Por lo tanto, la dificultad en la tarea tiene mayor efecto a medida que aumenta la edad de los sujetos.

En tareas que implican un gran esfuerzo, el P300 puede ser mayor en niños que en adultos. Los efectos de la probabilidad de la onda P300 en los niños son similares a los efectos producidos en los adultos [24].

Varios estudios han demostrado que la distribución del P300 en el córtex cambia de forma significativa con la edad del sujeto. A medida que los niños crecen, la distribución de la onda P300 pasa de localizarse en la zona parietal del córtex a la zona frontocentral.

Por otra parte, la distribución del P300 asociado a los estímulos visuales y auditivos es diferente en niños y adultos. En niños, se obtienen mejor los potenciales P300 visuales en las zonas frontocentrales que los potenciales P300 auditivos. En adultos, la distribución de los potenciales P300 es similar [24].

La latencia del P300 está relacionada con la capacidad de resolución de problemas, por lo que es posible pensar que también estará relacionada con la inteligencia del sujeto. Debido a esto, se ha estudiado la variación en la forma de la onda P300 dependiendo de la inteligencia o personalidad del sujeto. Tomando como definición de inteligencia la capacidad de un sujeto para resolver problemas simples, se ha demostrado que cuanto mayor es ésta, menor es la latencia y la amplitud del potencial evocado P300 [24].

En cuanto a la personalidad, la extroversión, el neuroticismo y el psicoticismo modifican la forma de la onda P300. Se ha demostrado que cuanto más extrovertido es un sujeto, menor es su RT. Esto puede ser debido a la impulsividad que caracteriza a los sujetos extrovertidos. Además, éstos suelen tener menor amplitud del potencial evocado P300, ya que los sujetos extrovertidos tienen mayor dificultad para prestar atención durante un mayor periodo de tiempo. También se ha demostrado que el neuroticismo, o inestabilidad emocional, está correlado negativamente con la latencia del P300, es decir, cuanto más inestable emocionalmente es un sujeto, menor es la latencia del P300. Finalmente, el psicoticismo, caracterizado por la agresividad y antipatía, está correlado de forma

negativa con la amplitud del P300, es decir, cuanto más psicótico es un sujeto, menor es la amplitud de su P300 [24].

Por último, cabe destacar el efecto producido en el potencial evocado P300 de personas con oído absoluto. Las personas que poseen oído absoluto son capaces de identificar o producir el tono de un sonido sin necesidad de compararlo con un sonido de referencia. Esta capacidad está relacionada con la memoria auditiva. Se ha demostrado que los potenciales P300 auditivos generados mediante un paradigma *odd-ball* en sujetos con oído absoluto poseen una pequeña amplitud, ya que los sujetos con oído absoluto no necesitan prestar atención para distinguir un estímulo auditivo infrecuente.

En la Tabla 3-4, se muestra un resumen de las diferencias individuales entre sujetos que modifican la forma del potencial evocado P300 [24].

2. Procesado de potenciales evocados P300

Una vez que se han adquirido los potenciales evocados P300 y se han eliminado los posibles artefactos que pudieran estar contaminándolos, es necesario realizar un procesado de la señal para identificar dichos potenciales P300. Como ya hemos visto en el capítulo anterior, esto se va a realizar en dos etapas.

2.1. Extracción de características

En los sistemas BCI que utilizan como señal de control los potenciales P300, la etapa de extracción de características suele realizarse de forma sencilla. Como ya se expuso en el capítulo 2, los métodos de extracción de características más utilizados son los que trabajan en el dominio tiempo-frecuencia y aquellos que trabajan en el dominio espacial. Además, habitualmente se utiliza un método de cada tipo, independientemente del orden de aplicación.

La eliminación de artefactos y la extracción de características son dos etapas que no están completamente diferenciadas en este tipo de sistemas. De esta forma, es posible aplicar un mismo método para ambas etapas. En esos casos, los métodos de eliminación de artefactos pueden considerarse como técnicas de extracción de características debido a que su salida se introduce como entrada del algoritmo de traducción.

En los siguientes apartados, se describen algunos de los métodos de extracción de características más utilizados en sistemas BCI, cuya señal de control son los potenciales evocados P300. En el caso los paradigmas desarrollados, se va a utilizar como método de extracción de características en el dominio espacial un filtrado CAR y, después, se va a realizar un submuestreo de la señal, que servirá de entrada al clasificador. Sin embargo, para visualizar los potenciales P300 de los sujetos, se utilizará el promediado sincronizado.

a) Dominio espacial

El objetivo de utilizar un método de extracción de características en el dominio espacial es disminuir el difuminado espacial. El difuminado espacial es un efecto producido por la distancia que existe entre los electrodos y las fuentes que generan la señal dentro del cerebro, debido a la no homogeneidad de los tejidos que se encuentran entre ellos.

TABLA 3-4. Diferencias individuales entre sujetos que modifican la forma del potencial evocado P300.

Diferencias Individuales entre Sujetos				
		Amplitud del P300	**Latencia del P300**	**Otros efectos**
Edad	**Adultos**	Disminuye	Aumenta	Disminuye el RT del sujeto
	Niños	Aumenta (hasta los 13 años)	Disminuye	
↑ Inteligencia		Disminuye	Disminuye	
Personalidad	**↑ Extroversión**	Disminuye		Disminuye el RT del sujeto
	↑ Neuroticismo		Disminuye	
	↑ Psicoticismo	Disminuye		

Para lograr este objetivo, dentro de los métodos en el dominio espacial, los más utilizados son el filtro Laplaciano, el análisis de componentes independientes (ICA) y los métodos de referencia de media común (CAR) [26].

a.1) Filtro Laplaciano

El filtro Laplaciano calcula, para cada electrodo, la segunda derivada de la distribución instantánea del voltaje espacial y, de esta forma, enfatiza la actividad que se genera en un radio concreto alrededor de cada electrodo. Por lo tanto, se trata de un filtro espacial paso-alto que acentúa la actividad localizada y reduce la actividad más difusa.

Para hallar el valor del Laplaciano en cada electrodo, se calcula la combinación entre el valor en cada electrodo y los valores de un conjunto de electrodos circundantes. La distancia desde el electrodo a los electrodos adyacentes determina las características del filtro Laplaciano. Cuanto menor es la distancia, más sensible es el Laplaciano a los potenciales con altas frecuencias espaciales y menos sensible es a los potenciales con bajas frecuencias espaciales. El Laplaciano funcionará mejor con señales de control muy localizadas en el córtex [26].

Para obtener el Laplaciano se utiliza un método de diferencia finita, que aproxima la segunda derivada de la distribución espacial Gaussiana en dos dimensiones del cuero cabelludo en un electrodo, restando la actividad media de los electrodos circundantes al canal de interés. La fórmula utilizada es la siguiente [26]:

$$v_i^{LAP}(t) = v_i^{ER}(t) - \sum_{j \in Si} w_{i,j}\, v_j^{ER}(t) \qquad (3.1)$$

donde $v_n^{ER}(t)$ es el potencial recogido entre el n-ésimo canal y la referencia, siendo i el canal en el que se aplica el filtrado y j los canales adyacentes. Si se corresponde con el conjunto de canales circundantes y $w_{i,j}$ es el peso en función de la distancia entre el electrodo de interés y los electrodos adyacentes [26].

$$w_{i,j} = \frac{\frac{1}{d_{i,j}}}{\sum_{j \in Si} \frac{1}{d_{i,j}}} \tag{3.2}$$

Por lo tanto, el Laplaciano corto es aquel en el que intervienen solamente los electrodos más cercanos que rodean el electrodo al que se le aplica el filtrado. Sin embargo, en el Laplaciano largo, intervienen electrodos que se encuentran a una cierta distancia del electrodo de interés. En la Figura 3-4, se muestra el conjunto de canales utilizados para calcular el Laplaciano. Ya que el número de canales es 4, el peso de cada canal adyacente será de $w_{i,j} = 0.25$.

a.2) Método de referencia de media común

El método de referencia de media común (*Common Average Reference*, CAR) se calcula como la diferencia entre el valor promedio común de todos los canales y el canal de interés. Si el cuero cabelludo está cubierto por electrodos equiespaciados y el potencial es generado por fuentes puntuales, el CAR da como resultado una distribución de voltaje espacial con media nula.

Una característica destacada del CAR es que enfatiza las componentes que se encuentran más localizadas, por lo que puede utilizarse como un filtro espacial paso-alto. En la Figura 3-4, se muestran los electrodos involucrados en el cálculo.

Para obtener el CAR del canal deseado, se resta al voltaje del canal de interés la media común del voltaje obtenido en el resto de los electrodos. Para realizar dicho cálculo se utiliza la siguiente expresión [26]:

$$v_i^{CAR}(t) = v_i^{ER}(t) - \frac{1}{n}\sum_{j=1}^{n} v_j^{ER}(t) \tag{3.3}$$

donde $v_n^{ER}(t)$ es el potencial recogido entre el i-ésimo canal y la referencia, siendo i el canal en el que se aplica el filtrado; y n es el número total de electrodos utilizados, incluyendo el canal filtrado.

a.3) Análisis de componentes independientes

El análisis de componentes independientes (*Independent Component Analysis*, ICA) es una técnica estadística utilizada como solución para el problema de la separación ciega de fuentes. Este método obtiene las fuentes originales que generan las señales a partir de una mezcla de dichas señales. Por lo tanto, ICA trata de dividir la señal EEG en componentes independientes, suponiendo que la señal es estadísticamente independiente y posee una distribución no gaussiana.

El objetivo de esta técnica es revelar las características de la señal EEG, ocultas por el ruido de fondo. De esta forma, es posible detectar los potenciales P300 a través de una única prueba, llegando a obtenerse una precisión del 76.67% [27].

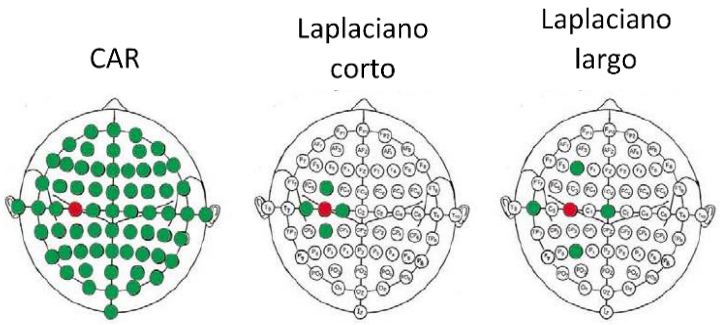


Figura 3-4. Filtrados espaciales para obtener la señal filtrada procedente del canal C3. El electrodo de interés se muestra en color rojo, mientras que, los electrodos adyacentes se muestran en color verde [5].

Un ejemplo del problema de la separación ciega de fuentes se explica a continuación. Dos sujetos (S_1 y S_2) están hablando de forma simultánea en una habitación en la que dos micrófonos (R_1 y R_2) graban su conversación desde ubicaciones diferentes de la habitación. Las señales registradas se pueden expresar como:

$$\begin{aligned} R_1(t) &= a_{11}S_1(t) + a_{12}S_2(t) \\ R_2(t) &= a_{21}S_1(t) + a_{22}S_2(t) \end{aligned} \tag{3.4}$$

Si se conocen los valores de a_{11}, a_{12}, a_{21} y a_{22} se pueden resolver las ecuaciones para obtener S_1 y S_2, y así distinguir las dos conversaciones sin que se produzca solapamiento. Lamentablemente, estos pesos son desconocidos y, por tanto, las ecuaciones solo pueden resolverse bajo la suposición de que S_1 y S_2 son señales independientes y no gaussianas. Este ejemplo se conoce como el problema *cocktail party* y es el que intenta resolver ICA [27].

El análisis de la señal EGG es un problema de tipo *cocktail party* ya que los electrodos registran la señal en distintas localizaciones del cuero cabelludo, produciéndose una mezcla. Por lo tanto, se aplica ICA en la señal EEG para intentar distinguir las fuentes independientes y, concretamente, las que producen los potenciales evocados P300 para facilitar su posterior detección.

De forma análoga, se puede expresar de forma generalizada la expresión (3.4) como:

$$\begin{aligned} x_1(t) &= a_{11}S_1(t) + a_{12}S_2(t) + \cdots \\ x_2(t) &= a_{21}S_1(t) + a_{22}S_2(t) + \cdots \\ &\cdots \end{aligned} \longrightarrow \quad \boldsymbol{x} = \boldsymbol{A} \cdot \boldsymbol{S} \tag{3.5}$$

donde $\boldsymbol{x}$ es la matriz de las señales mezcladas, $\boldsymbol{A}$ es la matriz que contiene los pesos a_{ij} y $\boldsymbol{S}$ es la matriz de las fuentes que se desean distinguir. Se asume que los coeficientes a_{ij} son desconocidos y que permiten que la matriz $\boldsymbol{A}$ sea invertible. Además, se asume, como ya se ha mencionado anteriormente, que las componentes de $\boldsymbol{S}$ son independientes y tienen distribución no gaussiana; que la mezcla de señales es lineal y espontánea y que los datos son estacionarios [27].

Utilizando las suposiciones anteriores, se puede concluir que existe una matriz $\boldsymbol{M}$ con coeficientes m_{ij} que permite obtener las fuentes estimadas $\boldsymbol{S'}$ al multiplicarse con las señales mezcladas, utilizando la siguiente expresión:

$$\boldsymbol{y = M \cdot x = M \cdot [A \cdot S] = S' \longrightarrow S' = M \cdot x} \qquad (3.6)$$

Por lo tanto, la matriz $\boldsymbol{M}$ es la matriz pseudoinversa de $\boldsymbol{A}$. Existen diversos algoritmos ICA para obtener la matriz $\boldsymbol{M}$, entre los que destacan *Informax, JADE y FastICA*, el que mejores resultados presenta [27].

b) Dominio tiempo-frecuencia

Los principales métodos de extracción de características en el dominio temporal son el promediado sincronizado, la detección de picos, el cálculo del área de la señal y el filtrado adaptativo.

Como se ha visto en capítulos anteriores, los métodos temporales destacan por su reducido coste computacional y por su rapidez a la hora de proporcionar retroalimentación al usuario. Sin embargo, para realizar un análisis offline más exhaustivo, normalmente, suelen utilizarse técnicas de análisis espectral. Entre estas técnicas destacan la transformada de Fourier de tiempo corto (*Short Time Fourier Transform*, STFT) y las transformadas *Wavelet.*

b.1) Promediado sincronizado

El promediado sincronizado permite distinguir los potenciales evocados P300 del ruido del EEG basal. Debido a que los espectros de la señal de interés y el ruido se superponen, los filtros lineales (filtro paso-bajo, filtro paso-alto o filtro paso-banda) no permiten separar las dos señales.

Para calcular el promediado sincronizado de una señal, es necesario disponer de varias épocas, obtenidas mediante la aplicación repetida del estímulo; y de una referencia temporal para poder alinearlas [25].

Como se muestra en la expresión (3.7), esta técnica consiste en realizar el promedio de todas las épocas de la señal, estableciendo como referencia el punto en el que se ha generado el estímulo que produce el potencial evocado P300, es decir, cuando n es 0.

$$Prom\,(n) = \sum_{k=1}^{M} y_k(n) = \sum_{k=1}^{M} x_k(n) + \sum_{k=1}^{M} \eta_k(n)\,; \qquad n = 1, 2, \dots, N \qquad (3.7)$$

donde $y_k(n) = x_k(n) + \eta_k(n)$, siendo $y_k(n)$ una realización de la señal, $x_k(n)$ la señal original sin ruido y $\eta_k(n)$ el ruido de la k-ésima época. k representa el índice del promediado, n es el índice de tiempo discreto, M representa el número de épocas y N es el número de muestras de la señal [25]. Este promediado enfatiza las componentes que se repiten en las distintas realizaciones de $x_k(n)$ sobre el promediado del ruido, convirtiéndose en un método muy adecuado para la visualización de los potenciales evocados P300.

Cabe destacar la aplicación de este método en la selección de caracteres. Cuando el sujeto selecciona una celda, tanto la fila como la columna que contienen la celda deseada generan un potencial P300 en una época de la señal. Debido a que no es posible visualizar la presencia de un potencial P300 en una única época, es necesario aplicar un promediado sincronizado de todas las filas y columnas de la matriz. En el caso de que la matriz de selección esté formada por 6 filas y 6 columnas, se realizará un promedio de 36 épocas. A continuación, se aplica la detección de picos o

el cálculo del área para detectar los potenciales P300 de forma automática y obtener la celda que el sujeto ha seleccionado [28].

b.2) Detección de picos y cálculo del área

La detección de picos o *peak picking* permite detectar los potenciales evocados P300 monitorizando la amplitud del promediado sincronizado de la señal. Esta técnica consiste en obtener la diferencia entre el punto más bajo anterior a la ventana establecida y el punto más alto dentro de la ventana. Posteriormente, se compara con el valor obtenido para un segmento basal. Aunque la ventana utilizada puede tener diversos tamaños, típicamente, el tamaño suele estar comprendido entre cientos de milisegundos antes del estímulo hasta 500 ms después de este, para abarcar el potencial evocado P300.

El cálculo del área comprendida bajo la señal promediada es un método que, también, nos permite detectar la presencia de potenciales evocados P300. Para obtener el valor del área se suman todos los puntos de la ventana establecida. Cuando el área comprendida bajo la señal es muy superior que el obtenido a partir de un segmento basal, se habrá detectado un P300 [18].

b.3) Filtrado adaptativo

El filtro adaptativo, como su nombre indica, se trata de un filtro que se adapta continuamente a las características de la señal, cambiando el vector de pesos con el tiempo, como se muestra en la Figura 3-5.

Se parte de una señal de entrada $x(n)$, resultado de la suma de la señal de interés $v(n)$ y el ruido primario $m(n)$, asumiendo que ambas señales están incorreladas, como se expresa en la ecuación (3.8) [25].

$$x(n) = v(n) + m(n) \tag{3.8}$$

El filtro adaptativo filtra la entrada de referencia $r(n)$, fuertemente correlada con el ruido primario $m(n)$, para obtener $y(n)$. Esta señal debe ser lo más parecida posible al ruido primario. A continuación, a la señal de entrada $x(n)$ se le resta $y(n)$, obteniendo la estimación de la señal de interés $e(n)$. Este proceso se recoge en la ecuación (3.9).

$$e(n) = \hat{v}(n) = x(n) - y(n) \tag{3.9}$$

El algoritmo de mínimos cuadrados promediados (*Least Mean Squares*, LMS) es el método utilizado para ajustar los coeficientes del filtro adaptativo. Se encarga de ajustar el vector de pesos para minimizar el error cuadrático medio (*Mean Square Error*, MSE), calculando un nuevo vector de pesos en base al vector de pesos actual más una correlación proporcional al gradiente negativo MSE. La ecuación (3.9) recoge la expresión utilizada para actualizar los pesos en cada momento. Se trata de una particularización de la regla de Widrow-Hoff [25].

$$e^2(n) = x^2(n) - 2x(n)\boldsymbol{r^T}(n)\,\boldsymbol{w}(n) + \boldsymbol{w}^T(n)\boldsymbol{r}(n)\boldsymbol{r^T}(n)\,\boldsymbol{w}(n) \tag{3.10}$$

El algoritmo que permite ajustar el vector de pesos es fácil de implementar y no requiere realizar operaciones de derivación, potencias o promedios, aunque se base en minimizar el MSE. Otro

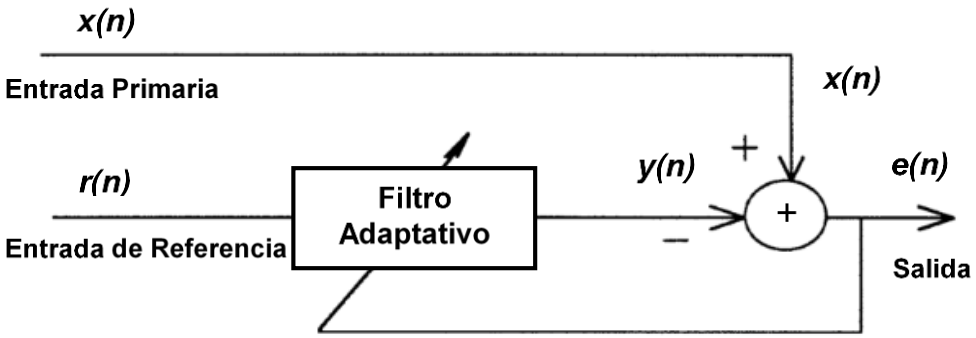


Figura 3-5. Esquema de filtrado adaptativo.

algoritmo que puede utilizarse para este fin es el algoritmo de mínimos cuadrados recursivos (*Recursive Least Square*, RLS), óptimo cuando las señales varían mucho en oscilaciones rápidas de tiempo.

El filtrado adaptativo no es un método muy utilizado en aplicaciones que involucran potenciales evocados P300 debido a su estacionariedad. Sin embargo, se trata de una técnica apropiada para la eliminación de artefactos. En la señal EEG, este método se encarga de eliminar artefactos sin asumir estacionariedad en la señal, aunque requiere una señal fuertemente correlada con el ruido que se pretende eliminar [25].

b.4) Transformada de Fourier de Tiempo Corto

La transformada de Fourier de tiempo corto (*Short-Time Fourier Transform*, STFT) es una técnica que permite dividir la señal en segmentos, lo suficientemente pequeños para que sean estacionarios, y calcula su transformada de Fourier [29]. Debido a que la STFT no calcula la trasformada de Fourier continua sino una versión enventanada, aparece distorsión debida a la ventana empleada. Esto produce un compromiso entre la resolución en tiempo y la resolución en frecuencia.

La expresión matemática de la STFT se formula en la ecuación (3.8).

$$X_w[n,k] = \sum_{m=0}^{N-1} x[m] \cdot w^*[m-n] \cdot \exp\left(-\frac{j2\pi km}{N}\right) \qquad (3.11)$$
$$k = 0, \dots, L-1, \qquad n = 0, \dots, N_T - 1$$

donde w$[n]$ representa la función ventana, $x[m]$ es el segmento de la señal al que se le aplica la STFT y $N_T = N/L$ es el número de segmentos no solapados en los que se ha dividido la señal [12].

La principal desventaja de esta técnica es que, una vez fijada la ventana, el espectrograma presenta una resolución fija en el plano tiempo-frecuencia [29]. En la Figura 3-6 se representa el plano tiempo-frecuencia para la STFT. Las señales biomédicas suelen presentar, de forma simultánea, oscilaciones rápidas en intervalos cortos o variaciones lentas en intervalos largos, lo que requiere una resolución variable. Debido a esto, las transformadas *Wavelet* son más adecuadas para el análisis de señales biomédicas.

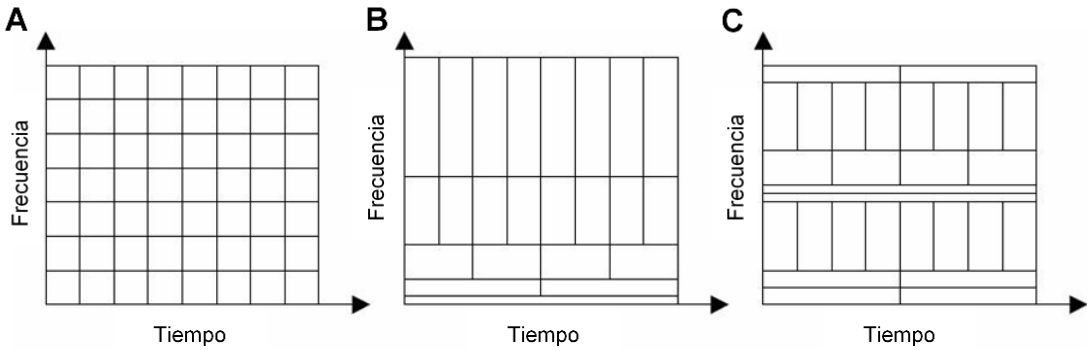


Figura 3-6. Plano tiempo-frecuencia para STFT en (A), DWT en (B) y WP en (C).

b.5) Transformadas Wavelet

Las transformadas Wavelet surgen como solución alternativa a la STFT, debido a las ventajas que presentan al aplicarlas a las señales EEG. Las técnicas *Wavelet* expuestas a continuación se basan en la traslación y el escalado de una función *Wavelet* madre, cuya expresión se recoge en la ecuación (3.9). La traslación y el escalado permiten obtener buena resolución temporal para oscilaciones rápidas y buena resolución frecuencial para oscilaciones lentas, es decir, la resolución es variable en el plano tiempo-frecuencia.

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}} \cdot \psi\left(\frac{t - \tau}{s}\right) \tag{3.12}$$

donde $\psi_{\tau,s}\ (t)$ es la función *Wavelet* madre, τ es el parámetro de traslación y s es el parámetro de escalado.

Existe una gran variedad de *Wavelets* madre, entre las que destacan la *Wavelet* de Morlet, de Meyer, de Daubechies, de Haar y de Symlets. Se ha demostrado que la elección de la *Wavelet* madre afecta directamente al rendimiento de la aplicación

La transformada *Wavelet* continua (*Continuous Wavelet Transform*, CWT) se define como la correlación entre la señal EEG y la función *wavelet* madre escalada y desplazada, cuya expresión se recoge en la ecuación (3.10).

$$w(s,\tau) = \int_{-\infty}^{\infty} x(t)\ \cdot \frac{1}{\sqrt{s}} \cdot \psi\left(\frac{t - \tau}{s}\right) dt \tag{3.13}$$

A partir de la CWT se obtiene una representación de la señal en el plano tiempo-frecuencia o tiempo-escala, cuyo resultado es una buena resolución frecuencial a bajas frecuencias y una buena resolución temporal a altas frecuencias [29]. Este fenómeno puede verse representado en la Figura 3-6.

Sin embargo, a pesar de sus ventajas, la CWT es un método demasiado redundante. Debido a esto, surge la transformada *Wavelet* discreta (*Discrete Wavelet Transform*, DWT). La DWT muestrea los valores de los parámetros de traslación (τ) y escalado (s) de forma diádica, como se expone en la ecuación (3.14). Debido a que la CWT es demasiado redundante, la DWT no pierde mucha

información efectiva sobre la señal. Las principales ventajas de la DWT son la reducción en el coste computacional y la posibilidad de realizar un análisis multiresolución.

$$s = 2^{-j}\,, \quad \tau = k \cdot 2^{-j} \qquad \psi_{j,k}(t) = 2^{j/2} \cdot \psi\left(2^{j} \cdot t - k\right) \tag{3.14}$$

donde $2^{j/2}$ es el factor de normalización, que garantiza la condición de ortonormalidad.

El análisis multiresolución consiste en descomponer la señal en una parte de aproximación, con coeficientes $c_{j,k}$, y una parte de detalle, con coeficientes $d_{j,k}$. La parte de aproximación refleja las características principales de la señal y la parte de detalle contiene las fluctuaciones más rápidas, por lo que se pueden considerar un filtro paso-bajo y un filtro paso-alto, respectivamente. A su vez, la parte de aproximación se subdivide en una parte de aproximación y una parte de detalle, y así sucesivamente hasta alcanzar el número de escalas de descomposición deseado.

Los resultados más precisos se obtienen utilizando los paquetes *Wavelet* (*Wavelet Packets*, WP). Esta técnica se basa en el análisis multiresolución que se lleva a cabo en la DWT. La diferencia con respecto al análisis multiresolución se encuentra en la descomposición de la parte de detalle. Los WP se descomponen en una parte de aproximación y una parte de detalle que, a su vez, se descomponen en una parte de aproximación y una parte de detalle, y así sucesivamente. De esta forma, se obtiene un árbol de descomposición más extenso que permite realizar cualquier cambio en la resolución del plano tiempo-frecuencia, aumentando el coste computacional.

2.2. Traducción de características

Una vez se han obtenido los potenciales P300 de la señal EEG, será necesario clasificar la presencia o ausencia de un potencial P300. Para ello, se van a utilizar los métodos de traducción de características o clasificadores. Además de realizar la clasificación, esta etapa se encarga de asegurar el buen funcionamiento del clasificador y de considerar la adaptabilidad al usuario [5].

Como ya se explicó en el capítulo 2, los algoritmos de clasificación se pueden dividir en dos grandes grupos, clasificadores lineales y clasificadores no lineales. En los siguientes apartados, se describen algunos de los métodos de traducción de características más utilizados en la clasificación de potenciales evocados P300. Entre los clasificadores más utilizados destacan el discriminante lineal de Fisher, el análisis discriminante lineal paso-a-paso y las máquinas de soporte vectorial.

Determinar la presencia o ausencia de un potencial evocado P300 a partir de las características del EEG se puede considerar un problema de clasificación binario con una función discriminante, cuyo hiperplano de decisión está definido por la siguiente expresión [30].

$$\boldsymbol{w} \cdot \ f(\boldsymbol{x}) + b = 0 \tag{3.15}$$

donde $\boldsymbol{x}$ es el vector de características, $f(\cdot)$ es la función de transformación, $\boldsymbol{w}$ es un vector de pesos y b es el sesgo. Para los métodos no lineales, la función $f(\cdot)$ puede representar una transformación que asigna las características a un espacio de mayor dimensión, para crear un conjunto de datos separado linealmente. Para los métodos lineales, la función $f(\cdot)$ es una transformación simple que se puede expresar como: $f(\boldsymbol{x}) = a\boldsymbol{x} + c$.

El vector de pesos $\boldsymbol{w}$ asigna una etiqueta de clase a cada época o estímulo, siendo +1 la etiqueta para el estímulo objetivo y -1 para el estímulo no objetivo. De esta forma, el diseño propuesto

permite distinguir la respuesta con la mayor distancia positiva del hiperplano de separación entrenado. En la aplicación de selección de caracteres, se obtiene la mayor distancia positiva de todas las filas y columnas. A partir de ello, se determina la celda seleccionada por el sujeto [30].

Los métodos que se describen a continuación son distintos enfoques utilizados para calcular el vector de pesos, en el caso de ser métodos lineales; y, el vector de pesos y la función de transformación, en el caso de ser métodos no lineales.

a) Discriminante lineal de Fisher

El discriminante lineal de Fisher (*Fisher's Linear Discriminat*, FLD) es el método de referencia utilizado para determinar el hiperplano de separación óptimo entre dos clases. Este método es sencillo de calcular y, además, proporciona una clasificación robusta que es óptima cuando las dos clases son gaussianas con igual covarianza.

Para realizar clasificaciones binarias como la que se expone aquí, el FLD y la solución de regresión obtenida mediante LMS son equivalentes. A partir de esto, se busca una dirección del espacio geométrico resultante de proyectar los datos, tal que el MSE de los datos proyectados sea mínimo [30].

El vector de pesos para el caso propuesto se puede expresar a través de la siguiente ecuación:

$$\widehat{\boldsymbol{w}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\,\boldsymbol{X}\right)^{-\mathbf{1}} \cdot \boldsymbol{X}^{\boldsymbol{T}} \cdot \boldsymbol{Y} \tag{3.16}$$

donde $\boldsymbol{X}$ es la matriz formada por los vectores de características e $\boldsymbol{Y}$ es el vector de etiquetas de cada clase [30].

Los resultados obtenidos con este método reflejan la buena calidad del estimador. La principal ventaja del FLD es que utiliza la covarianza entre características.

b) Análisis discriminante lineal paso-a-paso

El análisis discriminante lineal paso-a-paso o SWLDA es uno de los métodos de clasificación más utilizados en sistemas BCI basados en potenciales evocados P300 y se trata del método empleado en este trabajo.

Los métodos de análisis discriminante lineal (*Linear Discriminant Analysis*, LDA) buscan una combinación lineal de características que sea capaz de separar dos o más clases de eventos, utilizando estadísticas de segundo orden para obtener el vector de pesos $\boldsymbol{w}$ [31]. La calidad del clasificador resultante, por lo tanto, depende de la calidad de la covarianza estimada.

El algoritmo SWLDA surge debido a que, en la práctica, es complicado obtener la cantidad de datos necesarios para establecer un buen clasificador LDA. Este método trata de mejorar la covarianza estimada minimizando el número de pesos mayores que cero. Partiendo de un subconjunto de dimensiones de datos, el algoritmo SWLDA determina para cada dimensión, cuánto afecta a la calidad de la estimación el incluir dicha dimensión o eliminarla en el cálculo y, posteriormente, modifica el conjunto de datos en consecuencia.

Este análisis es una particularización del método de regresión paso-a-paso, que combina dos métodos básicos de selección de características, el análisis paso-a-paso hacia delante y el análisis paso-a-paso hacia atrás [32].

El análisis paso-a-paso hacia delante (*forward stepwise regression*) parte de un conjunto sin características y evalúa la significancia de cada una, añadiéndola al conjunto si supera un umbral o criterio de entrada y descartándola en caso contrario.

El análisis paso-a-paso hacia atrás (*backward stepwise regression*) realiza el proceso inverso. Partiendo de un conjunto con todas las características posibles, evalúa cada una de ellas, analizando si deberían ser incluidas en el conjunto o no. Para ello, se fija un criterio de salida y se recorre el conjunto de características, comprobando si éstas cumplen el criterio. Si se cumple el criterio, se descarta, reduciendo el tamaño del conjunto final.

El algoritmo SWLDA combina estos dos métodos como se explica a continuación. Partiendo de un conjunto sin características, el algoritmo SWLDA realiza un análisis paso-a-paso hacia delante para añadir las características más significativas, es decir, aquellas características que cumplan con el criterio de entrada fijado. Cada vez que se introduce una nueva característica en el conjunto de datos, se calculan de nuevo los *p-valores* y se analiza si alguna de las características incluidas en el conjunto de datos cumple con el criterio de salida fijado, para lo que se utiliza el análisis paso-a-paso hacia atrás. Si alguna característica cumple el criterio, se descarta y se vuelve a realizar un análisis paso-a-paso hacia delante. Este proceso se repite hasta que el conjunto de datos alcance el número máximo de características o hasta que no existan características que cumplan con los criterios [30] [32].

Numerosos estudios han demostrado la utilidad de este algoritmo de entrenamiento en aplicaciones como la selección de caracteres [30], [33]. En este tipo de aplicaciones, se utilizan como criterios de entrada y salida los que se muestran en la ecuación (3.17).

$$\begin{aligned} &Criterio\ de\ entrada: && p - valor\ < 0.1 \\ &Criterio\ de\ salida: && p - valor\ > 0.15 \end{aligned} \tag{3.17}$$

De esta forma, se añaden al conjunto de datos (función discriminante) las características cuyo *p-valor* sea inferior a 0.1 y se eliminan aquellas cuyo *p-valor* sea superior a 0.15 [30].

c) Máquinas de soporte vectorial

Las máquinas de soporte vectorial (*Support Vector Machines*, SVM) son un conjunto de algoritmos de aprendizaje supervisado, diseñadas para determinar un hiperplano o conjunto de hiperplanos en un conjunto con una dimensionalidad alta. Suelen utilizarse en problemas de clasificación o regresiones no lineales.

El hiperplano buscado es aquel que maximice la distancia (margen) entre los puntos más cercanos a dicho hiperplano, también conocido como clasificador de margen máximo [30]. En la Figura 3-7, se muestra un ejemplo de un hiperplano de separación. La forma más simple de establecer la frontera de decisión y separar las clases es utilizando una función lineal. No siempre es posible utilizar una función lineal, por lo que es necesario proyectar la información en un espacio de características de mayor dimensión utilizando funciones Kernel [34].

Dado el vector de características x y el vector de etiquetas de clases d (con longitud N), cuyos valores cumplen que $d_i \in \{\pm 1\}$, la ecuación (3.15) se puede reformular en una nueva expresión, recogida en la ecuación (3.18) [30].

$$d_i \left(\boldsymbol{w^T} \cdot x_i + b\right) \geq 1, \qquad para\ i \in [1, N] \tag{3.18}$$

El hiperplano óptimo de separación deberá maximizar la ecuación (3.18), cuyo margen de separación para la clasificación binaria es equivalente a minimizar la forma euclídea del vector de pesos óptimo $\boldsymbol{w}_0$, como expresa la ecuación (3.18) [34].

$$\rho = \frac{2}{\|\boldsymbol{w}_0\|} \tag{3.19}$$

Los vectores de soporte son los puntos que se encuentran más próximos al hiperplano de separación y cuyo margen de separación ρ_0 es máximo.

Una vez conocida la expresión del margen de separación, es necesario maximizar la ecuación (3.18), para lo que se va a utilizar el método de los multiplicadores de Lagrange [25].

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \boldsymbol{x}_i^T \boldsymbol{x}_j \tag{3.20}$$

A continuación, se obtienen los multiplicadores de Lagrange óptimos $\alpha[\alpha_0, \dots, \alpha_i]$ que maximizan la ecuación (3.20), sujeta a las siguientes restricciones.

$$\sum_{i=1}^{N} \alpha_i d_i = 0\,, \qquad con\ \alpha_i \geq 0\ para\ i = 1, \dots, N \tag{3.21}$$

En las ecuaciones (3.22) y (3.23), se expresan el hiperplano óptimo, el vector de pesos óptimo $\boldsymbol{w_0}$ y el sesgo óptimo b_0 para una separación lineal [34].

$$\boldsymbol{w_0} = \sum_{i=1}^{N} \alpha_{0,i} d_i x_i \tag{3.22}$$

$$b_0 = 1 - \boldsymbol{w_0^T} x^{(s)}, \qquad con\ d^{(s)} = 1 \tag{3.23}$$

Desafortunadamente, cuando las clases no pueden separarse linealmente no se puede obtener el hiperplano utilizando las ecuaciones (3.22) y (3.23) sin que se produzcan errores de clasificación. Para medir la diferencia entre el patrón considerado y el hiperplano óptimo, se utiliza el parámetro ξ_i [34]. Este parámetro cuantifica el error, a partir del cual se pueden dar dos casos que se ilustran en la Figura 3-7.

1. Uno de los datos se localiza en la zona de decisión del lado correcto del hiperplano de separación. En este caso, el parámetro ξ_i tendrá un valor entre 0 y 1 ($0 \leq \xi_i \leq 1$).

2. Uno de los datos se localiza en el lado incorrecto de la frontera de decisión establecida por el hiperplano de separación. En este caso, el parámetro ξ_i tendrá un valor superior a 1 ($\xi_i > 1$).

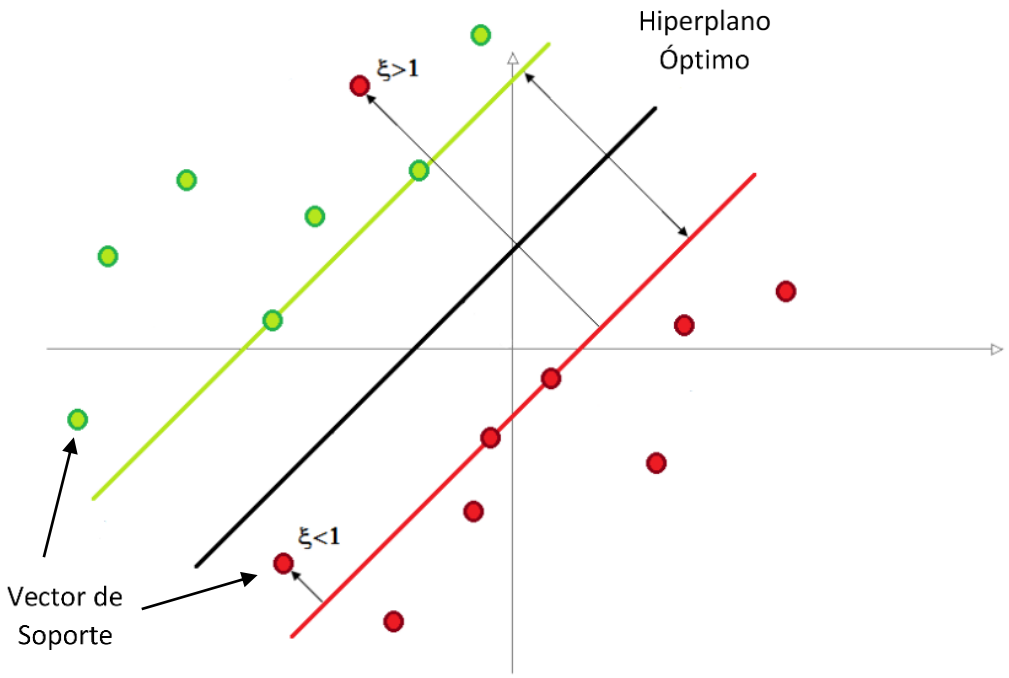


Figura 3-7. Hiperplano óptimo de separación y posibles errores de clasificación SVM.

Para obtener el hiperplano óptimo cuando las variables no son linealmente separables, se aumenta la dimensión del conjunto de datos y se minimiza la ecuación (3.24), donde C es un parámetro escalar determinado de forma experimental.

$$\phi(\boldsymbol{w},\xi) = \frac{1}{2}\,\boldsymbol{w^T w} + C\sum_{i=1}^{N}\xi_i \qquad (3.24)$$

La transformación no lineal se realiza por medio de funciones Kernel gaussianas, como se expresa en la ecuación (3.25). Se utilizan funciones Kernel gaussianas debido a sus propiedades de aproximación universal [30].

$$Score_{SVM} = \sum_{i=1}^{N}\alpha_i y_i \cdot K(x_i, x_j), \quad con\ K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} \qquad (3.25)$$

donde los parámetros C y σ^2 se obtiene de forma experimental. Estos parámetros varían entre sujetos, aunque se ha demostrado que los valores de estos parámetros con los que se obtiene el mejor rendimiento son $C = 10$ y $\sigma^2 = 10^3$ [30].

Este clasificador se ha aplicado satisfactoriamente en numerosas aplicaciones BCI con potenciales evocados P300 [35].

CAPÍTULO IV

DESARROLLO DE PARADIGMAS ODD-BALL CON OPENVIBE

1. Objetivos de la aplicación

El principal objetivo de las aplicaciones desarrolladas es proporcionar una herramienta que aumente la independencia y la capacidad de comunicación de personas que sufren una grave discapacidad o enfermedades neurodegenerativas. La finalidad es permitir a los usuarios escribir mediante dos aplicaciones de selección de caracteres, utilizando únicamente las ondas cerebrales, sin que intervengan los nervios y músculos periféricos.

El método seleccionado para registrar la actividad cerebral es el EEG, debido a que se trata de un método no invasivo, portátil, de bajo coste y fácil uso. En el capítulo 2, se realizó un estudio de los posibles paradigmas *odd-ball* a implementar en las aplicaciones de selección de caracteres utilizando señales EEG. En las aplicaciones desarrolladas en este trabajo se ha utilizado como señal de control los potenciales evocados P300, ya que se trata de una señal que no requiere entrenamiento y se procesan de forma rápida y sencilla.

Se pretende que las aplicaciones a desarrollar sean aplicaciones en las que el usuario pueda desenvolverse de forma independiente, sin necesidad de supervisión. A continuación, se detallan los objetivos específicos de cada aplicación de selección de caracteres desarrollada.

- El objetivo principal del desarrollo del paradigma RCP es poder comparar los resultados obtenidos en las aplicaciones desarrolladas utilizando los paradigmas a implementar seleccionados en las mismas condiciones.
- El objetivo principal del paradigma CBP es aumentar la precisión y la tasa de transferencia. Además, pretende reducir la tasa de error debida a las distracciones adyacentes y el problema del *doble-flash*.
- El objetivo principal del paradigma HSRD es ayudar a los usuarios a enfocar su atención en el estímulo visual. Además, este paradigma pretende aumentar la precisión y la tasa de transferencia, y reducir las distracciones producidas por caracteres adyacentes.
- El objetivo principal del paradigma HSRDCBP es ayudar a los usuarios a enfocar su atención en el estímulo visual, evitando las distracciones adyacentes. Además, con este nuevo paradigma unificado se pretende aumentar la precisión y reducir el problema del *doble-flash.*

Con el objetivo de determinar las características de los paradigmas desarrollados, así como su efectividad y rendimiento, se van a comparar las precisiones obtenidas para cada número de secuencias (con un máximo de 10) y la respuesta evocada por los estímulos. Para ello, se realizarán cuatro sesiones de evaluación en las que participará un conjunto de 5 sujetos de control y se realizará un análisis cualitativo adicional para recoger las sugerencias y opiniones de los usuarios.

2. Partes de la aplicación

En la Figura 4-1, se muestran las partes que componen las aplicaciones de selección de caracteres que se han desarrollado, la adquisición de la señal y la aplicación de selección de caracteres desarrollada mediante la plataforma *OpenViBE*. Ambas aplicaciones están compuestas por las mismas partes.

La primera parte del desarrollo del paradigma consiste en obtener la señal EEG. La señal de cada usuario se registra a través de electrodos localizados sobre el cuero cabelludo y un amplificador de la empresa *G·Tec*. En la segunda parte del desarrollo, la plataforma *OpenViBE* se encarga de procesar la señal EEG, extraer las características de los potenciales evocados P300 y presentar al usuario la selección realizada. Para procesar la señal EEG, la plataforma *OpenViBE*, en primer lugar, aplica un filtro paso-banda de Butterworth con frecuencias de corte de 1 y 20 Hz. Este filtrado se realiza para eliminar la componente continua y el ruido de alta frecuencia que contiene la señal EEG. En segundo lugar, se realiza un submuestreo de las épocas para extraer las características de la señal. A continuación, *OpenViBE* traduce las características utilizando un algoritmo LDA y clasifica los potenciales para determinar qué carácter deseaba seleccionar el usuario [36].

3. Funcionamiento de la aplicación

Como se ha explicado en el apartado anterior, las aplicaciones constan de dos partes principales que se comunican entre sí para que el usuario pueda escribir utilizando únicamente las ondas cerebrales.

La interfaz gráfica de las diferentes aplicaciones está formada por una matriz de 6 filas y 6 columnas, con un total de 36 caracteres alfanuméricos. Esta matriz se encarga seleccionar el carácter que el usuario desea escribir, utilizando el paradigma *odd-ball*. A continuación, se explica el funcionamiento específico de cada una de las aplicaciones.

3.1. Row-Column Paradigm

En la aplicación que utiliza el paradigma RCP, las filas y columnas que formas la matriz de caracteres se iluminan de forma aleatoria. Como ya se ha explicado, la iluminación de las filas y columnas se realiza utilizando el paradigma *odd-ball*. En la Figura 4-2, se muestra la interfaz de la aplicación cuando se está iluminando una columna [18].

Al percibir una iluminación del carácter que se desea seleccionar, el cerebro genera un potencial evocado P300 en la señal EEG del usuario. Después, estos potenciales se analizan y se detectan para determinar cuál es el carácter que el usuario ha seleccionado.

3.2. Checkerboard Paradigm

En la aplicación desarrollada utilizando el paradigma CBP, los caracteres que conforman la matriz se superponen virtualmente con un tablero de ajedrez. A partir de esto, los caracteres pertenecientes a las celdas blancas se aíslan en una matriz blanca de tamaño 6x3 y los de las celdas negras en una matriz negra de tamaño 6x3. Antes de que comience la secuencia de iluminación, los caracteres se distribuyen de forma aleatoria en las matrices blanca y negra [19].

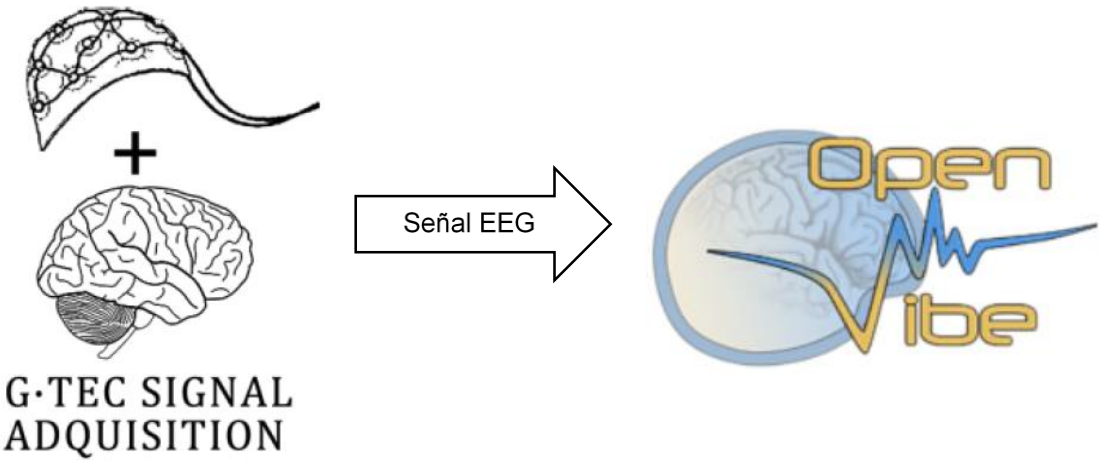


Figura 4-1. Estructura principal de las aplicaciones de selección de caracteres desarrolladas.

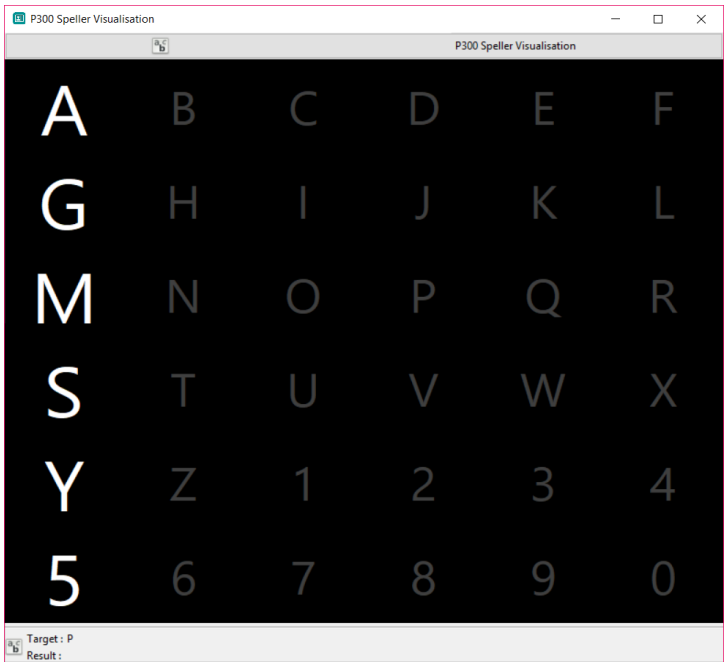


Figura 4-2. Interfaz gráfica de la aplicación desarrollada mediante el paradigma tradicional.

En esta aplicación, la secuencia de iluminaciones comienza con la iluminación de las filas virtuales de la matriz blanca, seguidas por las filas virtuales de la matriz negra. Después, pasan a iluminarse las columnas virtuales de la matriz blanca, seguidas por las columnas virtuales de la matriz negra. En la Figura 4-3, se muestra la interfaz de la aplicación cuando se está iluminando una fila.

Como se describe en el apartado anterior, al percibir una iluminación del carácter que se desea seleccionar, el cerebro genera un potencial evocado P300 en la señal EEG del usuario. Después, estos potenciales se analizan y se detectan para determinar cuál es el carácter que el usuario ha seleccionado [19].

Figura 4-3. Interfaz gráfica de la aplicación desarrollada mediante el paradigma CBP.

3.3. Honeycomb-Shaped Red Dots Paradigm

En el paradigma HSRD, se sustituye la iluminación de un carácter de la matriz por la aparición de un panal de color verde, en cuyo interior se encuentran distribuidos puntos de color rojo. El número de puntos varía de 1 a 3 de forma aleatoria, lo que permite que el usuario pueda contarlos de una forma rápida y sencilla. El patrón de puntos generado es el mismo para todos los caracteres en cada iluminación. En la Figura 4-4, se muestra la interfaz de la aplicación [23].

Como se describe en apartados anteriores, al percibir una iluminación del carácter que se desea seleccionar, el cerebro genera un potencial evocado P300 en la señal EEG del usuario. Después, estos potenciales se analizan y se detectan para determinar cuál es el carácter que el usuario ha seleccionado.

3.4. Honeycomb-Shaped Red Dots – Checkerboard Paradigm

En el paradigma unificado HSRDCBP, al igual que ocurre en el paradigma HSRD, se sustituye la iluminación de un carácter de la matriz por la aparición de un panal de color verde, en cuyo interior se encuentran distribuidos puntos de color rojo. El número de puntos varía de 1 a 3 de forma aleatoria, lo que permite que el usuario pueda contarlos de una forma rápida y sencilla. El patrón de puntos generado es el mismo para todos los caracteres en cada iluminación. En la Figura 4-5, se muestra la interfaz de la aplicación [23]. La iluminación de los caracteres se realiza de forma aleatoria, como ocurre en el paradigma CBP.

Como se describe en apartados anteriores, al percibir una iluminación del carácter que se desea seleccionar, el cerebro genera un potencial evocado P300 en la señal EEG del usuario. Después, estos potenciales se analizan y se detectan para determinar cuál es el carácter que el usuario ha seleccionado.

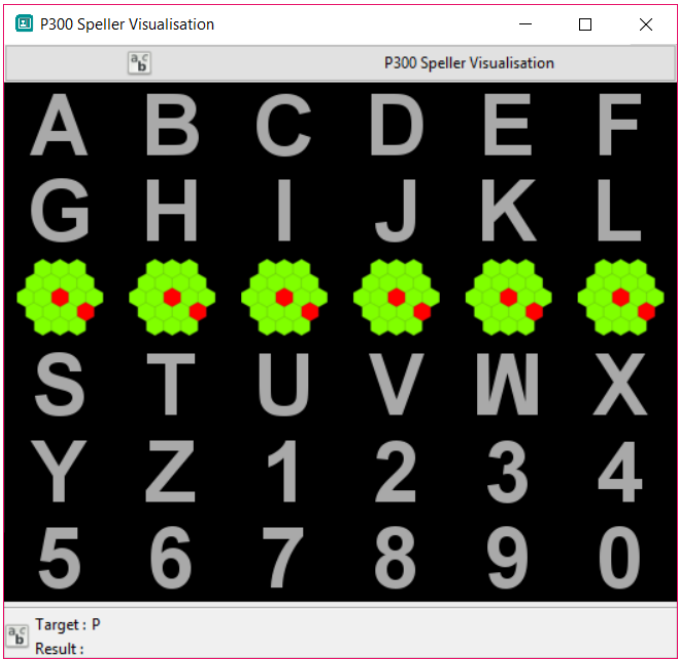


Figura 4-4. Interfaz gráfica de la aplicación desarrollada mediante el paradigma HSRD.

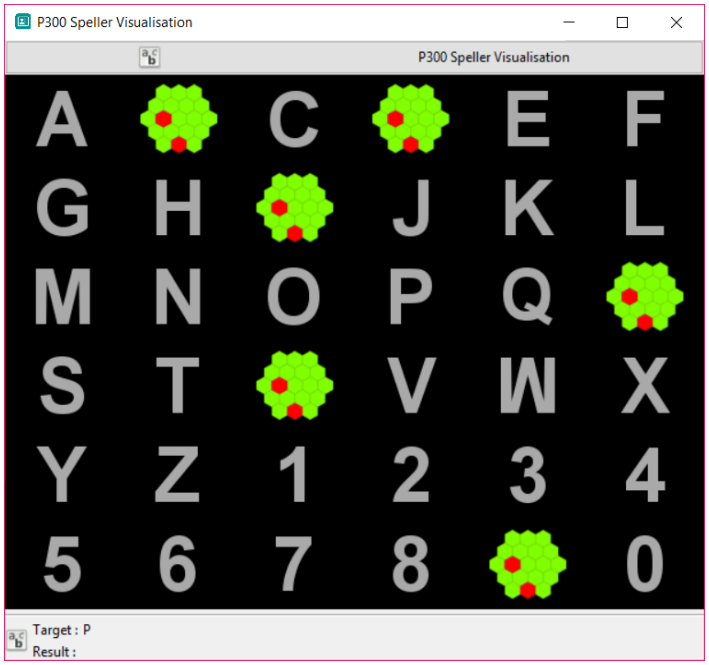


Figura 4-5. Interfaz gráfica de la aplicación desarrollada mediante el paradigma HSRDCBP.

4. Adquisición de la señal EEG

La señal EEG se adquiere a través de electrodos situados en un gorro que se coloca sobre la cabeza del usuario, cubriendo el cuero cabelludo. En la Figura 4-6, se muestra la configuración típica para obtener los potenciales evocados P300. Esta configuración utiliza 8 electrodos, distribuidos

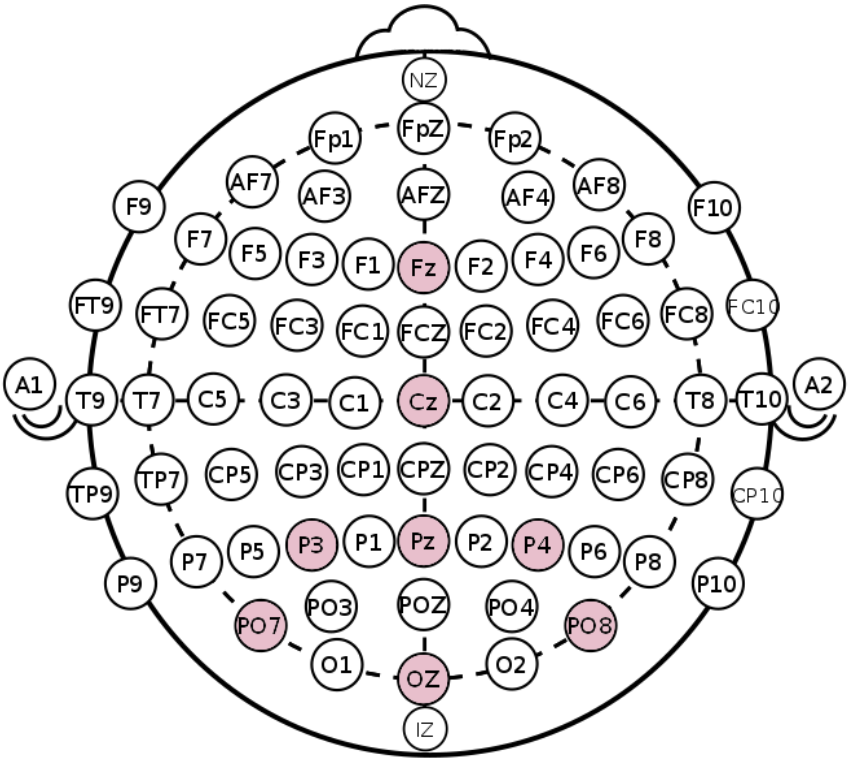


Figura 4-6. Disposición de los electrodos utilizados para adquirir la señal EEG, según el sistema internacional 10-20.

según el Sistema Internacional 10-20, llamados Fz, Cz, Pz, P3, P4, PO7, PO8 y Oz. Además, utiliza el electrodo colocado en el lóbulo derecho como referencia y el electrodo FPz como tierra.

Como ya se ha explicado, es necesario añadir un gel conductor entre los electrodos y el cuero cabelludo de los sujetos, con el fin de reducir la impedancia del cráneo y amplificar la señal deseada. El proceso de amplificación se lleva a cabo utilizando el amplificador *g.USBamp,* desarrollado por la empresa *G·Tec*. El rango de entrada de dicho amplificador abarca los ±250 mV, con una resolución inferior a 30 nV, pudiendo registrarse cualquier señal electrofisiológica sin ningún tipo de *hardware* adicional.

La aplicación fija la frecuencia de muestreo en 256 Hz, siendo ésta una frecuencia adecuada para análisis del EEG ya que cumple el teorema de Nyquist, evitando el solapamiento en el dominio de la frecuencia.

5. Procesado del EEG mediante OpenViBE

La plataforma *OpenViBE* permite al usuario configurar los distintos parámetros que caracterizan las aplicaciones, generar los potenciales evocados P300, procesar la señal EEG de forma adecuada y presentar al usuario la selección que ha realizado.

La implementación de las aplicaciones de selección de caracteres se ha realizado utilizando *OpenViBE p300 Speller*. La finalidad de esta plataforma es facilitar el desarrollo, la investigación y la

monitorización de aplicaciones en tiempo real que requieren adquirir la señal EEG y procesarla, utilizando diferentes señales de control.

OpenViBE p300 Speller se compone de dos módulos: *OpenViBE Acquisition Server* y *OpenViBE Designer*. Estos módulos son programas independientes que se comunican entre sí a través de un protocolo basado en TCP/IP. Sin embargo, como las aplicaciones desarrolladas no requieren un sistema distribuido, los módulos se comunican en la máquina local.

En la Figura 4-7, se muestra la interfaz del módulo *OpenViBE Acquisition Server*. En este módulo, la señal se procesa de forma síncrona en bloques que contienen un número determinado de muestras. Cada vez que se adquiere un nuevo bloque de datos, se envía al módulo *OpenViBE Designer*, quien se encarga de producir las señales de control[37].

El módulo *OpenViBE Acquisition Server* permite configurar el amplificador utilizado para amplificar la señal EEG, el puerto de conexión del servidor y el número de muestras enviado en cada bloque de datos. También permite configurar la frecuencia de muestreo, así como el número de canales o el nombre de éstos [37].

El módulo *OpenViBE Designer* está formado por cuatro módulos comunicados entre sí: "Monitorización de la señal"," Adquisición", "Clasificador" y "Test Online"[38].

La adquisición de la señal EEG se realiza utilizando los módulos "Monitorización de la señal" y "Adquisición". El módulo "Adquisición" obtiene la señal EEG analizada y marcada con eventos que facilitan la distinción de las señales de control. Proporciona al usuario una interfaz gráfica que permite configurar los distintos parámetros del sistema para poder realizar el entrenamiento del clasificador. La aplicación está optimizada para que la interfaz se adapte a las pantallas panorámicas. Además, permite iniciar, parar, suspender o continuar la ejecución de las aplicaciones [38].

El módulo "Clasificador de entrenamiento" se encarga de extraer y traducir las características recibidas del módulo "Adquisición". Después de llevar a cabo la traducción, se calculan los pesos para obtener el clasificador.

El último módulo, "Test Online", obtiene la señal EEG analizada y marcada con eventos que facilitan la distinción de las señales de control para poder realizar un análisis *offline* gracias al almacenamiento de la señal en el dispositivo. Además, proporciona al usuario una interfaz gráfica que permite configurar los distintos parámetros del sistema para poder realizar la fase de test *online*, así como iniciar, parar, suspender o continuar la ejecución de las aplicaciones. La aplicación está optimizada para que la interfaz se adapte a las pantallas panorámicas [38].

OpenViBE p300 Speller propone una solución para implementar el paradigma *odd-ball* visual con el fin de generar los potenciales evocados P300 a través de una matriz cuyas filas y columnas se iluminan de forma aleatoria. Esta solución sugiere utilizar, como métodos de extracción de características, un filtrado paso-banda con frecuencias de corte de 1 y 20 Hz, un submuestreo por un factor 4 y un promediado sincronizado, y un algoritmo LDA como método de traducción.

En el Trabajo Fin de Grado realizado, se ha mantenido la utilización de estos métodos, pero se ha adaptado la aplicación para poder utilizar los paradigmas desarrollados. Para ello, se han modificado solamente los módulos "Adquisición" y "Test Online" .

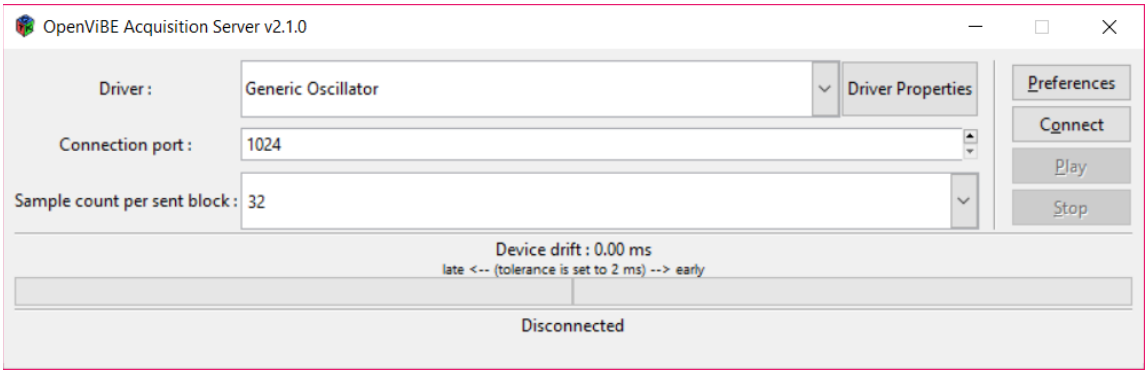


Figura 4-7. Interfaz del módulo *OpenViBE Acquisition Server*.

En el siguiente apartado, se describe la configuración de los parámetros realizada en el módulo *OpenViBE Designer*.

5.1. Configuración de parámetros

Al iniciar la aplicación desarrollada, se inicia el módulo "Monitorización de la señal" para adquirir la señal. En este módulo se pueden configurar los electrodos a partir de los cuales se obtiene la señal EEG y el filtro temporal que se quiere utilizar.

A continuación, se inicia el módulo "Adquisición" en el que el usuario puede configurar diversos parámetros. Antes de iniciar la aplicación, es necesario establecer ciertos parámetros que determinarán el funcionamiento del paradigma *odd-ball*. Considerando como intento (*trial*) la selección de carácter determinado, éste puede descomponerse en una serie de fases cuya duración elige el usuario en función de la finalidad de la sesión. A continuación, se describen los parámetros que se pueden configurar en el módulo "Adquisición" [39].

- ***Launch Box:***
  - *Time to send:* establece la pausa que inicia la ejecución del intento (*trial*).
- ***P300 Speller Stimulator Box:***
  - *Number of rows:* establece el número de filas de la matriz que genera los P300.
  - *Number of columns:* establece el número de columnas de la matriz que genera los P300.
  - *Number of repetitions:* establece el número de iluminaciones de las filas y columnas que componen la matriz.
  - *Number of trials:* establece el número de caracteres que el usuario debe escribir.
  - *Flash duration (in sec):* establece la duración de la iluminación.
  - *No flash duration (in sec):* establece la duración entre iluminaciones.
  - *Inter-repetition delay (in sec):* establece la duración del intervalo entre secuencias de iluminaciones. Especifican el número de segundos que pasa entre

dos iluminaciones. Este valor es el mismo que el del parámetro *No flash duration.*

- *Inter-trial delay (in sec):* establece la pausa entre secuencias de caracteres. Especifican el número de segundos que pasa entre la selección de un carácter y el siguiente. Para ello, se considera la iluminación de todas las filas y columnas de la matriz.

- ***Target Letter Generation Box:***
  - *Word to Spell:* identifica el grupo de caracteres que el usuario debe escribir. Si no se especifica una palabra que escribir, se utilizará un número de caracteres escogido aleatoriamente.
  - *Delay Before Sending (in sec):* establece la pausa que finaliza la ejecución del intento (*trial*).
- ***P300 Speller Visualisation Box:*** en este módulo se pueden configurar parámetros sobre el aspecto de la interfaz, como son el tamaño o el color de los caracteres y el tamaño y el color de las iluminaciones.
- ***Generic Stream Writer Box:*** en este módulo se especifica la dirección en la que se almacena el archivo *.ov*.

En la Tabla 4-1 se muestran los parámetros involucrados en la duración de un intento, así como los valores que se han utilizado en este trabajo.

En el módulo "Clasificador de entrenamiento", se pueden configurar la dirección en la que se encuentra el archivo *.ov* que se debe cargar para calcular el clasificador, así como el filtro temporal, el factor de diezmado o el promediado de las épocas que se desea utilizar. En cuanto a los parámetros del clasificador, se puede configurar el tipo de algoritmo utilizado en la clasificación.

Por último, en el módulo "Test Online", se puede configurar un conjunto de parámetros utilizados en el resto de módulos. Estos parámetros son el filtro temporal, el factor de diezmado o el promediado de las épocas que se desea utilizar. También se puede configurar la dirección en la que se desea almacenar el archivo *.ov* con el resultado de la prueba. En cuanto a los parámetros que determinarán el funcionamiento del paradigma *odd-ball* de la aplicación ejecutada, se pueden configurar todos aquellos a los que se hace referencia en la configuración del módulo "Adquisición".

6. Guía de usuario

Con el fin de facilitar la comprensión del funcionamiento de la aplicación, se ha realizado una guía de usuario que resume brevemente cómo se utiliza cada aplicación y cuáles son sus principales funcionalidades.

En el Anexo B, se muestran las guías de usuario realizadas para las diferentes aplicaciones desarrolladas a lo largo del trabajo.

TABLA 4-1. Parámetros que establecer la línea de tiempo de cada intento (trial).

PARÁMETRO	VALOR
Time to send	1
Flash duration (in sec)	0.0625
No flash duration (in sec)	0.125
Inter-trial delay (in sec)	3
Delay Before Sending (in sec)	1

7. Procedimiento de evaluación

Las aplicaciones desarrolladas han sido evaluadas por cinco sujetos sanos, cuatro hombres y una mujer, cuyas edades comprenden desde los 24 hasta los 33 años, como se muestra en la Tabla 4-2.

A continuación, se describen los procedimientos de evaluación llevados a cabo en cada una de las sesiones. Antes de finalizar el capítulo, se expone el cuestionario de satisfacción utilizado para evaluar las aplicaciones desarrolladas.

7.1. Sesiones de evaluación

Para evaluar las aplicaciones desarrolladas fueron necesarias 4 sesiones de evaluación en días consecutivos, una sesión por cada paradigma. Cada sesión consistía en una fase de entrenamiento y una fase de test *online* en las que se utilizó una secuencia de 10 iluminaciones y se conservaron los parámetros relativos a la duración del intento, mostrados en la Tabla 4-1. En la fase de entrenamiento, los usuarios debían escribir una palabra de 6 letras (“puntos”). Sin embargo, en la fase de test *online*, debían escribir cinco palabras cada una de ellas de 6 letras (“cabras”, “hinojo”, “gacela”, “sabana”, “agosto”). El orden de las palabras fue siempre el mismo. Las especificaciones de la fase de test *online* aparecen recogidas en la Tabla 4-3.

Una vez completada la fase de entrenamiento, se obtuvo el clasificador de cada usuario mediante la herramienta *OpenViBE p300 Speller*. Para ello, se cargó el archivo *.ov* generado por la plataforma *OpenViBE* y se calculó el vector de pesos, utilizando el algoritmo LDA.

En la primera sesión de evaluación, se realizó una prueba utilizando el paradigma tradicional para poder realizar una comparación con las aplicaciones desarrolladas en igualdad de condiciones. Durante el proceso de selección, los usuarios debían fijar la atención en el carácter que debían seleccionar. Para ello, se aconsejó a los usuarios que contaran el número de veces que se iluminaba el carácter que debían seleccionar. La duración de la primera sesión fue de 20 minutos, aproximadamente.

TABLA 4-2. Edad y sexo de los sujetos que realizaron las sesiones de evaluación.

	Edad	**Sexo**
U01	33	Masculino
U02	24	Femenino
U03	25	Masculino
U04	26	Masculino
U05	26	Masculino

TABLA 4-3. Especificaciones de la fase de calibración.

FASE DE TEST ONLINE		
RONDAS	INTENTOS	SECUENCIAS
5	6	10

En la segunda sesión, se evaluó la aplicación desarrollada utilizando el paradigma HSRD. Durante el proceso de selección, los usuarios debían fijar la atención en el carácter que debían seleccionar. Para ello, los usuarios debían contar el número de puntos situados dentro del panal que aparece en cada iluminación. En la Figura 4-8, se muestra a un sujeto realizando la fase de test *online*. La duración de la segunda sesión también fue de alrededor de 20 minutos.

En la tercera sesión, se llevó a cabo la evaluación de la aplicación desarrollada utilizando el paradigma CBP. Durante el proceso de selección, los usuarios debían fijar la atención en el carácter que debían seleccionar. Para ello, al igual que en el paradigma tradicional, se aconsejó a los usuarios que contaran el número de veces que se iluminaba el carácter que debían seleccionar. En la Figura 4-9, se muestra a un sujeto realizando la fase de test *online* perteneciente a esta sesión. La duración de la tercera sesión fue de 30 minutos aproximadamente.

En la cuarta y última sesión, se propuso evaluar un nuevo paradigma al que se le ha llamado HSRDCBP. Como ya se ha explicado, se trata de una aplicación que unifica ambos paradigmas. Durante el proceso de selección, los usuarios debían fijar la atención en el carácter que debían seleccionar. Para ello, al igual que en el paradigma HSRD, los usuarios debían contar el número de puntos situados dentro del panal que aparece en cada iluminación. En la Figura 4-10, se muestra al sujeto U05 realizando la fase de test *online*. La duración de la cuarta sesión fue alrededor de 28 minutos.

En el capítulo siguiente, se muestran los resultados de las diferentes sesiones de evaluación para cada uno de los usuarios.

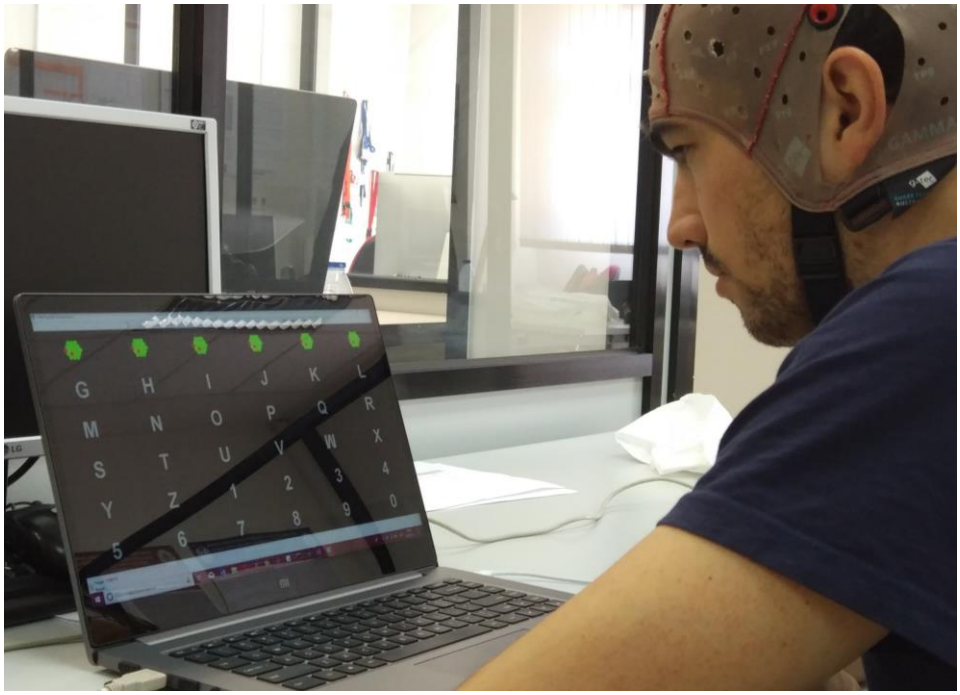


Figura 4-8. Sujeto U03 evaluando la aplicación en la segunda sesión.

Figura 4-9. Sujeto U05 evaluando la aplicación en la tercera sesión.

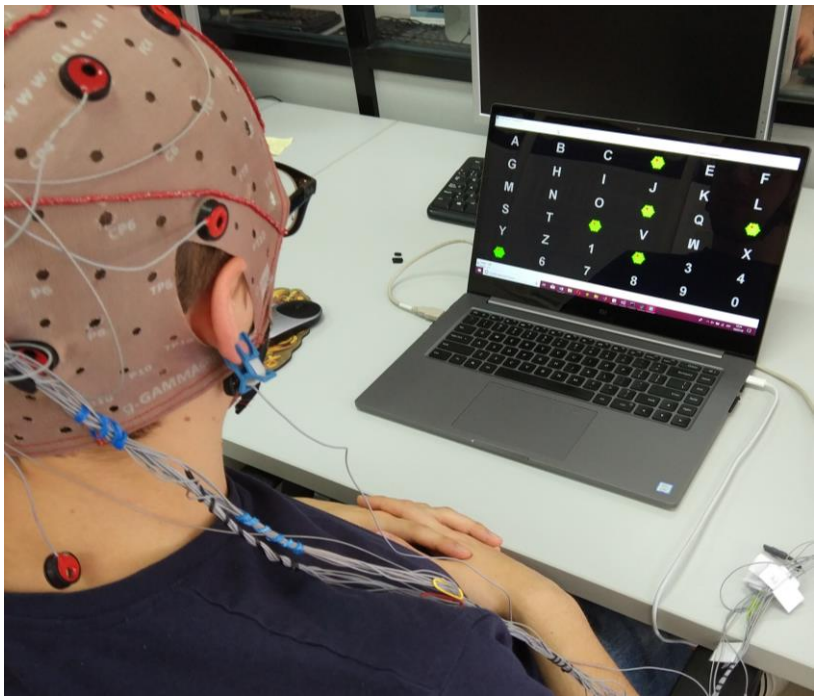


Figura 4-10. Sujeto U05 evaluando la aplicación en la cuarta y última sesión.

7.2. Cuestionario de satisfacción

Una vez finalizadas las sesiones de evaluación, se entregó a cada usuario un cuestionario de satisfacción, para recoger su experiencia personal en las diferentes sesiones realizadas y las sugerencias de mejora de las aplicaciones desarrolladas.

El cuestionario de satisfacción está compuesto por 10 afirmaciones, tanto positivas como negativas, y se muestra en el Anexo C.

CAPÍTULO V

RESULTADOS

En este capítulo, se muestran los resultados obtenidos en las sesiones de evaluación que realizaron cada uno de los sujetos. El procedimiento de evaluación utilizado se ha descrito en el capítulo 4. Además, se incluyen los resultados de los cuestionarios de satisfacción y las sugerencias de mejora propuestas por los usuarios.

1. Sesiones de evaluación

En este apartado, se muestran los resultados obtenidos por los usuarios en las diferentes sesiones de evaluación realizadas.

La precisión de cada sesión se ha obtenido mediante una fase de test *offline*, en la que se ha realizado un procesado de la señal EEG alternativo al realizado por la plataforma *OpenViBE*. Para ello, se ha aplicado un filtrado *hardware* paso-banda entre 0.1 y 60 Hz y un filtro de ranura de 50 Hz, con el objetivo de eliminar las interferencias de la red eléctrica. Además, se han empleado como métodos de extracción de características un promediado sincronizado, para visualizar los potenciales P300; un submuestreo, para obtener las precisiones; y un filtrado espacial CAR, para tratar de reducir el difuminado espacial; y como método de clasificación de características, un sistema de discriminación lineal para determinar si se ha producido o no un P300 en cada iluminación.

En la extracción de características, se realiza un submuestreo de la señal EEG utilizando una ventana temporal de un segundo. Esta ventana debe ser lo suficientemente grande como para detectar los potenciales evocados P300 y lo suficientemente pequeña como para que no se solapen las iluminaciones consecutivas. La frecuencia de submuestreo utilizada es de 20 Hz, es decir, se generan 20 muestras por cada segundo de registro.

Además, en el análisis *offline* se ha realizado un estudio variando el número de iluminaciones con el fin de identificar el número óptimo que maximiza la precisión en la clasificación y determinar qué paradigma obtiene mejor rendimiento. Los resultados obtenidos se analizarán en los siguientes capítulos.

En la Tabla 5-1, se muestra un resumen de los resultados obtenidos por cada usuario en las diferentes sesiones realizadas para evaluar las aplicaciones desarrolladas. Además, se exponen los resultados obtenidos con el paradigma tradicional para poder realizar una comparación con los obtenidos en los paradigmas desarrollados. A continuación, se describen para cada paradigma los resultados que ha obtenido cada usuario.

1.1. Row-Column Paradigm

La primera sesión de evaluación consistió en evaluar el paradigma tradicional. Esta primera sesión se realizó para poder comparar los resultados obtenidos con los diferentes paradigmas desarrollados.

TABLA 5-1. Resultados obtenidos en las sesiones de evaluación.

SUJETO U01				
	RCP	**HSRD**	**CBP**	**HSRDCBP**
Precisión	86.67%	96.67%	76.67%	**100%**
Duración	20 min	19 min	27 min	30 min
Caracteres Erróneos	14	5	16	1
Errores Adyacentes	8	5	9*	1*
SUJETO U02				
	RCP	**HSRD**	**CBP**	**HSRDCBP**
Precisión	96.67%	**100%**	93.33%	**100%**
Duración	20 min	19 min	37 min	29 min
Caracteres Erróneos	9	3	6	2
Errores Adyacentes	7	3	5*	2*
SUJETO U03				
	RCP	**HSRD**	**CBP**	**HSRDCBP**
Precisión	**100%**	93.33%	56.67%	**100%**
Duración	20 min	20 min	28 min	28 min
Caracteres Erróneos	3	5	16	4
Errores Adyacentes	3	5	7*	2*

SUJETO U04				
	RCP	**HSRD**	**CBP**	**HSRDCBP**
Precisión	86.67%	**100%**	**100%**	**100%**
Duración	19 min	20 min	26 min	28 min
Caracteres Erróneos	13	0	0	0
Errores Adyacentes	10	0	0	0
SUJETO U05				
	RCP	**HSRD**	**CBP**	**HSRDCBP**
Precisión	93.33%	**100%**	96.67%	**100%**
Duración	21 min	20 min	32 min	25 min
Caracteres Erróneos	3	0	4	1
Errores Adyacentes	1	0	3*	1*

* Tanto en el paradigma CBP como en el paradigma unificado HSRDCBP, los errores adyacentes se refieren a selecciones cuyos caracteres se localizan en la misma fila o columna de la matriz virtual en la que se encuentre el carácter que se desea seleccionar.

El primer sujeto, identificado como sujeto U01, seleccionó 14 caracteres de forma errónea el transcurso de la tarea. De los 14 errores cometidos, 8 fueron errores producidos por la selección de un carácter adyacente. Tras la fase de test *offline*, el sujeto U01 obtuvo una precisión del 86.67%.

El segundo sujeto, identificado como sujeto U02, cometió 9 errores en el transcurso de la tarea que debía realizar, de los cuales 7 fueron errores debidos a distracciones adyacentes. A pesar de los errores cometidos, tras la fase de test *offline* se obtiene una precisión del 96.67%.

El tercer sujeto, identificado como sujeto U03, cometió tres errores en la realización de las tareas propuestas. Todos los errores registrados pertenecen a selecciones de caracteres adyacentes. Al realizar la fase de test *offline* se obtuvo una precisión del 100%.

El cuarto sujeto, identificado como sujeto U04, cometió 13 errores en la evaluación del paradigma tradicional. De los 13 errores cometidos, 10 fueron errores producidos por la selección de un carácter adyacente. Analizando los datos mediante una fase *offline* se obtiene una precisión del 86.67%.

El quinto y último sujeto, identificado como sujeto U05, cometió tres errores en la evaluación del paradigma RCP. De los 3 errores registrados, solamente uno fue debido a la selección de un carácter adyacente al que se debía seleccionar. Tras la fase de test *offline* realizada para esta sesión, se obtiene una precisión del 93.33%.

1.2. Checkerboard Paradigm

En la tercera sesión de evaluación, los usuarios probaron la aplicación desarrollada utilizando el paradigma CBP. Antes de exponer los resultados obtenidos por cada usuario, cabe destacar que los errores adyacentes en este caso se refieren a la selección de caracteres dispuestos de forma adyacente en las matrices virtuales blanca y negra.

El primer sujeto cometió 16 errores en el transcurso de la tarea. De los 16 errores registrados, 9 fueron errores producidos por la selección de un carácter adyacente. Tras la fase de test *offline* se obtiene una precisión del 76.67%, bastante inferior a la precisión obtenida con el paradigma tradicional.

El sujeto U02 cometió 6 errores en el transcurso de la tarea que debía realizar, de los cuales 5 fueron errores por distracciones adyacentes. Sin embargo, a pesar de los errores cometidos, tras la fase de test *offline* se obtiene una precisión del 93.33%. Comparando este resultado con el obtenido utilizando el paradigma tradicional, se puede observar una reducción en la precisión.

El sujeto U03 al igual que el sujeto U01 cometió 16 errores en la realización de las tareas propuestas en la fase de test *offline*. De los 16 errores, solo 7 fueron errores debidos a distracciones adyacentes. La precisión obtenida en la fase de test *offline* es de 56.67%, la menor precisión obtenida de las sesiones de evaluación realizadas.

El cuarto sujeto no cometió ningún error en la evaluación del paradigma desarrollado. Por lo tanto, tras el análisis *offline* de los datos, la precisión obtenida es del 100%. Si se compara esta precisión con la obtenida en la primera sesión de evaluación se puede observar que se produce una mejora evidente en la comunicación del usuario.

El quinto y último sujeto cometió cuatro errores en esta sesión de evaluación. De los 4 errores producidos, 3 fueron debidos a la selección de un carácter adyacente al que se debía seleccionar. Tras la fase de test *offline* realizada para esta sesión, se obtiene una precisión del 96.67%.

1.3. Honeycomb-Shaped Red Dots Paradigm

El paradigma HSRD se evaluó en la segunda sesión. Antes de analizar los datos obtenidos por cada usuario cabe destacar que los errores adyacentes se refieren a la selección de caracteres dispuestos en la misma fila o columna que el carácter que se debe seleccionar.

El primer sujeto cometió cinco errores en la realización de las tareas de la fase de test *online*. Todos los errores cometidos fueron errores producidos por la selección de un carácter adyacente al carácter que se debía seleccionar. A partir de los datos obtenidos se ha realizado un análisis *offline* en el que se ha obtenido una precisión del 96.67%.

El sujeto U02 cometió tres errores en la selección de caracteres propuesta. Al igual que ocurre con el usuario anterior, todos los errores son debidos a la selección de un carácter adyacente al que

el usuario debía seleccionar. Analizando los datos mediante una fase *offline* se obtiene una precisión del 100%.

El tercer sujeto, sujeto U03, al igual que el sujeto U01 cometió cinco errores en la realización de las tareas propuestas en la fase de test *offline.* Como ocurre en los sujetos anteriores, los errores cometidos son errores producidos por seleccionar un carácter adyacente al que se debía seleccionar. En el análisis *offline* realizado posteriormente, se ha obtenido un 93.33% de precisión.

El cuarto sujeto no cometió ningún error en la evaluación del paradigma, por lo que la precisión obtenida al realizar el análisis *offline* es del 100%.

El sujeto U05 tampoco cometió ningún error en la selección de caracteres. Realizando el análisis *offline*, al igual que en el sujeto anterior, se obtiene una precisión del 100%.

1.4. Honeycomb-Shaped Red Dots – Checkerboard Paradigm

Para realizar la evaluación del nuevo paradigma HSRDCBP se realizó una cuarta y última sesión. Al igual que ocurre en el paradigma CBP, los errores adyacentes se refieren a la selección de caracteres dispuestos de forma adyacente en las matrices virtuales blanca y negra.

El sujeto U01 cometió un solo error en la tarea de evaluación propuesta para este paradigma. El error cometido fue producido por la selección de un carácter adyacente al carácter que se debía seleccionar. A partir del análisis *offline* realizado se obtiene una precisión del 100%.

El segundo sujeto cometió dos errores en la tarea de selección de caracteres. Al igual que ocurre con el usuario anterior, todos los errores son debidos a la selección de caracteres adyacen.es a los que se debían seleccionar. En la fase de análisis *offline*, se obtiene una precisión del 100%.

El sujeto U03 cometió cuatro errores en la realización de las tareas propuestas en la fase de test *online.* Solo dos de los errores cometidos son errores producidos por seleccionar un carácter adyacente. A partir de estos datos, se ha realizado un análisis *offline* en el que se ha obtenido un 100% de precisión.

Al igual que ocurre en las otras dos sesiones, el cuarto sujeto no cometió ningún error en la evaluación del paradigma, por lo que la precisión obtenida al realizar el análisis *offline* es del 100%.

Por último, el sujeto U05 cometió un único error en la tarea de selección de caracteres. El error registrado se produce debido a la selección de un carácter adyacente al que se debía seleccionar. Realizando el análisis offline, al igual que en el sujeto anterior, se obtiene una precisión del 100%.

2. Cuestionario de satisfacción

Como ya se ha detallado en el capítulo anterior, una vez finalizadas las sesiones de evaluación, se entregó a cada usuario un cuestionario de satisfacción.

El cuestionario de satisfacción estaba compuesto por 10 preguntas, de las cuales 7 se respondían utilizando una escala de satisfacción del 1 a l 5, siendo 1 poco satisfactorio y 5 muy satisfactorio. En la Tabla 5-2, se recoge la media de cada cuestión, cuyos resultados se discutirán en el próximo capítulo.

TABLA 5-2. Resultados obtenidos en el cuestionario de satisfacción.

Pregunta	Paradigma	Puntuación Media	Desviación
1. La interfaz de la aplicación es intuitiva y fácil de entender.	**RCP**	4.20	1.30
	HSRD	3.60	0.89
	CBP	4.20	1.30
	HSRDCBP	3.60	0.89
2. Me ha resultado sencillo seleccionar las letras de la matriz.	**RCP**	2.60	0.55
	HSRD	4.60	0.55
	CBP	3.20	0.84
	HSRDCBP	5	0
3. La iluminación de la letra resulta molesta.	**RCP**	2.60	1.52
	HSRD	1.40	0.55
	CBP	2.40	1.34
	HSRDCBP	1.40	0.55
4. El tiempo necesario para escribir una letra es elevado.	2.60		1.14
5. Habitualmente se seleccionan letras que no deseo seleccionar.	2		0.71
6. La utilización del gorro y los electrodos es poco práctica.	3.80		0.84
7. La duración de las sesiones realizadas es adecuada.	4.20		1.30
8. ¿Volvería a participar en proyectos con estas características?	Sí		0
9. En su opinión, ¿qué paradigma es mejor?	HSRDCBP		0

De las 3 preguntas restantes, una de las preguntas estaba formulada de la siguiente forma: "¿Volvería a participar en proyectos con estas características?". Los usuarios U01 y U04 contestaron a la pregunta utilizando la siguiente respuesta: "Sí, aunque me ha supuesto un esfuerzo". Los usuarios U02, U03 y U05 respondieron utilizando otra de las respuestas, formulada como: "Sí, me ha parecido un proyecto interesante".

La pregunta expresada como: "En su opinión, ¿qué debería mejorarse de las diferentes aplicaciones para futuras versiones?", buscaba recoger las sugerencias de los usuarios para mejorar las dos aplicaciones. Tanto el usuario U01 como el usuario U04 expusieron la necesidad de reducir el número de iluminaciones de cada carácter. Además, el usuario U04 indicó que, en su opinión, son necesarios algunos segundos (entre 5 y 10 segundos) entre la selección de caracteres.

Para finalizar, la última pregunta hacía referencia a cuál era el paradigma más completo, formulada como: "En su opinión, ¿qué paradigma es mejor?", a la que todos los usuarios contestaron utilizando la respuesta formulada como: "HSRDCBP".

DISCUSIÓN

El objetivo de este capítulo es discutir los resultados mostrados en el capítulo anterior. En primer lugar, se analizarán los resultados obtenidos en las diferentes sesiones de evaluación. A continuación, se discuten los resultados obtenidos en los cuestionarios de satisfacción y las sugerencias de mejora propuestas por los usuarios. Por último, se comparan los resultados con otros estudios y se determinan las limitaciones de las aplicaciones desarrolladas.

1. Sesiones de evaluación

A continuación, se discuten los resultados obtenidos en las diferentes sesiones de evaluación realizadas para los cinco sujetos.

1.1. Row-Column Paradigm

En la evaluación del paradigma tradicional, ninguno de los sujetos consiguió finalizar las tareas propuestas sin errores. Los sujetos U04 y U01 fueron los usuarios que más errores cometieron con 13 y 14 selecciones incorrectas, respectivamente. El siguiente sujeto que más errores cometió fue U02 con 9. Tanto el sujeto U03 como el sujeto U05 cometieron 3 errores en las tareas de evaluación propuestas. En todos los casos, la mayoría de los errores cometidos se deben a selecciones de caracteres adyacentes, lo que sugiere que la iluminación de filas y columnas puede ser inadecuada.

En términos de precisión, solamente el usuario U03 consiguió alcanzar el 100%. El sujeto U02 consiguió una precisión del 96.67%, seguido del sujeto U05, quien consiguió un 93.33%. A pesar del alto número de errores cometidos por los usuarios U04 y U01, ambos consiguieron una precisión del 86.67%. Como se muestra en la Tabla 6-1, la precisión media obtenida por el paradigma RCP es de casi un 93%. Además, en esta tabla se recoge la desviación obtenida para este paradigma.

Cabe destacar que, a pesar de que el usuario U03 cometió 3 errores en la evaluación de las tareas propuestas obtuvo una precisión del 100%, lo que indica que el procesado realizado por la plataforma *OpenViBE* puede ser ineficiente. Además, la precisión obtenida por este sujeto podría no ser relevante debido a que el usuario tuvo que repetir esta sesión de evaluación al finalizar el resto de sesiones.

1.2. Checkerboard Paradigm

En la evaluación del paradigma CBP, solamente el sujeto U04 consiguió terminar las tareas de evaluación sin ningún error. Los sujetos U02 y U05 cometieron 6 y 4 errores, respectivamente. Por último, tanto el sujeto U01 como el sujeto U03 cometieron 16 errores en las tareas de evaluación realizadas. A primera vista, estos resultados no favorecen la utilización del paradigma CBP frente al paradigma tradicional.

En términos de precisión, solo el usuario U04 alcanzo el 100% de precisión. A esta precisión le siguen el 96.67% perteneciente al usuario U05 y el 93.33% perteneciente al usuario U02. Los usuarios que más errores cometieron obtuvieron precisiones del 76.67% (precisión del sujeto U01) y 56.67% (precisión del sujeto U03). Como se muestra en la Tabla 6-1, la precisión media obtenida

TABLA 6-1. Resultados obtenidos en las sesiones de evaluación.

	Puntuación Media	Desviación
RCP	92.67%	5.96
HSRD	98%	2.98
CBP	84.67%	18.04
HSRDCBP	100%	0

por el paradigma CBP es de casi un 85%. En esta tabla también se muestra la desviación obtenida para este paradigma.

Analizando estos datos, es destacable comprobar que, a pesar de tener el mismo número de errores, el usuario U03 tiene un 20% menos de precisión que el usuario U01. Esto puede ser debido a que el usuario U03 no realizó las tareas de evaluación de forma adecuada o a que el clasificador obtenido a través de la fase de entrenamiento no se calculó correctamente.

1.3. Honeycomb-Shaped Red Dots Paradigm

En la evaluación del paradigma HSRD, solo los usuarios U04 y U05 no cometieron errores al realizar las tareas propuestas para la evaluación. Tanto el usuario U01 como el usuario U03 cometieron 5 errores en la selección de caracteres para evaluar este paradigma. De los 5 errores cometidos, los 5 fueron errores por distracciones adyacentes, es decir, por la iluminación de caracteres adyacentes al que se deseaba seleccionar. Por último, el sujeto U02 cometió 3 errores en la evaluación de las tareas propuestas. Al igual que ocurre con los sujetos U01 y U03, los errores cometidos eran debidos a selecciones de caracteres adyacentes. Estos datos sugieren que la iluminación de filas y columnas propuesta por este paradigma podría no ser la indicada.

En términos de precisión, los usuarios U02, U04 y U05 obtuvieron la máxima precisión posible. En cuanto a los dos sujetos restantes, el sujeto U01 obtuvo una precisión del 96.67%, seguido por el usuario U03 con una precisión del 93.33%. En la Tabla 6-1, se muestra la precisión media obtenida por el paradigma HSRD, así como la desviación obtenida para este paradigma.

A pesar de que la mayoría de los sujetos no ha conseguido terminar las tareas propuestas sin errores, las precisiones obtenidas por cada usuario son bastante elevadas. Estas cifras sugieren que el paradigma HSRD es una buena alternativa al paradigma RCP. Cabe destacar que los resultados de precisión obtenidos no concuerdan con los errores cometidos por cada sujeto. Esto es debido al procesado realizado por la plataforma *OpenViBE*.

1.4. Honeycomb-Shaped Red Dots – Checkerboard Paradigm

A pesar de que, en esta sesión de evaluación, solamente el sujeto U04 consiguió terminar la tarea propuesta sin ningún error, las precisiones obtenidas en la fase de test *offline* son del 100% para todos los sujetos. Por lo tanto, como se muestra en la Tabla 6-1, el paradigma HSRDCBP obtiene

una precisión media del 100%. En esta tabla también se muestra la desviación obtenida para este paradigma.

Estos resultados ponen de manifiesto el buen funcionamiento de la nueva aplicación desarrollada y la mala gestión del procesado de los datos que realiza la plataforma *OpenViBE*. Al unificar los dos paradigmas estudiados, se reducen los errores producidos por selecciones de caracteres adyacentes y se aumenta el nivel de atención debido a la sustitución de la iluminación por un panal con puntos rojos en su interior.

1.5. Análisis general

En la Figura 6-1, se muestra una gráfica comparativa con las precisiones obtenidas por cada sujeto en cada una de las sesiones de evaluación. Lo más destacado es que la cuarta y última sesión ha obtenido un 100% de precisión para todos los sujetos. A partir de este dato, se puede concluir que el nuevo paradigma desarrollado es la alternativa más viable al paradigma tradicional.

Como se puede observar, la precisión obtenida varía a lo largo de las diferentes sesiones de evaluación realizadas. Esta variación se debe a cambios entre sesiones de la motivación, la predisposición, la fatiga o la atención del sujeto, así como la hora a la que se realizó la sesión o incluso pequeñas variaciones en la localización del gorro [24].

A pesar de esto, es interesante analizar los potenciales evocados P300 de cada sujeto para poder identificar las causas por las que se obtuvieron las precisiones para cada uno de ellos. En la Figura 6-2, se muestra un promediado sincronizado de las muestras obtenidas en las tareas de evaluación. En color negro se muestra la respuesta del electrodo Pz y en color magenta se muestra la respuesta promediada de los electrodos PO7 y PO8.

Como se puede observar, en general se pueden identificar los potenciales evocados P300 en casi todos los sujetos. Quizá los sujetos U01 y U03 son los que muestran una respuesta más atenuada, lo que concuerda con los resultados obtenidos en las sesiones de evaluación.

Otro aspecto a valorar en las sesiones de evaluación realizadas es el número de iluminaciones necesarias para alcanzar una precisión adecuada. En la Figura 6-3, se muestran los resultados obtenidos por los cinco sujetos en las diferentes sesiones de evaluación realizadas, en función del número de iluminaciones.

Como se puede observar en la Figura 6-3, el sujeto U01 obtiene los mejores resultados con el nuevo paradigma unificado, con el que utilizando 8 iluminaciones el usuario seleccionaría la letra de forma correcta. Siguiendo a este paradigma se encuentra el paradigma HSRD, con el que no es posible alcanzar el 100% de precisión utilizando 10 iluminaciones. Por último, el paradigma CBP es el que peores resultados obtiene de los tres paradigmas estudiados. La precisión obtenida apenas llega al 90% utilizando 10 iluminaciones.

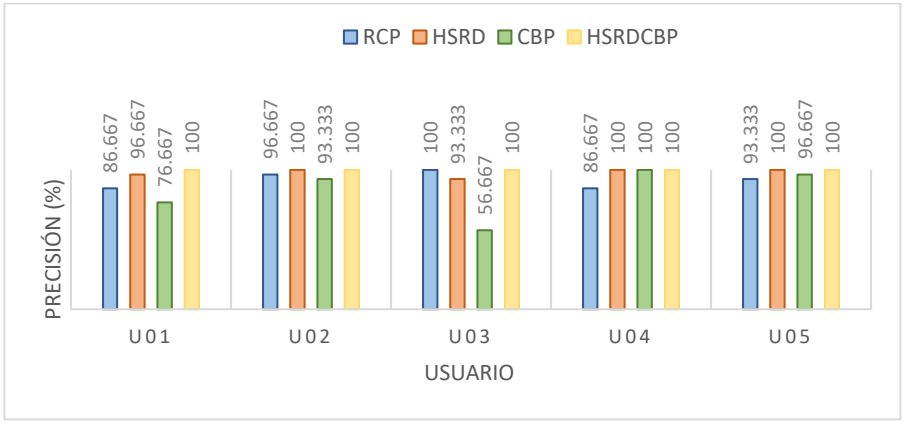


Figura 6-1. Precisiones obtenidas en cada una de las sesiones de evaluación por los diferentes sujetos.

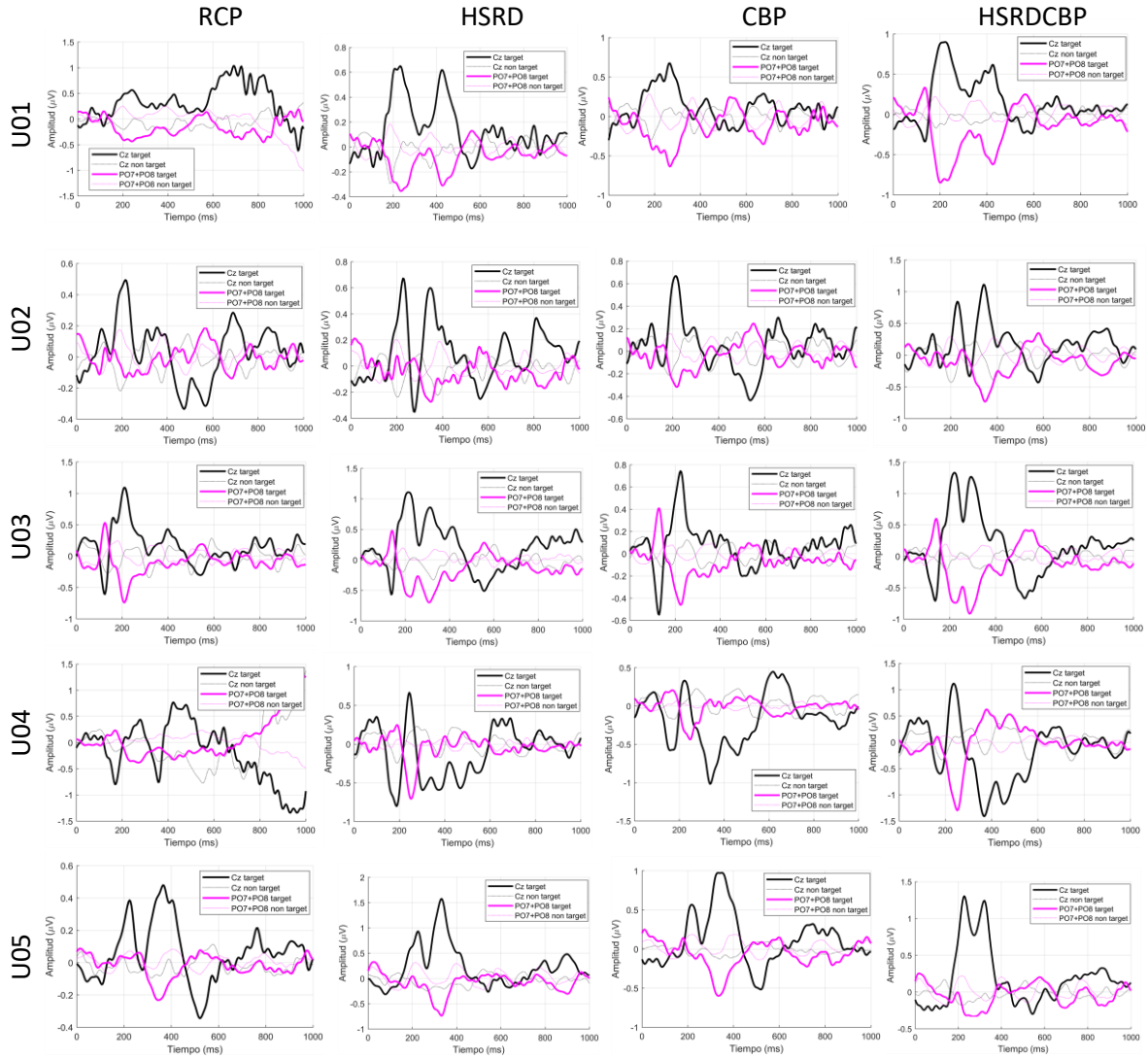


Figura 6-2. Potenciales evocados P300 captados en los electrodos Pz, PO7 y PO8 durante las sesiones de evaluación.

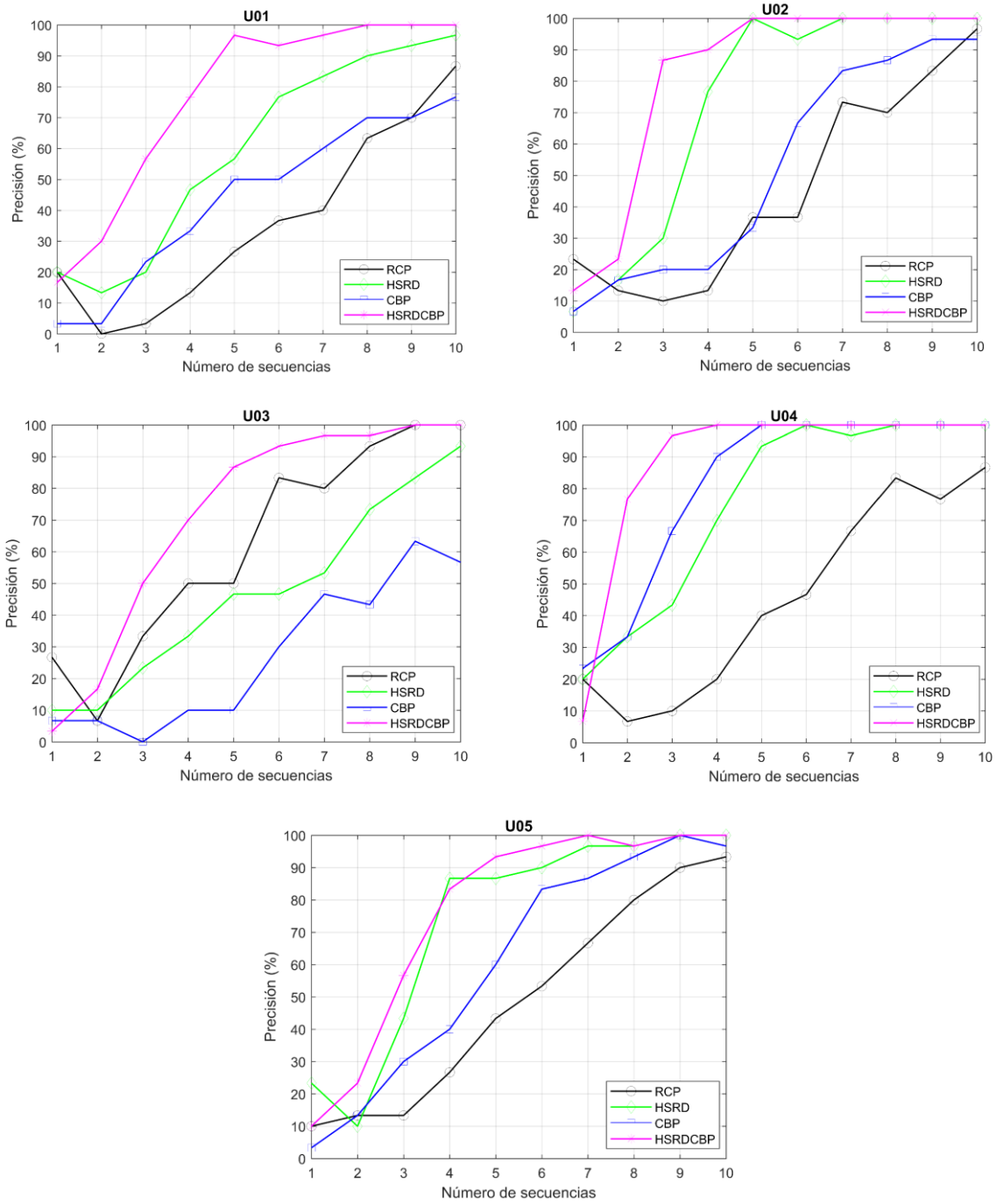


Figura 6-3. Resultados obtenidos por los sujetos en los diferentes paradigmas analizados. Se muestra la precisión en función del número de iluminaciones.

En cuanto los resultados del sujeto U02, al igual que ocurre con el usuario U01, los mejores resultados se obtienen con el paradigma HSRDCBP, con el que utilizando 5 iluminaciones el usuario seleccionaría la letra de forma correcta, con una precisión del 100%. El siguiente paradigma que mejores resultados obtiene es el HSRD, con el que también se obtiene una buena precisión utilizando pocas iluminaciones. El 100% de precisión se obtiene para 7 iluminaciones. Por último, el

paradigma CBP es el que peores resultados obtiene de los tres paradigmas estudiados. La precisión obtenida no llega al 100% utilizando 10 iluminaciones.

En el caso del sujeto U03, la mayor precisión se obtiene con el paradigma HSRDCBP, aunque los resultados no son tan adecuados como los de los usuarios U01 y U02. A continuación del paradigma HSRDCBP, los mejores resultados los obtiene el paradigma tradicional. Cabe destacar que este usuario realizó la sesión de evaluación del paradigma RCP el último día, pudiendo influir en los resultados obtenidos. Siguiendo al paradigma tradicional, se encuentra el paradigma HSRD, cuya precisión máxima supera el 95% para 10 iluminaciones. Por último, el paradigma que peores resultados obtuvo fue el CBP con una precisión inferior al 60%.

A partir de la Figura 6-3 se puede observar que el sujeto U04 es el que mejores resultados obtiene en las diferentes sesiones realizadas para evaluar los paradigmas estudiados. El paradigma HSRDCBP obtiene una precisión del 100% utilizando solamente 4 iluminaciones. El siguiente paradigma que mejores resultados obtiene es el CBP. Al contrario de lo que ocurre en el resto de los usuarios, con el paradigma CBP el usuario U04 obtiene una precisión del 100% utilizando 5 iluminaciones. Por último, el paradigma HSRD es el que peores resultados obtiene de los tres paradigmas estudiados. La precisión máxima se obtiene para 8 iluminaciones.

Por último, en el caso del sujeto U05, la mayor precisión utilizando pocas iluminaciones también se obtiene con el paradigma HSRDCBP. El sujeto necesita 9 iluminaciones para alcanzar el 100% de precisión. A continuación del paradigma HSRDCBP, los mejores resultados los obtiene el paradigma HSRD, con el mismo número de iluminaciones que el paradigma analizado anteriormente. Por último, el paradigma que peores resultados obtuvo fue el CBP con una precisión superior al 95% utilizando 10 iluminaciones.

Comparando los resultados de todos los sujetos, se puede concluir que el único paradigma que puede ser una fuerte alternativa al paradigma tradicional es el nuevo paradigma unificado HSRDCBP. A pesar de que el Trabajo Fin de Grado desarrollado se centra en el estudio de los paradigmas HSRD y CBP, se ha comprobado que los resultados son más exitosos cuando se utilizan de forma conjunta. En general cabe destacar que, entre los dos paradigmas estudiados, el paradigma HSRD consigue mejores resultados que el CBP.

Para finalizar, se ha realizado un análisis estadístico con el fin de comparar la precisión obtenida en el paradigma HSRDCBP con las precisiones de los diferentes paradigmas estudiados. Para realizar este análisis se ha utilizado el test no paramétrico "*Mann-Whitney U Test*" ya que los datos son univariados->diferencia->independientes (distintos sujetos)->valores continuos y no normales->2 grupos. En la Tabla 6-2 se muestran los resultados del *p*-valor obtenidos en las diferentes comparativas realizadas. La diferencia más significativa se produce al comparar con los resultados teóricos del paradigma HSRD, donde el *p*-valor es de 0.0019, muy inferior a 0.05. El siguiente *p*-valor más significativo es el perteneciente a la comparativa realizada con el paradigma CBP, cuyo *p*-valor es de 0.0044.

TABLA 6-2. Resultados del p-valor obtenidos.

Comparativa		*p*-valor (1)
CBP - HSRDCBP		0.0044**
Variantes HoS	***Variante HoS* - HSRDCBP**	0.0140*
	***Cake Speller* - HSRDCBP**	0.0131*
	***Center Speller* - HSRDCBP**	0.0049**
RSVP	***NoColor 116 ms* - HSRDCBP**	*0.1849*
	***Color 116 ms* - HSRDCBP**	0.0068**
	***Color 83 ms* - HSRDCBP**	0.0181*
HSRD - HSRDCBP		0.0019**

(1) Test estadístico comparativo entre HSRDCBP y cada estudio con el test no paramétrico *"Mann-Whitney U Test".*

2. Cuestionario de satisfacción

En cuanto al paradigma tradicional, los sujetos U02, U03 Y U05 opinaron que la interfaz de la aplicación es intuitiva y fácil de entender. Sin embargo, el sujeto U04 declaró que existe cierta dificultad en comprender cómo funciona. El sujeto U01 indicó que la interfaz casi no era comprensible. Estas aportaciones indican que es necesario el uso de un manual de usuario o una explicación del funcionamiento para asegurar una buena utilización de la aplicación.

Con respecto a si la selección de caracteres resulta sencilla, tres sujetos opinaron que la selección les resultó fácil. Los dos sujetos restantes indicaron que, a veces, la selección les pareció algo complicada. Tres sujetos indicaron que la iluminación de los caracteres producida en este paradigma les resultó bastante molesta. Por el contrario, los otros dos usuarios opinaron que la iluminación no era nada molesta.

Los resultados obtenidos para el paradigma HSRD demostraron que, para la mayoría de los usuarios, la interfaz apenas era intuitiva. Solamente el usuario U01 no estuvo de acuerdo con dicha opinión. En cuanto a la dificultad producida en la selección de caracteres, los usuarios U01, U02 y U04 indicaron que la selección les había resultado muy sencilla. Los dos usuarios restantes, U03 y U05, expusieron que habían seleccionado los caracteres sin mucha dificultad. Tres sujetos se pusieron de acuerdo en afirmar que la iluminación realizada en este paradigma no les había resultado nada molesta. En cambio, los dos usuarios restantes indicaron que a veces, los panales les habían causado cierta molestia.

En el análisis del paradigma CBP, la mayor parte de los usuarios indicó que la interfaz era bastante intuitiva y fácil de entender. Sin embargo, un usuario opinó que la interfaz no le había resultado nada comprensible. Solamente el usuario U01 indicó que la selección de caracteres le había resultado bastante complicada. El resto de los usuarios reflejaron cierta dificultad a la hora de seleccionar el carácter deseado. En cuanto a si la iluminación había resultado molesta, los usuarios U02, U03 y U04 indicaron cierta molestia debida a las iluminaciones. Los otros sujet os mostraron desacuerdo en esta afirmación.

La interfaz del nuevo paradigma desarrollado, el paradigma HSRDCBP, resultó ser sencilla y fácil de entender para cuatro de los cinco sujetos. En cuanto a la selección de caracteres, todos estuvieron de acuerdo en que la selección había resultado muy sencilla. Sin embargo, a la hora de valorar la molestia que producía la iluminación, los usuarios U01, U02 Y U05 indicaron que los panales no les habían resultado nada molestos, mientas que, los usuarios U03 y U04 reflejaron que algunas veces las iluminaciones les molestaron.

Con respecto a si el tiempo necesario para escribir una letra es elevado, los usuarios mostraron diferentes opiniones. Solamente un usuario opinaba que el tiempo de selección era adecuado. Los cuatro sujetos restantes mostraron su disconformidad con respecto a esta afirmación.

Tres usuarios indicaron que a veces se seleccionaban letras que no deseaban seleccionar. El usuario U05 reflejó que, durante las diferentes sesiones realizadas, apenas se habían seleccionado letras erróneas. Por el contrario, el sujeto U03 indicó que las aplicaciones seleccionaron caracteres erróneos con bastante frecuencia.

La opinión con respecto a si la utilización del gorro y los electrodos es poco práctica es diversa. Dos usuarios indicaron que su utilización no les parecía muy práctica. Otros dos usuarios reflejaron que la utilización práctica del gorro y los electrodos le pareció adecuada. Por último, un único usuario opinó que los métodos de adquisición de la señal no eran nada prácticos.

En cuanto a la duración de las sesiones realizadas, los usuarios U03, U04 y U05 reflejaron que la duración les pareció adecuada; mientras que, los dos usuarios restantes opinaron que las sesiones habían resultado largas.

Los usuarios U02, U03 Y U05 indicaron que volverían a participar en un proyecto de estas características, puesto que les había parecido un proyecto interesante. Los dos sujetos restantes indicaron que volverían a participar, aunque les había supuesto un esfuerzo. Para finalizar, todos los sujetos estuvieron de acuerdo en que el mejor paradigma utilizado fue el HSRDCBP.

Solamente dos usuarios propusieron sugerencias para mejorar las aplicaciones. Tanto el usuario U01 como el usuario U04 propusieron reducir el número de iluminaciones de cada carácter. Dado que las aplicaciones desarrolladas necesitan entrenamiento, es necesario un número mínimo de iluminaciones para poder obtener una precisión adecuada en la selección de caracteres. La solución a esto es realizar una sesión de entrenamiento más completa.

Además, el usuario U04 indicó que, en su opinión, son necesarios algunos segundos (entre 5 y 10 segundos) entre caracteres consecutivos. Esto permitiría tener más tiempo para localizar el siguiente carácter, pero reduce la velocidad de la aplicación. Por lo tanto, se produce un compromiso.

3. Comparación con otros estudios

Analizando la forma en la que se iluminan los caracteres de la matriz, solamente el paradigma CBP ilumina la matriz de caracteres de forma aleatoria, dividiendo la matriz principal en dos matrices virtuales.

El resto de los paradigmas utilizan una metodología diferente para iluminar los caracteres de la matriz de selección. En el paradigma HoS [20], se utilizan 6 discos dispuestos en forma circular, cuyo proceso de selección se divide en dos etapas sucesivas. En la primera etapa, cada uno de los 6 discos contiene un quinto de los caracteres. Una vez se ha elegido uno de los 6 discos mostrados en la primera etapa, los caracteres del disco elegido se expanden ocupando los 6 discos para poder selec cionar el carácter deseado.

En el paradigma de las variantes del HoS [21], cada variante utiliza una forma de iluminación. La variante directa del HoS, utilizaba 6 discos de diferentes colores. El *Cake Speller* utiliza 6 triángulos de diferentes colores que se unen formando un hexágono. Por último, el *Center Speller* utilizaba formas geométricas, cada una de ellas con un color diferente. A diferencia de la primera y segunda variantes, las formas se presentaban de forma secuencial en el centro de la pantalla.

En el paradigma RSVP [22], los caracteres se presentan aleatoriamente de uno en uno situados en el centro de la pantalla. Se analizaron tres variantes que diferían con respecto al SOA y al color de las letras. En la condición de Color se utilizaron dos SOA diferentes, 83 ms y 116 ms, para estudiar cómo influye esta característica a la hora de seleccionar un carácter. Los caracteres se mostraban de 5 colores diferentes, rojo, blanco, azul, verde y negro. En la condición de NoColor todos los caracteres eran de color negro.

Por último, en el paradigma HSRD [23], se sustituye la iluminación de un carácter de la matriz por la aparición de un panal de color verde, en cuyo interior se encuentran distribuidos puntos de color rojo.

Si se analiza la disposición de los caracteres en cada uno de los paradigmas, solamente los paradigmas desarrollados, el paradigma RCP, el paradigma CBP y el paradigma HSRD, utilizan una matriz en la que los caracteres se distribuyen en 6 filas y 6 columnas.

En términos de precisión, como se muestra en la Tabla 6-2, la aplicación de selección de caracteres desarrollada utilizando el paradigma CBP consigue una precisión media del 84.667% para los cinco sujetos sanos. Como se muestra en la Tabla 6-1, esta precisión no alcanza la precisión obtenida por el paradigma CBP en el estudio publicado por Townsend *et al.*, 2010 [19].

En cuanto a la forma de iluminación que utiliza el paradigma HSRD, solamente el paradigma tradicional realiza la iluminación de los caracteres como lo hace el paradigma HSRD. En cuanto a la disposición de los caracteres, tanto el paradigma tradicional como el paradigma CBP utilizan la misma disposición que el paradigma HSRD.

En cuestión de precisión, la aplicación de selección de caracteres desarrollada utilizando el paradigma HSRD consigue una precisión media del 98% para los cinco sujetos sanos, por encima de la precisión obtenida por este paradigma en el estudio publicado por Jin *et al.*, 2017. Además, como se puede observar en la Tabla 6-3, esta precisión supera a la precisión obtenida por el resto de estudios analizados anteriormente.

Cabe destacar que, en la comparativa mostrada en la Tabla 6-2, no se está utilizando el mismo número de sujetos y que las sesiones de evaluación se han realizado con distintos parámetros y métodos de procesado.

Para finalizar, el paradigma unificado HSRDCBP utiliza una forma de iluminación fruto de la mezcla de las iluminaciones que realizan los paradigmas CBP y HSRD. Este paradigma ilumina los caracteres de forma aleatoria, sustituyendo en cada iluminación el carácter por un panal de color verde en cuyo interior se encuentran distribuidos de 1 a 3 puntos rojos.

En cuanto a la precisión obtenida, la aplicación desarrollada utilizando el paradigma HSRDCBP consigue una precisión media del 100% para los cinco sujetos sanos. Como se muestra en la Tabla 6-2, la precisión obtenida es la más alta de entre todos los paradigmas.

Existe una diferencia evidente entre las precisiones obtenidas en este trabajo y las precisiones obtenidas en los estudios analizados. Esta diferencia podría deberse al reducido número de sujetos empleado en este trabajo.

4. Limitaciones de los paradigmas desarrollados

La principal limitación de la aplicación desarrollada utilizando el paradigma CBP es el tiempo invertido en seleccionar un carácter. Debido a que la matriz original se divide virtualmente en dos matrices y la secuencia de iluminaciones alterna entre la matriz virtual blanca y la matriz virtual negra, el procesado se ralentiza.

La principal limitación de la aplicación basada en el paradigma HSRD es la iluminación de filas y columnas. A pesar de que en la iluminación el carácter se sustituye por un panal, en cada iluminación, los panales de los caracteres adyacentes pueden provocar la distracción del usuario.

En cuanto al paradigma unificado HSRDCBP, la principal limitación es el tiempo que tarda el usuario en seleccionar un carácter. Al igual que ocurre con el paradigma CBP, esto se debe a que la matriz original se divide virtualmente en dos matrices y la secuencia de iluminaciones alterna entre la matriz virtual blanca y la matriz virtual negra.

Aunque las precisiones obtenidas con ambas aplicaciones son elevadas, se podrían utilizar otros métodos de extracción y traducción de características que permitieran mejorar el rendimiento. En el caso de los métodos de extracción, se podrían utilizar métodos basados en la transformada *Wavelet* o basados en algoritmos evolutivos. En cuanto a los métodos de traducción, se podrían utilizar SVM o un análisis discriminante de *kernel* de regresión espectral (*Spectral Regression Kernel Discriminant Analysis*, SRKDA).

Sin embargo, hay que tener en cuenta que, aunque las precisiones son altas, se han obtenido para un número de sujetos reducido y que los sujetos elegidos para la evaluación eran sujetos sanos. Debido a esto, sería conveniente evaluar ambas aplicaciones utilizando un mayor número de usuarios e incluir en el proceso de evaluación sujetos con alguna discapacidad o enfermedades neurodegenerativas.

TABLA 6-3. Comparativa de precisiones obtenidas con los diferentes paradigmas.

Estudio	Sujetos	Paradigma	Precisión (%)	*p*-valor (1)
Townsend *et al.*, 2010 [19]	18 SC 3 ELA	RCP	77.32 (2)	-
		CBP	91.52 (2)	0.0044**
Treder *et al.*, 2010 [20]	13 SC	HoS	69.00	-
Treder *et al.*, 2011 [21]	13 SC	Variante HoS	91.30	0.0140*
		Cake Speller	88.20	0.0131*
		Center Speller	91.70	0.0049**
Acqualagna *et al.*, 2013 [22]	12 SC	RSVP NoColor 116 ms	94.80	*0.1849*
		RSVP Color 116 ms	94.70	0.0068**
		RSVP Color 83 ms	93.60	0.0181*
Jin *et al.*, 2017 [23]	10 SC	HSRD	89.70	0.0019**
Presente estudio, 2018	5 SC	RCP	92.67	-
		HSRD	98	-
		CBP	84.67	-
		HSRDCBP	**100**	-

SC: Sujetos de Control

ELA: Enfermos con ELA

*: significativo (*p*-valor < 0.05)

**: muy significativo (*p*-valor < 0.01)

(1) Test estadístico comparativo entre HSRDCBP y cada estudio con el test no paramétrico "*Mann-Whitney U Test*".

(2) Precisión obtenida por los sujetos de control

1. Conclusiones

A lo largo del Trabajo Fin de Grado se ha llevado a cabo un estudio sobre los sistemas BCI, describiendo los principales métodos para registrar la actividad cerebral y los tipos de señales de control utilizados en estos sistemas. El EEG ha sido el método de registro elegido para desarrollar las aplicaciones de selección de caracteres debido a que se trata de un método de bajo coste, portátil y no invasivo. Profundizando en este método de registro de la actividad cerebral, se ha estudiado la estructura llevada a cabo para procesar la señal, comenzando con la etapa de adquisición de la señal, seguido de la extracción de características y, para finalizar, la etapa de traducción.

Se ha realizado un estudio sobre los posibles paradigmas *odd-ball* a implementar y se han comparado entre sí para comprobar qué paradigma obtenía mejores resultados. En base a los resultados reflejados en la comparativa, se han elegido dos paradigmas para implementar las aplicaciones de selección de caracteres, el paradigma CBP y el paradigma HSRD. Se han seleccionado estos dos paradigmas por los buenos resultados obtenidos en los estudios analizados y porque, además, reducían el efecto de los dos problemas principales del paradigma tradicional, los errores debidos a distracciones adyacentes y el problema del *doble-flash.*

Posteriormente, se han detallado las características de los potenciales evocados P300 y cómo varía su forma en función de los aspectos psicológicos y farmacológicos, de las enfermedades neuronales de los sujetos o de las diferencias individuales entre ellos. Una vez se ha estudiado la señal de control que se va a utilizar, se han descrito los diferentes métodos de extracción de características para los potenciales P300, considerando un análisis en el dominio espacial y en el dominio tiempo-frecuencia. Dentro del análisis espacial se han descrito el filtrado Laplaciano, los métodos de referencia de media común y el análisis de componentes independientes. Con respecto a los métodos utilizados en el dominio tiempo-frecuencia, se han analizado el promediado sincronizado, el submuestreo, la detección de picos y el cálculo del área, así como la transformada de Fourier de tiempo corto y las transformadas *Wavelet*. A continuación, se han analizado los principales métodos de traducción de características, entre los que destacan el discriminante lineal de Fisher, el análisis discriminante lineal paso-a-paso y las máquinas de soporte vectorial. Los métodos de extracción y traducción de características utilizados en este trabajo son el submuestreo y el algoritmo LDA, respectivamente, ya que han sido los métodos más utilizados para extraer los potenciales evocados P300 en tiempo real [30], [40], [41].

En cuanto al diseño de las aplicaciones de selección de caracteres, primero se han expuesto los diferentes objetivos de cada aplicación. A continuación, se ha descrito cual es la estructura de las aplicaciones y su funcionamiento. Para desarrollar las aplicaciones se ha utilizado la plataforma *OpenViBE* ya que se trata de una plataforma que facilita el desarrollo, la investigación y la monitorización de aplicaciones en tiempo real que requieren adquirir la señal EEG y procesarla.

A través de la plataforma *OpenViBE* y el lenguaje de programación C++, se han desarrollado las diferentes aplicaciones de selección de caracteres. La plataforma *OpenViBE* utiliza un filtro temporal paso-banda y un promediado sincronizado como métodos de extracción de características, y un

algoritmo LDA como método de traducción. Además, se encarga de producir los potenciales evocados P300 necesarios para controlar las aplicaciones mediante el paradigma *odd-ball*, cuyo funcionamiento consiste en presentar el estímulo deseado entre estímulos frecuentes, generándose un potencial evocado P300 cuando el usuario detecta el estímulo deseado. Las aplicaciones desarrolladas utilizan cuatro paradigmas diferentes: el paradigma RCP, el paradigma HSRD, el paradigma CBP y el paradigma HSRDCBP. Debido a que en este trabajo se busca obtener una mayor precisión y mayor velocidad de transmisión, se ha implementado el paradigma RCP para poder comparar los resultados obtenidos con los paradigmas HSRD y CBP. También, se ha decidido unificar los paradigmas HSRD y CBP dando lugar al nuevo paradigma HSRDCBP, debido a los resultados que obtuvieron por separado.

Una vez desarrollada la aplicación, se ha evaluado por parte de cinco sujetos sanos en cuatro sesiones distintas, una por cada paradigma. La disposición de los electrodos se realiza siguiendo la norma internacional 10-20, en la que los electrodos se localizan en las posiciones Fz, Cz, Pz, P3, P4, PO7, PO8 y Oz. Antes de comenzar cada sesión, los usuarios han realizado una fase de entrenamiento para obtener el clasificador LDA de cada uno de ellos, tratando de estimar el vector de pesos adecuado. A continuación, se han realizado las fases de test *online* en las que se han evaluado las aplicaciones desarrolladas. Al finalizar todas las sesiones de evaluación, se ha entregado a cada usuario un cuestionario de satisfacción, donde han reflejado su experiencia y las sugerencias de mejora.

Finalmente, se han expuesto y discutido los resultados obtenidos. A partir de estos resultados, se ha concluido que solo la nueva aplicación que unifica los dos paradigmas estudiados es una alternativa viable al paradigma tradicional. La precisión media obtenida para este paradigma es del 100%. En cuanto al cuestionario de satisfacción, la mayoría de los usuarios opinan que la interfaz del paradigma HSRD es poco intuitiva, al contrario que la interfaz del paradigma CBP; que la selección de comandos con el paradigma HSRD ha sido mucho más sencilla que con el paradigma CBP; y que el mecanismo de adquisición de la señal EEG es poco práctico.

Se concluye que las dos aplicaciones obtienen mejores precisiones en la selección de caracteres que el paradigma tradicional, siendo éste el principal objetivo del desarrollo. A pesar de esto, la aplicación realizada en la que se unifican los dos paradigmas es la que mejores resultados obtiene. Sin embargo, hay que tener en cuenta que los resultados pertenecen a la evaluación únicamente por parte de sujetos sanos.

A continuación, se resumen las principales conclusiones extraídas:

1. Selección del EEG como método de registro y los potenciales evocados P300 como señal de control. El EEG se ha utilizado debido a que es un método de bajo coste, portátil y no invasivo. Los potenciales P300 se han utilizado como señal de control ya que se trata de una señal exógena y permite selecciones múltiples utilizando el paradigma *odd-ball*.

2. Diseño y desarrollo de aplicaciones de selección de caracteres utilizando la plataforma *OpenViBE*. Tras realizar una comparativa se han seleccionado dos paradigmas para implementar las aplicaciones de selección de caracteres, el paradigma CBP y el paradigma HSRD. Se han seleccionado estos dos paradigmas por los buenos resultados

obtenidos en los estudios analizados y porque, además, reducían el efecto de los dos problemas principales del paradigma tradicional, los errores debidos a distracciones adyacentes y el problema del *doble-flash.* Después de seleccionar los paradigmas, se han implementado las aplicaciones utilizando la plataforma *OpenViBE*

3. Selección del submuestreo de la señal EEG como método de extracción de características y el algoritmo LDA como método de clasificación. Atendiendo al elevado uso en estudios anteriores, se ha utilizado el submuestreo de la señal EEG en crudo para extraer las características y el algoritmo LDA como algoritmo de traducción de estas.

4. Evaluación de las aplicaciones de selección de caracteres utilizando cinco sujetos sanos. Se han evaluado las cuatro aplicaciones desarrolladas utilizando el paradigma tradicional (RCP), el paradigma CBP, el paradigma HSRD y el paradigma unificado HSRDCBP, con cinco sujetos sanos en cuatro sesiones consecutivas, una sesión por cada paradigma *odd-ball*. Cada sesión se dividía en una fase de entrenamiento, en la que los usuarios debían escribir una única palabra; y una fase de test online, en la que debían escribir cinco nuevas palabras. Adicionalmente, se ha entrega un cuestionario de satisfacción a cada sujeto, con el fin de recoger su experiencia y las sugerencias de mejora.

5. Discusión de los resultados y comparación con otros estudios. Se han discutido los resultados, declarando el buen funcionamiento general de las aplicaciones desarrolladas. Analizando las precisiones, el paradigma tradicional obtiene un 92.67%, el paradigma CBP un 84.67%, el paradigma HSRD un 98% y el nuevo paradigma unificado HSRDCBP obtiene un 100% de precisión. A partir de esto se concluye que la aplicación que utiliza el paradigma HSRDCBP es que mejores resultados obtiene.

2. *Líneas futuras*

Atendiendo al procesado de la señal EEG, se podrían implementar técnicas de procesado diferentes, tanto para realizar la extracción de características como para su posterior traducción, teniendo en cuenta que la aplicación debe trabajar en tiempo real y comparar los resultados para determinar qué técnica es más adecuada.

En cuanto a la aplicación desarrollada utilizando el paradigma HSRD, se podría estudiar la sustitución de los panales por otras imágenes. Esto podría ayudar al usuario a mantener la concentración.

Como se ha explicado en capítulos anteriores, se han unificado los dos paradigmas estudiados en una única aplicación a la que se ha denominado HSRDCBP. Una posible línea futura es realizar un estudio más detallado de esta aplicación.

A pesar de que se ha realizado una comparativa con 4 paradigmas, en otra línea futura se podría realizar un estudio que incluyera el paradigma HoS o el *Cake Speller*, estudiados en el capítulo 2. En este caso, se podría tener en cuenta el mismo número de usuarios para la evaluación y los mismos métodos de procesado.

En el cuestionario de satisfacción, uno de los usuarios sugirió que sería útil realizar un descanso entre palabras. Por ello, podría introducirse una pausa de 5 segundos con el fin de que el usuario descanse la vista y aumente el nivel de atención en la siguiente palabra.

Para finalizar, una posible línea futura útil para ambas aplicaciones puede ser la evaluación de estas por parte de usuarios con grave discapacidad o enfermedades neurodegenerativas, ya que es la finalidad de la aplicación desarrollada y, de esta forma, podrían obtenerse nuevas propuestas de mejora.

[1] A. Nowakowski, "Quantitative Active Dynamic Thermal IR-Imaging and Thermal Tomography in Medical Diagnostics," in *The Biomedical Engineering Handbook. Medical Devices and Systems*, Taylor & Francis Group, 2006, p. 22.1-22.30.

[2] J. R. Wolpaw *et al.*, "Brain-Computer Interface technology: a review of the first international meeting.," *IEEE Trans. Rehabil. Eng.*, vol. 8, no. 2, pp. 164–173, 2000.

[3] J. J. Vidal, "Real-time detection of brain events in EEG," *Proc. IEEE*, vol. 65, no. 5, pp. 633–641, 1977.

[4] J. S. Kumar and P. Bhuvaneswari, "Analysis of electroencephalography (EEG) signals and its categorization - A study," *Procedia Eng.*, vol. 38, pp. 2525–2536, 2012.

[5] 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.

[6] J. R. Wolpaw *et al.*, "BCI Meeting 2005 - Workshop on signals and recording methods," *IEEE Trans. Rehabil. Eng.*, vol. 14, no. 2, pp. 138–141, 2006.

[7] O. Friman, I. Volosyak, and A. Gräser, "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.

[8] M. Middendorf, G. McMillan, G. Calhoun, and K. S. Jones, "Brain-computer interfaces based on the steady-state visual-evoked response," *IEEE Trans. Rehabil. Eng.*, vol. 8, no. 2, pp. 211–214, 2000.

[9] T. Hinterberger, N. Weiskopf, R. Veit, B. Wilhelm, E. Betta, and N. Birbaumer, "An EEG-driven Brain-Computer Interface combined with functional Magnetic Resonance Imaging (fMRI)," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 971–974, 2004.

[10] G. Pfurtscheller and C. Neuper, "Motor imagery and direct brain- computer communication," *Proc. IEEE*, vol. 89, no. 7, pp. 1123–1134, 2001.

[11] J. A. Pineda, B. Z. Allison, and A. Vankov, "The effects of self-movement, observation, and imagination on μ rhythms and readiness potentials (RP'S): Toward a brain-computer interface (BCI)," *IEEE Trans. Rehabil. Eng.*, vol. 8, no. 2, pp. 219–222, 2000.

[12] L. Sörnmo and P. Laguna, "Bioelectrical Signal Processing in Cardiac and Neurological Applications," *1st ed. Sweden Acad. Press*, pp. 1–24, 2005.

[13] D. J. McFarland, C. W. Anderson, K. R. Müller, A. Schlögl, and D. J. Krusienski, "BCI Meeting 2005 - Workshop on BCI signal processing: Feature extraction and translation," *IEEE Trans. Rehabil. Eng.*, vol. 14, no. 2, pp. 135–138, 2006.

[14] K.-R. Müller, C. W. Anderson, and G. E. Birch, "Linear and nonlinear methods for Brain-Computer Interfaces.," *IEEE Trans. Rehabil. Eng.*, vol. 11, no. 2, pp. 165–169, 2003.

[15] A. Akhtar, J. J. S. Norton, M. Kasraie, and T. Bretl, "Playing checkers with your mind: An interactive multiplayer hardware game platform for brain-computer interfaces," *2014 36th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBC 2014*, pp. 1650–1653, 2014.

[16] M. Bensch *et al.*, "Nessi: An EEG-controlled web browser for severely paralyzed patients," *Comput. Intell. Neurosci.*, vol. 2007, pp. 1–5, 2007.

[17] J. Pineda, D. Silverman, and A. Vankov, "Learning to control brain rhythms: making a brain-computer interface possible," *IEEE Trans.*, vol. 11, no. 2, pp. 181–184, 2003.

[18] L. A. Farwell and E. Donchin, "Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials," *Electroencephalogr. Clin. Neurophysiol.*, vol. 70, no. 6, pp. 510–523, 1988.

[19] G. Townsend *et al.*, "A novel P300-based brain-computer interface stimulus presentation paradigm: Moving beyond rows and columns," *Clin. Neurophysiol.*, vol. 121, no. 7, pp. 1109–1120, 2010.

[20] M. S. Treder and B. Blankertz, "(C)overt attention and visual speller design in an ERP-based brain-computer interface," *Behav. Brain Funct.*, vol. 6, pp. 1–13, 2010.

[21] M. S. Treder, N. M. Schmidt, and B. Blankertz, "Gaze-independent brain-computer interfaces based on covert attention and feature attention," *J. Neural Eng.*, vol. 8, no. 6, pp. 1–6, 2011.

[22] L. Acqualagna and B. Blankertz, "Gaze-independent BCI-spelling using rapid serial visual presentation (RSVP)," *Clin. Neurophysiol.*, vol. 124, no. 5, pp. 901–908, 2013.

[23] J. Jin, H. Zhang, I. Daly, X. Wang, and A. Cichocki, "An improved P300 pattern in BCI to catch user's attention," *J. Neural Eng.*, vol. 14, no. 3, pp. 1–11, 2017.

[24] T. W. Picton, "The P300 wave of the human event-related potential," *J. Clin. Neurophysiol.*, vol. 9, no. 4, pp. 456–479, 1992.

[25] R. M. Rangayyan, "Filtering for Removal Artifacts," in *Biomedical Signal Analysis: A Case-Study Approach*, Canada: IEEE Press & Wiley, 2002, pp. 73–176.

[26] D. J. McFarland, L. M. McCane, S. V. David, and J. R. Wolpaw, "Spatial filter selection for EEG-based communication," *Electroencephalogr. Clin. Neurophysiol.*, vol. 103, no. 3, pp. 386–394, 1997.

[27] K. Li, R. Sankar, Y. Arberl, and E. Donchin, "Single trial independent component analysis for P300 BCI system," *Eng. Med. Biol. Soc. EMBC 2009. Annu. Int. Conf. IEEE*, pp. 4035–4038, 2009.

[28] E. Donchin, K. M. Spencer, and R. Wijesinghe, "The Mental Prosthesis: Assessing the speed of a P300-based Brain-Computer Interface," *IEEE Trans. Rehab. Eng*, vol. 8, no. 2, pp. 174–179, 2000.

[29] M. Kaur, P. Ahmed, and M. Q. Rafiq, "Analysis of extracting distinct functional components of P300 using Wavelet Transform," *Proc. 4th MMES 2nd ICDCC*, pp. 57–62, 2013.

[30] D. J. Krusienski *et al.*, "A comparison of classification techniques for the P300 Speller," *J. Neural Eng.*, vol. 3, no. 4, pp. 299–305, 2006.

[31] L. Mayaud *et al.*, “A comparison of recording modalities of P300 event-related potentials (ERP) for brain-computer interface (BCI) paradigm,” *Neurophysiol. Clin.*, vol. 43, no. 4, pp. 217–227, 2013.

[32] N. R. Draper and H. Smith, “Selecting the Best Regression Equation,” in *Applied Regression Analysis, 3rd ed*, New York: John Wiley & Sons, 1981, pp. 335–339.

[33] N. V. Manyakov, N. Chumerin, A. Combaz, and M. M. Van Hulle, “Comparison of classification methods for P300 brain-computer interface on disabled subjects,” *Comput. Intell. Neurosci.*, vol. 2011, no. 2, pp. 1–12, 2011.

[34] M. Kaper, P. Meinicke, U. Grossekathoefer, T. Lingner, and H. Ritter, “BCI competition 2003 - Data set IIb: Support vector machines for the P300 speller paradigm,” *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 1073–1076, 2004.

[35] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, and B. Arnaldi, “A review of classification algorithms for EEG-based Brain-Computer Interfaces,” *J. Neural Eng.*, vol. 4, no. 2, pp. R1–R13, 2007.

[36] “Box Algorithm Classifier Trainer,” *OpenVibe*. [Online]. Available: http://openvibe.inria.fr/documentation/1.0.1/Doc_BoxAlgorithm_ClassifierTrainer.html. [Accessed: 24-Aug-2018].

[37] “Acquisition Server | OpenViBE,” *OpenVibe*, 2011. [Online]. Available: http://openvibe.inria.fr/acquisition-server/. [Accessed: 24-Aug-2018].

[38] “P300: Basic P300 Speller demo | OpenViBE,” *OpenVibe*, 2011. [Online]. Available: http://openvibe.inria.fr/openvibe-p300-speller/. [Accessed: 24-Aug-2018].

[39] “Box Algorithms List,” *OpenVibe*. [Online]. Available: http://openvibe.inria.fr/documentation/2.1.0/Doc_BoxAlgorithms.html. [Accessed: 24-Aug-2018].

[40] V. Martínez-Cagigal, J. Gomez-Pilar, D. Álvarez, and R. Hornero, “An Asynchronous P300-Based Brain-Computer Interface Web Browser for Severely Disabled People,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 25, no. 8, pp. 1332–1342, 2017.

[41] D. J. Krusienski, E. W. Sellers, D. J. McFarland, T. M. Vaughan, and J. R. Wolpaw, “Toward enhanced P300 speller performance,” *J. Neurosci. Methods*, vol. 167, no. 1, pp. 15–21, 2008.

Anexo A

ÍNDICE DE ACRÓNIMOS

En este anexo, se recogen todos los acrónimos utilizados a lo largo de este Trabajo Fin de Grado por orden alfabético con una breve descripción en inglés y español.

ANN	*Artificial Neural Networks*
	Redes Neuronales Artificiales
AR	*Autoregressive Models*
	Modelos Autoregresivos
BCI	*Brain Computer Interface*
	Interfaz Cerebro-Computadora
CAR	*Common Average Reference*
	Referencia de Media Común
CBP	*Checkerboard Paradigm*
	Paradigma de Tablero de Ajedrez
CSP	*Common Spatial Patterns*
	Patrones Espaciales Comunes
CWT	*Continuous Wavelet Transform*
	Transformada Wavelet Continua
DWT	*Discrete Wavelet Transform*
	Transformada Wavelet Discreta
EEG	*Electroencefalography*
	Electroencefalografía
ECG	Electrocardiography
	Electrocardiografía
ECoG	*Electrocorticography*
	Electrocorticografía
ELA	*Amyotrophic Lateral Sclerosis*
	Esclerosis Lateral Amiotrófica
EMG	Electromyography
	Electromiografía
EOG	Electrooculography
	Electrooculografía

ERD	*Event-Related Desynchronization*
	Desincronización Relacionada con Eventos
ERS	*Event-Related Synchronization*
	Sincronización Relacionada con Eventos
FLD	*Fisher's Linear Discriminant*
	Discriminante Lineal de Fisher
fMRI	*Functional Magnetic Resonance Imaging*
	Imagen por Resonancia Magnética Funcional
HMM	*Hidden Markov Model*
	Modelo Oculto de Markov
HoS	*Hex-o-Spell Paradigm*
	Paradigma del Hexágono
HSRD	*Honeycomb-Shaped Red Dots*
	Paradigma de Puntos Rojos en forma de Panal
ICA	*Independent Component Analysis*
	Análisis de Componentes Independientes
LDA	*Linear Discriminant Analysis*
	Análisis Discriminante Lineal
LMS	*Least Mean Squares*
	Mínimos Cuadrados Promediados
MEG	*Magnetoencephalography*
	Magnetoencefalografía
MLP	*Multi-Lateral Perceptron*
	Perceptrón Multicapa
MSE	*Mean Square Error*
	Erros Cuadrático Medio
PCA	*Principal Component Analysis*
	Análisis de Componentes Principales
PET	*Positron Emission Tomography*
	Tomografía por Emisión de Positrones
PLS	*Partial Least Squares*
	Mínimos Cuadrados Parciales
QDA	*Quadratic Discriminant Analysis*
	Análisis Discriminante Cuadrático

RLS	*Recursive Least Squares*
	Mínimos Cuadrados Recursivos
RSVP	*Rapid Serial Visual Paradigm*
	Paradigma Visual en Serie Rápida
RT	*Reaction Time*
	Tiempo de Reacción
SCP	*Slow Cortical Potentials*
	Potenciales Corticales Lentos
SRKDA	*Spectral Regression Kernel Discriminant Analysis*
	Análisis Discriminante de Kernel de Regresión Espectral
SOA	*Stimulus Onset Asynchrony*
	Asincronía de Comienzo de Estímulo
STFT	*Short-Time Fourier Transform*
	Transformada de Fourier de Tiempo Corto
SVM	*Support Vector Machines*
	Máquinas de Soporte Vectorial
SW	*Slow Wave*
	Onda Lenta Positiva
SWLDA	*StepWise Linear Discriminant Analysis*
	Análisis Discriminante Lineal Paso-a-paso
VEP	Visual Evoked Potentials
	Potenciales Evocados Visuales
WP	*Wavelet Packets*
	Paquetes Wavelet

GUÍA DE USUARIO

En este anexo, se recogen las guías de usuario realizadas para ayudar a los sujetos a entender el funcionamiento de las diferentes aplicaciones desarrolladas.

1. Row-Column Paradigm

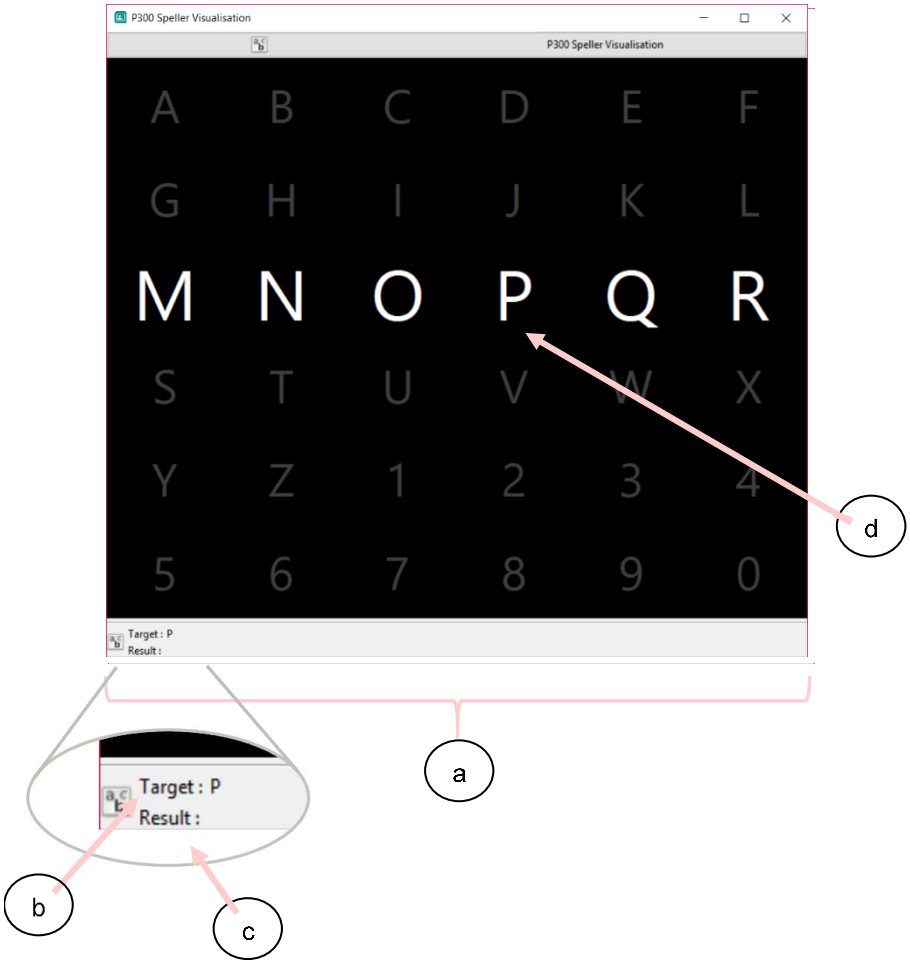
GUÍA DE USUARIO

APLICACIÓN QUE UTILIZA EL PARADIGMA RCP

Una vez tenga colocado el gorro con los electrodos y se haya aplicado el gel estará preparado/a para escribir utilizando la aplicación de selección de caracteres. Es importante que intente no mover la cabeza o cualquier extremidad para no dificultar la selección de los caracteres. Además, debe evitar hablar, estornudar y mover la mandíbula.

I. Estructura de la aplicación

La aplicación consta de una matriz de selección formada por 6 filas y 6 columnas con los caracteres alfanuméricos que deberá seleccionar para escribir. En la parte inferior de la interfaz se muestra un



a. Matriz de selección de caracteres
b. Indicador con el comando que debe seleccionar
c. Indicador con el comando seleccionado
d. Caracteres iluminados

recuadro con dos etiquetas, *target* y *result*. En la etiqueta *target*, situada en la parte superior del recuadro, irán apareciendo los caracteres que debe seleccionar. En la etiqueta *result*, situada en la parte inferior, aparecerán los caracteres que finalmente ha seleccionado.

II. Selección de caracteres

Al abrir la aplicación, aparecerá el carácter que debe seleccionar rodeado por un recuadro de color azul. Para seleccionar el carácter solamente tiene que mirar fijamente el icono. Mientras tanto, se iluminarán las filas y columnas de la matriz de forma aleatoria. Una recomendación para evitar perder la concentración y asegurarse de que fija usted la vista de forma correcta es contar el número de veces que se ilumina el carácter que desea seleccionar. Después de que el icono se haya iluminado un determinado número de veces, aparecerá el carácter que ha seleccionado rodeado por un recuadro de color verde. En la parte inferior de la interfaz, podrá consultar cual ha sido el resultado de su selección y si coincide con el carácter que debía seleccionar. Los caracteres seleccionados se muestran en 3 colores diferentes.

- Verde. El carácter seleccionado coincide con el carácter que debía seleccionar.
- Naranja. El carácter seleccionado no coincide con el carácter que debía seleccionar. El carácter aparece en este color debido a que se ha seleccionado un carácter adyacente al que se debía seleccionar.
- Rojo. El carácter seleccionado no coincide con el carácter que debía seleccionar.

2. Checkerboard Paradigm

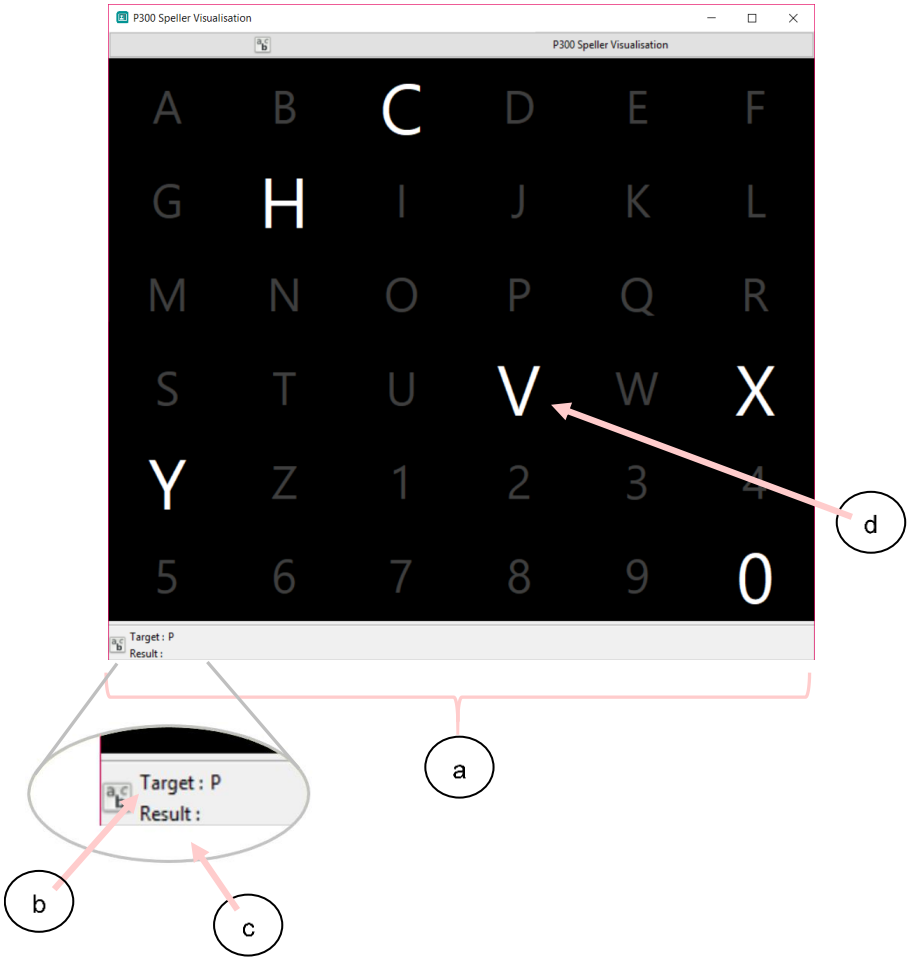
GUÍA DE USUARIO

APLICACIÓN QUE UTILIZA EL PARADIGMA CBP

Una vez tenga colocado el gorro con los electrodos y se haya aplicado el gel estará preparado/a para escribir utilizando la aplicación de selección de caracteres. Es importante que intente no mover la cabeza o cualquier extremidad para no dificultar la selección de los caracteres. Además, debe evitar hablar, estornudar y mover la mandíbula.

I. Estructura de la aplicación

La aplicación consta de una matriz de selección formada por 6 filas y 6 columnas con los caracteres alfanuméricos que deberá seleccionar para escribir. En la parte inferior de la interfaz se muestra un



a. Matriz de selección de caracteres
b. Indicador con el comando que debe seleccionar
c. Indicador con el comando seleccionado
d. Caracteres iluminados

recuadro con dos etiquetas, *target* y *result*. En la etiqueta *target*, situada en la parte superior del recuadro, irán apareciendo los caracteres que debe seleccionar. En la etiqueta *result*, situada en la parte inferior, aparecerán los caracteres que finalmente ha seleccionado.

II. Selección de caracteres

Al abrir la aplicación, aparecerá el carácter que debe seleccionar rodeado por un recuadro de color azul. Para seleccionar el carácter solamente tiene que mirar fijamente el icono. Mientras tanto, se iluminarán los caracteres de la matriz de forma aleatoria. Una recomendación para evitar perder la concentración y asegurarse de que fija usted la vista de forma correcta es contar el número de veces que se ilumina el carácter que desea seleccionar. Después de que el icono se haya iluminado un determinado número de veces, aparecerá el carácter que ha seleccionado rodeado por un recuadro de color verde. En la parte inferior de la interfaz, podrá consultar cual ha sido el resultado de su selección y si coincide con el carácter que debía seleccionar. Los caracteres seleccionados se muestran en 3 colores diferentes.

- Verde. El carácter seleccionado coincide con el carácter que debía seleccionar.
- Naranja. El carácter seleccionado no coincide con el carácter que debía seleccionar. El carácter aparece en este color debido a que se ha seleccionado un carácter adyacente al que se debía seleccionar.
- Rojo. El carácter seleccionado no coincide con el carácter que debía seleccionar.

3. Honeycomb-Shaped Red Dots Paradigm

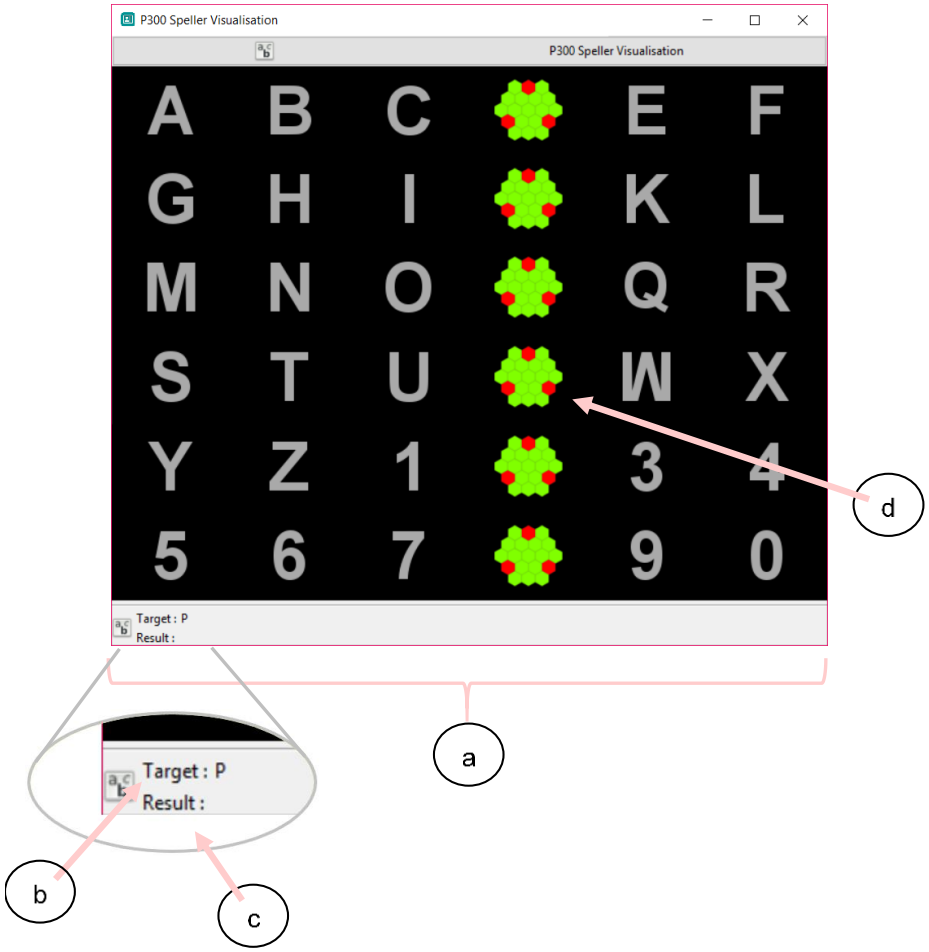
GUÍA DE USUARIO

APLICACIÓN QUE UTILIZA EL PARADIGMA HSRD

Una vez tenga colocado el gorro con los electrodos y se haya aplicado el gel estará preparado/a para escribir utilizando la aplicación de selección de caracteres. Es importante que intente no mover la cabeza o cualquier extremidad para no dificultar la selección de los caracteres. Además, debe evitar hablar, estornudar y mover la mandíbula.

I. Estructura de la aplicación

La aplicación consta de una matriz de selección formada por 6 filas y 6 columnas con los caracteres alfanuméricos que deberá seleccionar para escribir. En la parte inferior de la interfaz se muestra un



a. Matriz de selección de caracteres
b. Indicador con el comando que debe seleccionar
c. Indicador con el comando seleccionado
d. Caracteres iluminados

recuadro con dos etiquetas, *target* y *result*. En la etiqueta *target*, situada en la parte superior del recuadro, irán apareciendo los caracteres que debe seleccionar. En la etiqueta *result*, situada en la parte inferior, aparecerán los caracteres que finalmente ha seleccionado.

II. Selección de caracteres

Al abrir la aplicación, aparecerá el carácter que debe seleccionar rodeado por un recuadro de color azul. Para seleccionar el carácter solamente tiene que mirar fijamente el icono. Mientras tanto, se iluminarán las filas y columnas de la matriz de forma aleatoria. En cada iluminación, el carácter se sustituirá por un panal de color verde, en cuyo interior aparecerán distribuidos de 1 a 3 puntos de color rojo. Una recomendación para evitar perder la concentración y asegurarse de que fija usted la vista de forma correcta es contar el número de puntos que aparecen dentro del panal que sustituye al carácter que debe seleccionar. Después de que el icono se haya iluminado un determinado número de veces, aparecerá el carácter que ha seleccionado rodeado por un recuadro de color verde. En la parte inferior de la interfaz, podrá consultar cual ha sido el resultado de su selección y si coincide con el carácter que debía seleccionar. Los caracteres seleccionados se muestran en 3 colores diferentes.

- Verde. El carácter seleccionado coincide con el carácter que debía seleccionar.
- Naranja. El carácter seleccionado no coincide con el carácter que debía seleccionar. El carácter aparece en este color debido a que se ha seleccionado un carácter adyacente al que se debía seleccionar.
- Rojo. El carácter seleccionado no coincide con el carácter que debía seleccionar.

4. Honeycomb-Shaped Red Dots – Checkerboard Paradigm

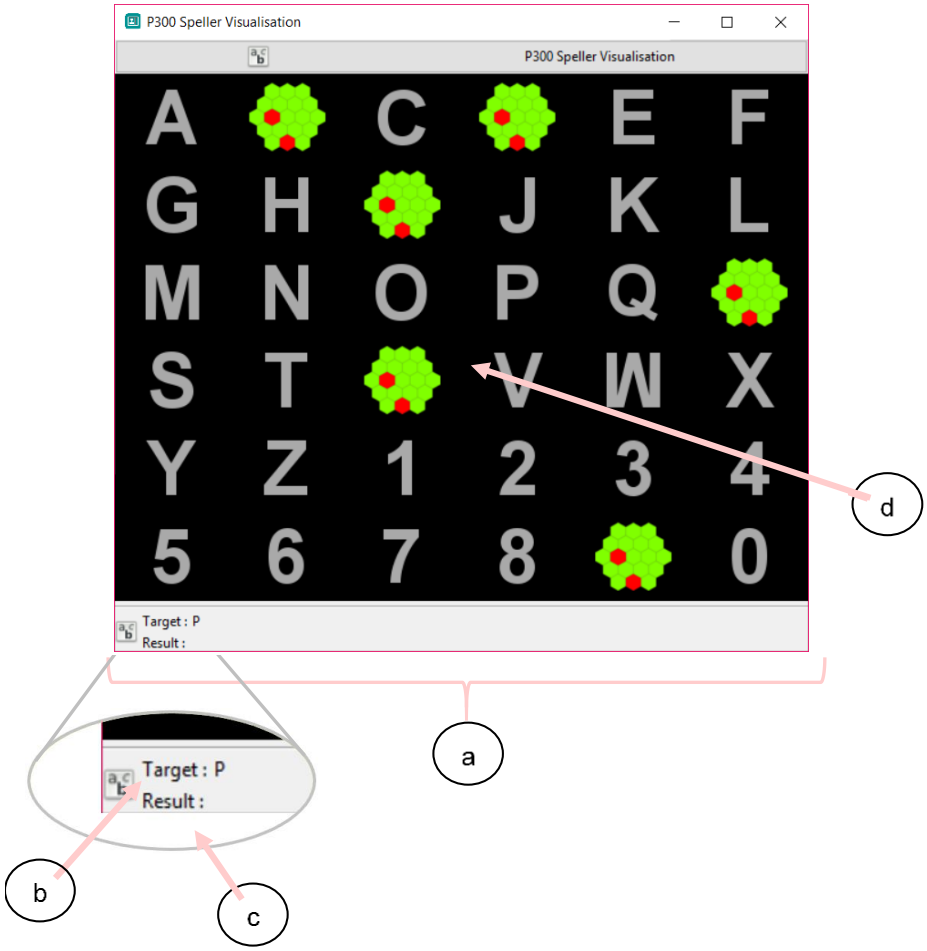
GUÍA DE USUARIO

APLICACIÓN QUE UTILIZA EL PARADIGMA HSRDCBP

Una vez tenga colocado el gorro con los electrodos y se haya aplicado el gel estará preparado/a para escribir utilizando la aplicación de selección de caracteres. Es importante que intente no mover la cabeza o cualquier extremidad para no dificultar la selección de los caracteres. Además, debe evitar hablar, estornudar y mover la mandíbula.

I. Estructura de la aplicación

La aplicación consta de una matriz de selección formada por 6 filas y 6 columnas con los caracteres alfanuméricos que deberá seleccionar para escribir. En la parte inferior de la interfaz se muestra un



a. Matriz de selección de caracteres
b. Indicador con el comando que debe seleccionar
c. Indicador con el comando seleccionado
d. Caracteres iluminados

recuadro con dos etiquetas, *target* y *result*. En la etiqueta *target*, situada en la parte superior del recuadro, irán apareciendo los caracteres que debe seleccionar. En la etiqueta *result*, situada en la parte inferior, aparecerán los caracteres que finalmente ha seleccionado.

II. Selección de caracteres

Al abrir la aplicación, aparecerá el carácter que debe seleccionar rodeado por un recuadro de color azul. Para seleccionar el carácter solamente tiene que mirar fijamente el icono. Mientras tanto, se iluminarán los caracteres de la matriz de forma aleatoria. En cada iluminación, el carácter se sustituirá por un panal de color verde, en cuyo interior aparecerán distribuidos de 1 a 3 puntos de color rojo. Una recomendación para evitar perder la concentración y asegurarse de que fija usted la vista de forma correcta es contar el número de puntos que aparecen dentro del panal que sustituye al carácter que debe seleccionar. Después de que el icono se haya iluminado un determinado número de veces, aparecerá el carácter que ha seleccionado rodeado por un recuadro de color verde. En la parte inferior de la interfaz, podrá consultar cual ha sido el resultado de su selección y si coincide con el carácter que debía seleccionar. Los caracteres seleccionados se muestran en 3 colores diferentes.

- Verde. El carácter seleccionado coincide con el carácter que debía seleccionar.
- Naranja. El carácter seleccionado no coincide con el carácter que debía seleccionar. El carácter aparece en este color debido a que se ha seleccionado un carácter adyacente al que se debía seleccionar.
- Rojo. El carácter seleccionado no coincide con el carácter que debía seleccionar.

CUESTIONARIO DE SATISFACCIÓN

En este anexo, se recoge el cuestionario de satisfacción entregado a los usuarios para evaluar las diferentes aplicaciones desarrolladas.

CUESTIONARIO DE SATISFACCIÓN

ESTUDIO DE PARADIGMAS ODD-BALL

Valore las siguientes afirmaciones sobre la aplicación BCI de selección de caracteres, siendo 1 poco satisfactorio y 5 muy satisfactorio.

1. La interfaz de la aplicación es intuitiva y fácil de entender.

	1	**2**	**3**	**4**	**5**
RCP					
HSRD					
CBP					
HSRDCBP					

2. Me ha resultado sencillo seleccionar las letras de la matriz.

	1	**2**	**3**	**4**	**5**
RCP					
HSRD					
CBP					
HSRDCBP					

3. La iluminación de la letra resulta molesta.

	1	**2**	**3**	**4**	**5**
RCP					
HSRD					
CBP					
HSRDCBP					

4. El tiempo necesario para escribir una letra es elevado.

1	**2**	**3**	**4**	**5**

5. Habitualmente se seleccionan letras que no deseo seleccionar.

1	2	3	4	5

6. La utilización del gorro y los electrodos es poco práctica.

1	2	3	4	5

7. La duración de las sesiones realizadas es adecuada.

1	2	3	4	5

8. ¿Volvería a participar en proyectos con estas características?
    a. Si, me ha parecido un proyecto interesante.
    b. Si, aunque me ha supuesto un esfuerzo.
    c. No. Indique la razón a continuación:

    ……..

9. En su opinión, ¿qué debería mejorarse de las diferentes aplicaciones para futuras versiones?

    ……..

10. En su opinión, ¿qué paradigma es mejor?

    ……..

CODIGO GENERADO EN C++

En este anexo, se recoge el código correspondiente a las aplicaciones desarrolladas a lo largo de este Trabajo Fin de Grado. Debido al gran número de líneas de código que incluyen los archivos, solamente se han incluido las funciones que se han modificado para el buen funcionamiento de las aplicaciones y las nuevas funciones creadas.

1. Checkerboard Paradigm

```
/*-------------------------------------------------------------------------------------------------------------*/
/*                                P300 SPELLER - Checkerboard Paradigm                                         */
/*-------------------------------------------------------------------------------------------------------------*/
/* Aplicación que permite al usuario escribir caracteres utilizando solamente las ondas cerebrales.           */
/* Para ello se utiliza el paradigma CBP.                                                                      */
/*                                                                                          Fecha: 25/06/2018 */
/*                                                                        Cristina Cantalapiedra Cabezas       */
/*-------------------------------------------------------------------------------------------------------------*/

/*************************************************************************************************************/
/** Método process(void) -Speller Stimulator-                                                               **/
/** Se encarga de producir la estimulación necesaria para que el usuario pueda escribir el carácter que    **/
/** desea.                                                                                                  **/
/*************************************************************************************************************/
       boolean CBoxAlgorithmP300SpellerStimulator:: process(void) {

              IBoxIO& l_rDynamicBoxContext = this->getDynamicBoxContext();

              uint32 l_ui32State = State_NoFlash;
              uint64 l_ui64CurrentTime = this->getPlayerContext().getCurrentTime();
              uint64 l_ui64FlashIndex = (uint64) - 1;

              CStimulationSet l_oStimulationSet;

              if(m_bStartReceived) {

                     if (l_ui64CurrentTime < m_ui64TrialStartTime) {

                            l_ui32State = State_TrialRest;
                     }
                     else {
                            if ((m_ui64TrialIndex > m_ui64TrialCount) && (m_ui64TrialCount > 0)) {

                                   l_ui32State = State_ExperimentStop;
                            }
                            else {
                                   uint64 l_ui64CurrentTimeInTrial = l_ui64CurrentTime - m_ui64TrialStartTime;

                                   uint64 l_ui64CurrentTimeInRepetition = l_ui64CurrentTimeInTrial % (m_ui64RepetitionDuration +
                                   m_ui64InterRepetitionDuration);
                                   uint64 l_ui64FlashIndexInRepetition = l_ui64CurrentTimeInRepetition / (m_ui64FlashDuration +
                                   m_ui64NoFlashDuration);
```

```
                l_ui64FlashIndex = l_ui64FlashIndexInRepetition;

                if (l_ui64CurrentTimeInTrial >= m_ui64TrialDuration) {

                        if (m_ui64TrialCount == 0 || m_ui64TrialIndex <= m_ui64TrialCount) {

                                m_ui64TrialStartTime = l_ui64CurrentTime + m_ui64InterTrialDuration;

                                l_ui32State = State_TrialRest;
                                l_ui64FlashIndex = (uint64) - 1;
                                m_ui64TrialIndex++;
                        }
                        else {
                                m_ui64TrialStartTime = l_ui64CurrentTime + m_ui64InterTrialDuration;
                                l_ui32State = State_None;
                        }
                }
                else {
                        if (l_ui64CurrentTimeInRepetition >= m_ui64RepetitionDuration) {

                                l_ui32State = State_RepetitionRest;
                                l_ui64FlashIndex = (uint64) - 1;
                        }
                        else {
                                if (l_ui64CurrentTimeInRepetition % (m_ui64FlashDuration + m_ui64NoFlashDuration) <
                                m_ui64FlashDuration) {

                                        l_ui32State = State_Flash;
                                }
                                else {
                                        l_ui32State = State_NoFlash;
                                }
                        }
                }
        }
}

if (l_ui32State != m_ui32LastState) {

        // Cuando l_bRow es true, se ilumina una fila. Cuando l_bColumn es false, se ilumina una columna
        boolean l_bRow = ((l_ui64FlashIndex & 1) == 1 ? true : false);

        // Declaración de variables
        long l_iRow;
```

```
long l_iColumn;

// Cuando haya una iluminación o no iluminación, entramos en el if. De esta forma, eliminamos el primer 0
que aparece en el vector
if (l_ui32State == State_Flash || l_ui32State == State_NoFlash) {

	// Si el contador es menor que 4, es decir, si aún no se han hecho 4 iluminaciones de filas, entra
	en el if
	if (cont < 4) {

		// Si m_bRow es true, es decir, si se tiene que iluminar una fila, entra en el if
		if (m_bRow) {

			// La primera fila que se va a iluminar se obtiene del primer elemento del vector
			m_vRow, que contiene la secuencia de iluminaciones. La primera columna no se
			ilumina, por lo que se le da el valor -1
			l_iRow = (long) m_vRow[0];
			l_iColumn = -1;

			// Si el contador es 3, es decir, ya se han iluminado 4 filas, cambiamos m_bRow a
			false, para que pasen a iluminarse las columnas
			if (cont == 3) {

				m_bRow = !m_bRow;
			}

			if (l_ui32State == State_NoFlash) {

				// Guardo en un vector auxiliar el vector m_vRow
				std::map < OpenViBE::uint64, OpenViBE::uint64 > l_vRowAux = m_vRow;
				int sizeR = m_vRow.size();

				m_vRow.clear();

				// Recorro el vector auxiliar, asignando cada posición a una posición menor
				del vector m_vRow. De esta forma, elimino el primer elemento del vector m_vRow
				for (int i = 0; i < (sizeR - 1); i++) {
					m_vRow[i] = l_vRowAux[i + 1];
				}
			}
		}

		// Si m_bRow es false, es decir, si se tiene que iluminar una columna, entra en el else
		else {
```

```
// La primera columna que se va a iluminar se obtiene del primer elemento del vector
m_vColumn, que contiene la secuencia de iluminaciones. La primera fila no se ilumina,
por lo que se le da el valor -1
l_iRow = -1;
l_iColumn = (long) m_vColumn[0];

// Si tenemos 12 columnas, entra en el if
if (m_ui64ColumnCount == 12) {

	// Si el contador es 3, es decir, ya se han iluminado 4 columnas, cambiamos
	m_bRow a true, para que pasen a iluminarse las filas
	if (cont == 3) {

		m_bRow = !m_bRow;
	}
}

// Si tenemos 6 columnas, entra en el else
else {
	// Si el contador es 1, es decir, ya se ha iluminado 1 columna, cambiamos
	m_bRow a true, para que pasen a iluminarse las filas. Se pone el contador a 3
	para que detecte que ya se han iluminado las columnas y pase a iluminar las
	filas
	if (cont == 1) {

		m_bRow = !m_bRow;
		cont = 3;
	}
}

if (l_ui32State == State_NoFlash) {

	// Guardo en un vector auxiliar el vector m_vColumn
	std::map < OpenViBE::uint64, OpenViBE::uint64 > l_vColAux = m_vColumn;
	int sizeC = m_vColumn.size();

	m_vColumn.clear();

	// Recorro el vector auxiliar, asignando cada posición a una posición menor
	del vector m_vColumn. De esta forma, elimino el primer elemento del vector
	m_vColumn
	for (int i = 0; i < (sizeC - 1); i++) {
		m_vColumn[i] = l_vColAux[i + 1];
	}
}
```

```
                }
                cont++;
        }

        // Si el contador es 4, reiniciamos el contador
        if (cont == 4)
        {
                cont = 0;
        }
}

// Si no se produce una iluminación o no iluminación, entramos en el else. De esta forma, calculamos el
primer elemento de inicialización
else {
        l_iRow = (long)(l_bRow ? m_vRow[l_ui64FlashIndex >> 1] : -1);
        l_iColumn = (long)(l_bRow ? -1 : m_vColumn[l_ui64FlashIndex >> 1]);
}

switch(m_ui32LastState)
{
        case State_Flash:
                l_oStimulationSet.appendStimulation
                (OVTK_StimulationId_VisualStimulationStop,l_ui64CurrentTime, 0);
                _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
                OVTK_StimulationId_VisualStimulationStop\n");

                break;

        case State_NoFlash:
                break;

        case State_RepetitionRest:

                if (l_ui32State != State_TrialRest && l_ui32State != State_None) {

                        l_oStimulationSet.appendStimulation(OVTK_StimulationId_SegmentStart,
                        l_ui64CurrentTime, 0);

                        _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
                        OVTK_StimulationId_SegmentStart\n");
                }
                break;
```

```
	case State_TrialRest:

		if (m_ui64TrialIndex<=m_ui64TrialCount) {

			l_oStimulationSet.appendStimulation(OVTK_StimulationId_RestStop,l_ui64CurrentTime,
			0);
			_OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
			OVTK_StimulationId_RestStop\n");

			l_oStimulationSet.appendStimulation(OVTK_StimulationId_TrialStart,
			l_ui64CurrentTime, 0);
			_OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
			OVTK_StimulationId_TrialStart\n");

			l_oStimulationSet.appendStimulation(OVTK_StimulationId_SegmentStart,
			l_ui64CurrentTime, 0);
			_OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
			OVTK_StimulationId_SegmentStart\n");
		}
		break;

	case State_None:

		if (m_ui64TrialIndex<=m_ui64TrialCount) {

			l_oStimulationSet.appendStimulation(OVTK_StimulationId_ExperimentStart,
			l_ui64CurrentTime, 0);
			_OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
			OVTK_StimulationId_ExperimentStart\n");
		}
		break;

	default:
		break;
}

switch(l_ui32State)
{
	case State_Flash:

		l_oStimulationSet.appendStimulation(m_bRow ? m_ui64RowStimulationBase +
		l_iRow:m_ui64ColumnStimulationBase + l_iColumn, l_ui64CurrentTime, 0);

		_OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
		OVTK_StimulationId_LabelId(x)\n");
```

```
        l_oStimulationSet.appendStimulation(OVTK_StimulationId_VisualStimulationStart,
        l_ui64CurrentTime, 0);

        _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
        OVTK_StimulationId_VisualStimulationStart\n");

        break;

case State_NoFlash:
        break;

case State_RepetitionRest:

        l_oStimulationSet.appendStimulation(OVTK_StimulationId_SegmentStop, l_ui64CurrentTime, 0);

        _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
        OVTK_StimulationId_SegmentStop\n");

        this->generate_sequence();

        break;

case State_TrialRest:

        if (m_ui32LastState != State_None) {

                if (m_ui32LastState != State_RepetitionRest) {

                        l_oStimulationSet.appendStimulation(OVTK_StimulationId_SegmentStop,
                        l_ui64CurrentTime, 0);

                        _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace <<
                        "sends OVTK_StimulationId_SegmentStop\n");
                }

                l_oStimulationSet.appendStimulation(OVTK_StimulationId_TrialStop,l_ui64CurrentTime,
                0);

                _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
                OVTK_StimulationId_TrialStop\n");
        }
```

```
                                        if (m_ui64TrialIndex <= m_ui64TrialCount) {

                                                l_oStimulationSet.appendStimulation(OVTK_StimulationId_RestStart, l_ui64CurrentTime,
                                                0);

                                                _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
                                                OVTK_StimulationId_RestStart\n");
                                        }
                                        break;

                                case State_None:

                                        if (m_ui32LastState != State_RepetitionRest) {

                                                l_oStimulationSet.appendStimulation(OVTK_StimulationId_SegmentStop,
                                                l_ui64CurrentTime, 0);

                                                _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
                                                OVTK_StimulationId_SegmentStop\n");
                                        }

                                        l_oStimulationSet.appendStimulation(OVTK_StimulationId_TrialStop, l_ui64CurrentTime, 0);

                                        _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
                                        OVTK_StimulationId_TrialStop\n");

                                        break;

                                case State_ExperimentStop:

                                        l_oStimulationSet.appendStimulation(OVTK_StimulationId_ExperimentStop,l_ui64CurrentTime   +
                                        ITimeArithmetics::secondsToTime(3.0), 0);

                                        _OPTIONAL_LOG_(this->getLogManager(), LogLevel_Trace << "sends
                                        OVTK_StimulationId_ExperimentStop\n");
                                        break;

                                default:
                                        break;
                        }

                        m_ui32LastState = l_ui32State;
                }

        }
```

```
		TParameterHandler < IStimulationSet* > ip_pStimulationSet(m_pStimulationEncoder->getInputParameter
		(OVP_GD_Algorithm_StimulationStreamEncoder_InputParameterId_StimulationSet));

		TParameterHandler < IMemoryBuffer* > op_pMemoryBuffer(m_pStimulationEncoder->getOutputParameter
		(OVP_GD_Algorithm_StimulationStreamEncoder_OutputParameterId_EncodedMemoryBuffer));

		ip_pStimulationSet = &l_oStimulationSet;
		op_pMemoryBuffer = l_rDynamicBoxContext.getOutputChunk(0);

		if (!m_bHeaderSent) {

			m_pStimulationEncoder->process(OVP_GD_Algorithm_StimulationStreamEncoder_InputTriggerId_EncodeHeader);
			l_rDynamicBoxContext.markOutputAsReadyToSend(0, m_ui64LastTime, l_ui64CurrentTime);
		}

		if (m_ui64LastTime != l_ui64CurrentTime) {

			m_pStimulationEncoder->process(OVP_GD_Algorithm_StimulationStreamEncoder_InputTriggerId_EncodeBuffer);
			l_rDynamicBoxContext.markOutputAsReadyToSend(0, m_ui64LastTime, l_ui64CurrentTime);
		}

		m_ui64LastTime = l_ui64CurrentTime;
		m_bHeaderSent = true;
		return true;
	}

/*****************************************************************************************************/
/** Método generate_sequence(void) -Speller Stimulator-                                             **/
/** Se encarga de generar la secuencia de iluminaciones.                                            **/
/*****************************************************************************************************/
	void CBoxAlgorithmP300SpellerStimulator::generate_sequence(void) {

		uint32 i, j;

		// Declaración e inicialización de variables
		bool whiteRow = true;
		int cont = 0;

		// Declaración de vectores
		std::vector < uint32 > l_vRow;
		std::vector < uint32 > l_vRowWhite;
		std::vector < uint32 > l_vRowBlack;

		m_vRow.clear();
```

```
for (i = 0; i < m_ui64RowCount; i++) {
	l_vRow.push_back(i);

	if (i < 6) {
		// Si la fila es inferior a 6, la guardamos en el vector de filas de la matriz blanca
		l_vRowWhite.push_back(i);
	}
	else {
		// Si la fila es superior a 6, la guardamos en el vector de filas de la matriz negra
		l_vRowBlack.push_back(i);
	}
}
for (i = 0; i < m_ui64RowCount; i++) {

	if (whiteRow) {

		// Si la variable es true, hacemos que las filas de la matriz blanca se distribuyan de forma aleatoria. Las
		guardamos en la variable m_vRow. De esta forma, primero se iluminarán las filas de la matriz blanca y
		después las de la matriz negra
		j = rand() % l_vRowWhite.size();
		m_vRow[i] = l_vRowWhite[j];
		l_vRowWhite.erase(l_vRowWhite.begin() + j);

		// Ponemos la variable a false
		whiteRow = false;
	}
	else {
		// Si la variable es false, hacemos que las filas de la matriz negra se distribuyan de forma aleatoria. Las
		guardamos en la variable m_vRow. De esta forma, primero se iluminarán las filas de la matriz blanca y
		después las de la matriz negra
		j = rand() % l_vRowBlack.size();
		m_vRow[i] = l_vRowBlack[j];
		l_vRowBlack.erase(l_vRowBlack.begin() + j);

		// Ponemos la variable a true
		whiteRow = true;
	}
}

// Declaración de vectores
std::vector < uint32 > l_vColumn;
std::vector < uint32 > l_vColumnWhite;
std::vector < uint32 > l_vColumnBlack;
std::vector < uint32 > l_vColumnUnused;
```

```
m_vColumn.clear();

for (i = 0; i < m_ui64ColumnCount; i++) {

	l_vColumn.push_back(i);
	if (i < 3) {

		// Si la columna es inferior a 3, la guardamos en el vector de columnas de la matriz blanca
		l_vColumnWhite.push_back(i);
	}
	else if (i >= 3 && i < 6) {

		// Si la columna es superior a 3 e inferior a 6, la guardamos en el vector de columnas de la matriz negra
		l_vColumnBlack.push_back(i);
	}
	else {
		// Si la columna es superior a 6, la guardamos en el vector de columnas no utilizadas
		l_vColumnUnused.push_back(i);
	}
}

if (m_ui64ColumnCount == 12) {

	for (i = 0; i < m_ui64ColumnCount; i++) {

		if (cont == 0) {

			// Si el contador es 0, hacemos que las columnas de la matriz de columnas no utilizadas se distribuyan
			de forma aleatoria. Las guardamos en la variable m_vRow. De esta forma, primero se iluminarán una
			columna de la matriz de columnas no utilizadas, después una columna de la matriz blanca, una columna
			de la matriz de columnas no utilizadas y, por último, una columna de la matriz negra
			j = rand() % l_vColumnUnused.size();
			m_vColumn[i] = l_vColumnUnused[j];
			l_vColumnUnused.erase(l_vColumnUnused.begin() + j);


			// Sumamos uno al contador
			cont++;
		}
		else if (cont == 1) {
			// Si el contador es 1, hacemos que las columnas de la matriz blanca se distribuyan de forma
			aleatoria. Las guardamos en la variable m_vRow. De esta forma, primero se iluminarán una columna de
			la matriz de columnas no utilizadas, después una columna de la matriz blanca, una columna de la
			matriz de columnas no utilizadas y, por último, una columna de la matriz negra
			j = rand() % l_vColumnWhite.size();
```

```
                        m_vColumn[i] = l_vColumnWhite[j];
                        l_vColumnWhite.erase(l_vColumnWhite.begin() + j);

                        // Sumamos uno al contador
                        cont++;
                }
                else if (cont == 2) {

                        // Si el contador es 2, hacemos que las columnas de la matriz de columnas no utilizadas se distribuyan
                        de forma aleatoria. Las guardamos en la variable m_vRow. De esta forma, primero se iluminarán una
                        columna de la matriz de columnas no utilizadas, después una columna de la matriz blanca, una columna
                        de la matriz de columnas no utilizadas y, por último, una columna de la matriz negra
                        j = rand() % l_vColumnUnused.size();
                        m_vColumn[i] = l_vColumnUnused[j];
                        l_vColumnUnused.erase(l_vColumnUnused.begin() + j);

                        // Sumamos uno al contador
                        cont++;
                }
                else if (cont == 3) {

                        // Si el contador es 3, hacemos que las columnas de la matriz negra se distribuyan de forma aleatoria.
                        Las guardamos en la variable m_vRow. De esta forma, primero se iluminarán una columna de la matriz
                        de columnas no utilizadas, después una columna de la matriz blanca, una columna de la matriz de
                        columnas no utilizadas y, por último, una columna de la matriz negra
                        j = rand() % l_vColumnBlack.size();
                        m_vColumn[i] = l_vColumnBlack[j];
                        l_vColumnBlack.erase(l_vColumnBlack.begin() + j);
                        // Ponemos el contador a 0
                        cont = 0;
                }
        }
}
else if (m_ui64ColumnCount == 6) {

        for (i = 0; i < m_ui64ColumnCount; i++) {

                if (cont == 0) {

                        // Si el contador es 0, hacemos que las columnas de la matriz blanca se distribuyan de forma
                        aleatoria. Las guardamos en la variable m_vRow. De esta forma, primero se iluminará una columna de
                        la matriz blanca y, después, una columna de la matriz negra
                        j = rand() % l_vColumnWhite.size();
                        m_vColumn[i] = l_vColumnWhite[j];
                        l_vColumnWhite.erase(l_vColumnWhite.begin() + j);
```

```
                                        // Sumamos uno al contador
                                        cont++;
                                }
                                else if (cont == 1) {

                                        // Si el contador es 1, hacemos que las columnas de la matriz negra se distribuyan de forma aleatoria.
                                        Las guardamos en la variable m_vRow. De esta forma, primero se iluminará una columna de la matriz
                                        blanca y, después, una columna de la matriz negra
                                        j = rand() % l_vColumnBlack.size();
                                        m_vColumn[i] = l_vColumnBlack[j];
                                        l_vColumnBlack.erase(l_vColumnBlack.begin() + j);
                                        // Ponemos el contado a 0
                                        cont = 0;
                                }
                        }
                }
        }

/*******************************************************************************************************/
/** Método process(void) -Speller Visualisation-                                                      **/
/** Se encarga de producir la estimulación necesaria para que el usuario pueda escribir el carácter   **/
/** que desea.                                                                                        **/
/*******************************************************************************************************/
        bool CBoxAlgorithmP300SpellerVisualisation::process(void) {

                IBoxIO& l_rDynamicBoxContext = this->getDynamicBoxContext();
                uint32 i, j, k;

                // --- Sequence stimulations
                for (i = 0; i<l_rDynamicBoxContext.getInputChunkCount(0); i++) {

                        CStimulationSet l_oFlaggingStimulationSet;

                        ip_pSequenceMemoryBuffer = l_rDynamicBoxContext.getInputChunk(0, i);
                        ip_pTargetFlaggingStimulationSet = &l_oFlaggingStimulationSet;
                        op_pTargetFlaggingMemoryBuffer = l_rDynamicBoxContext.getOutputChunk(0);

                        m_pSequenceStimulationDecoder->process();

                        m_ui64LastTime = l_rDynamicBoxContext.getInputChunkEndTime(0, i);

                        if (m_pSequenceStimulationDecoder->isOutputTriggerActive
                        (OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedHeader)) {
```

```
		m_pTargetFlaggingStimulationEncoder->process
			(OVP_GD_Algorithm_StimulationStreamEncoder_InputTriggerId_EncodeHeader);
	}

	if (m_pSequenceStimulationDecoder->isOutputTriggerActive
	(OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedBuffer)) {

		IStimulationSet* l_pStimulationSet = op_pSequenceStimulationSet;

		for (j = 0; j < l_pStimulationSet->getStimulationCount(); j++) {

			uint64 l_ui64StimulationIdentifier = l_pStimulationSet->getStimulationIdentifier(j);

			bool l_bFlash = false;
			int l_iRow = -1;
			int l_iColumn = -1;
			bool l_bIsTarget = false;

			if (l_ui64StimulationIdentifier >= m_ui64RowStimulationBase && l_ui64StimulationIdentifier <
			m_ui64RowStimulationBase + m_ui64RowCount) {

				// Se realiza la diferencia entre el identificador y la base. A partir de este valor,
				obtenemos la fila
				l_iRow = (int)(l_ui64StimulationIdentifier - m_ui64RowStimulationBase);

				// Hace que se ilumine la fila
				l_bFlash = true;

				if (l_iRow < m_ui64RowCountNew) {

					// Si la fila obtenida es menor que el nuevo contador de filas (6), ponemos a true
					la variable que identifica la matriz blanca
					l_bWhiteTable = true;
				}
				else {

					// Si la fila obtenida es mayor que el nuevo contador de filas (6), ponemos a false
					la variable que identifica la matriz blanca, por lo que se iluminará la matriz negra
					l_bWhiteTable = false;


					// La nueva fila será la fila obtenida menos el nuevo contador de  filas (6)
					l_iRow = l_iRow - m_ui64RowCountNew;
				}
```

```
		// La fila objetivo será aquella que cumpla que la fila sea igual a la última fila objetivo
		iluminada de las matrices blanca o negra y que la matriz objetivo coincida con la matriz
		blanca o negra
		l_bIsTarget = (l_iRow == m_iLastTargetColorRow) && (l_bWhiteTable ==
		m_bLastTargetWhiteTable);
	}

	if (l_ui64StimulationIdentifier >= m_ui64ColumnStimulationBase &&   l_ui64StimulationIdentifier <
	m_ui64ColumnStimulationBase + m_ui64ColumnCount) {

		// Se realiza la diferencia entre el identificador y la base. A partir de este valor,
		obtenemos la columna.
		l_iColumn = (int)(l_ui64StimulationIdentifier - m_ui64ColumnStimulationBase);

		// Hace que se ilumine la columna
		l_bFlash = true;

		if (l_iColumn < m_ui64ColumnCountNew) {

			// Si la columna obtenida es menor que el nuevo contador de columnas (3), ponemos a
			true la variable que identifica la matriz blanca
			l_bWhiteTable = true;
		}
		else {
			// Si la columna obtenida es mayor que el nuevo contador de columnas (3), ponemos a
			false la variable que identifica la matriz blanca, por lo que se iluminará la matriz
			negra
			l_bWhiteTable = false;


			// La nueva columna será la columna obtenida menos el nuevo contador de columnas (3)
			l_iColumn = l_iColumn - m_ui64ColumnCountNew;
		}

		// La columna objetivo será aquella que cumpla que la columna sea igual a la última columna
		objetivo iluminada de las matrices blanca o negra y que la matriz objetivo coincida con la
		matriz blanca o negra
		l_bIsTarget = (l_iColumn == m_iLastTargetColorColumn) && (l_bWhiteTable ==
		m_bLastTargetWhiteTable);
	}

	if (l_ui64StimulationIdentifier == OVTK_StimulationId_VisualStimulationStop) {

		this->getLogManager() << LogLevel_Debug << "Received
		OVTK_StimulationId_VisualStimulationStop - resets grid\n";
```

```
        this->_cache_for_each_
        (&CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_,
        &m_oNoFlashBackgroundColor);
        this->_cache_for_each_
        (&CBoxAlgorithmP300SpellerVisualisation::_cache_change_foreground_cb_,
        &m_oNoFlashForegroundColor);
        this->_cache_for_each_
        (&CBoxAlgorithmP300SpellerVisualisation::_cache_change_font_cb_,
        m_pNoFlashFontDescription);
}
if (l_ui64StimulationIdentifier == OVTK_StimulationId_Reset) {

        gtk_label_set_text(m_pTarget, "");
        gtk_label_set_text(m_pResult, "");
}

if (l_bFlash) {

        // Llamadas a la función que realiza las iluminaciones a través de la matriz blanca y la
        matriz negra
        this->_cache_for_each_if_color_(
                l_iRow,
                l_iColumn,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_foreground_cb_,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_foreground_cb_,
                &m_oFlashForegroundColor,
                &m_oNoFlashForegroundColor,
                l_bWhiteTable);
        this->_cache_for_each_if_color_(
                l_iRow,
                l_iColumn,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_font_cb_,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_font_cb_,
                m_pFlashFontDescription,
                m_pNoFlashFontDescription,
                l_bWhiteTable);

        if (l_bIsTarget) {

                m_vStimuliQueue.push_back(OVTK_StimulationId_Target);
                l_oFlaggingStimulationSet.appendStimulation
                (OVTK_StimulationId_Target,l_pStimulationSet->getStimulationDate(j), 0);
        }
        else {
                m_vStimuliQueue.push_back(OVTK_StimulationId_NonTarget);
```

```
                                        l_oFlaggingStimulationSet.appendStimulation
                                        (OVTK_StimulationId_NonTarget, l_pStimulationSet->getStimulationDate(j), 0);
                                }
                        }

                        m_vStimuliQueue.push_back(l_ui64StimulationIdentifier);
                }

                m_pTargetFlaggingStimulationEncoder->process
                (OVP_GD_Algorithm_StimulationStreamEncoder_InputTriggerId_EncodeBuffer);
        }

        if (m_pSequenceStimulationDecoder->isOutputTriggerActive
        (OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedEnd)) {

                m_pTargetFlaggingStimulationEncoder->process
                (OVP_GD_Algorithm_StimulationStreamEncoder_InputTriggerId_EncodeEnd);
        }

        l_rDynamicBoxContext.markInputAsDeprecated(0, i);
        l_rDynamicBoxContext.markOutputAsReadyToSend(0, l_rDynamicBoxContext.getInputChunkStartTime(0, i),
        l_rDynamicBoxContext.getInputChunkEndTime(0, i));
}


// --- Target stimulations
for (i = 0; i < l_rDynamicBoxContext.getInputChunkCount(1); i++) {

        if (m_ui64LastTime >= l_rDynamicBoxContext.getInputChunkStartTime(1, i)) {

                ip_pTargetMemoryBuffer = l_rDynamicBoxContext.getInputChunk(1, i);
                m_pTargetStimulationDecoder->process();

                if (m_pTargetStimulationDecoder->isOutputTriggerActive
                (OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedBuffer)) {

                        IStimulationSet* l_pStimulationSet = op_pTargetStimulationSet;

                        for (j = 0; j < l_pStimulationSet->getStimulationCount(); j++) {

                                uint64 l_ui64StimulationIdentifier = l_pStimulationSet->getStimulationIdentifier(j);

                                // Declaración de variables
                                bool l_bTarget = false;
```

```
if (l_ui64StimulationIdentifier >= m_ui64RowStimulationBase && l_ui64StimulationIdentifier
< m_ui64RowStimulationBase + m_ui64RowCount) {

	this->getLogManager() << LogLevel_Debug << "Received Target Row " <<
	l_ui64StimulationIdentifier << "\n";

	// Se realiza la diferencia entre el identificador y la base. A partir de este valor,
	obtenemos la fila.
	m_iTargetRow = (int)(l_ui64StimulationIdentifier -  m_ui64RowStimulationBase);

	// Hace que exista una fila objetivo
	l_bTarget = true;
}

if (l_ui64StimulationIdentifier >= m_ui64ColumnStimulationBase &&
l_ui64StimulationIdentifier < m_ui64ColumnStimulationBase + m_ui64ColumnCount) {

	this->getLogManager() << LogLevel_Debug << "Received Target Column" <<
	l_ui64StimulationIdentifier << "\n";

	// Se realiza la diferencia entre el identificador y la base. A partir de este valor,
	obtenemos la   columna.
	m_iTargetColumn = (int)(l_ui64StimulationIdentifier -
	m_ui64ColumnStimulationBase);

	// Hace que exista una fila objetivo
	l_bTarget = true;
}

if (l_bTarget && m_iTargetRow != -1 && m_iTargetColumn != -1) {

	this->getLogManager() << LogLevel_Debug << "Displays Target Cell\n";
	this->_cache_for_each_if_(
		m_iTargetRow,
		m_iTargetColumn,
		&CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_,
		&CBoxAlgorithmP300SpellerVisualisation::_cache_change_null_cb_,
		&m_oTargetBackgroundColor,
		NULL);
	this->_cache_for_each_if_(
		m_iTargetRow,
		m_iTargetColumn,
		&CBoxAlgorithmP300SpellerVisualisation::_cache_change_foreground_cb_,
		&CBoxAlgorithmP300SpellerVisualisation::_cache_change_null_cb_,
		&m_oTargetForegroundColor,
```

```
        NULL);
this->_cache_for_each_if_(
        m_iTargetRow,
        m_iTargetColumn,
        &CBoxAlgorithmP300SpellerVisualisation::_cache_change_font_cb_,
        &CBoxAlgorithmP300SpellerVisualisation::_cache_change_null_cb_,
        m_pTargetFontDescription,
        NULL);

std::vector < ::GtkWidget* > l_vWidgets;
this->_cache_for_each_if_(
        m_iTargetRow,
        m_iTargetColumn,
        &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
        &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
        &l_vWidgets,
        NULL);

{
m_vStimuliQueue.push_back(m_iTargetRow + m_ui64RowStimulationBase);
m_vStimuliQueue.push_back(m_iTargetColumn + m_ui64ColumnStimulationBase);
}

if (l_vWidgets.size() == 1) {

        if (GTK_IS_LABEL(l_vWidgets[0])) {

                std::string l_sString;
                l_sString = gtk_label_get_text(m_pTarget);
                l_sString += gtk_label_get_text(GTK_LABEL(l_vWidgets[0]));
                gtk_label_set_text(m_pTarget, l_sString.c_str());
        }
        else {
                this->getLogManager() << LogLevel_Warning << "Expected label class
                widget... could not find a valid text to append\n";
        }
}
else {

        this->getLogManager() << LogLevel_Warning << "Did not find a unique widget at
        row:" << (uint32)m_iTargetRow << " column:" << (uint32)m_iTargetColumn <<
        "\n";
}

m_vTargetHistory.push_back(std::make_pair(m_iTargetRow, m_iTargetColumn));
```

```
                                        m_iLastTargetRow = m_iTargetRow;
                                        m_iLastTargetColumn = m_iTargetColumn;

                                        // Llamada a la función que convierte las letras de la matriz 6x6 en dos matrices,
                                        una blanca y otra negra
                                        m_bLastTargetWhiteTable = _convert_original_table_to_color_table_
                                        (m_cTable[m_iLastTargetRow][m_iLastTargetColumn]);

                                        m_iTargetRow = -1;
                                        m_iTargetColumn = -1;
                                }
                        }
                }

                l_rDynamicBoxContext.markInputAsDeprecated(1, i);
        }
}

// --- Selection stimulations
for (k = 2; k < 4; k++) {

        IAlgorithmProxy* l_pSelectionStimulationDecoder = (k == 2 ? m_pRowSelectionStimulationDecoder :
        m_pColumnSelectionStimulationDecoder);
        TParameterHandler < const IMemoryBuffer* > ip_pSelectionMemoryBuffer(l_pSelectionStimulationDecoder->
        getInputParameter(OVP_GD_Algorithm_StimulationStreamDecoder_InputParameterId_MemoryBufferToDecode));
        TParameterHandler < IStimulationSet* > op_pSelectionStimulationSet(l_pSelectionStimulationDecoder->
        getOutputParameter(OVP_GD_Algorithm_StimulationStreamDecoder_OutputParameterId_StimulationSet));

        for (i = 0; i < l_rDynamicBoxContext.getInputChunkCount(k); i++) {

                if (m_ui64LastTime >= l_rDynamicBoxContext.getInputChunkStartTime(k, i)) {

                        ip_pSelectionMemoryBuffer = l_rDynamicBoxContext.getInputChunk(k, i);
                        l_pSelectionStimulationDecoder->process();
                        if (l_pSelectionStimulationDecoder->isOutputTriggerActive
                        (OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedBuffer)) {

                                IStimulationSet* l_pStimulationSet = op_pSelectionStimulationSet;

                                for (j = 0; j < l_pStimulationSet->getStimulationCount(); j++) {

                                        uint64 l_ui64StimulationIdentifier = l_pStimulationSet->getStimulationIdentifier(j);

                                        bool l_bSelected = false;
```

```
if (l_ui64StimulationIdentifier >= m_ui64RowStimulationBase &&
l_ui64StimulationIdentifier < m_ui64RowStimulationBase + m_ui64RowCount) {

        this->getLogManager() << LogLevel_Debug << "Received Selected Row " <<
        l_ui64StimulationIdentifier << "\n";
        m_iSelectedRow = (int)(l_ui64StimulationIdentifier -
        m_ui64RowStimulationBase);

        l_bSelected = true;
}
if (l_ui64StimulationIdentifier >= m_ui64ColumnStimulationBase &&
l_ui64StimulationIdentifier < m_ui64ColumnStimulationBase + m_ui64ColumnCount) {

        this->getLogManager() << LogLevel_Debug << "Received Selected Column " <<
        l_ui64StimulationIdentifier << "\n";
        m_iSelectedColumn = (int)(l_ui64StimulationIdentifier -
        m_ui64ColumnStimulationBase);

        l_bSelected = true;
}
if (l_ui64StimulationIdentifier == OVTK_StimulationId_Label_00) {

        if (k == 2) m_iSelectedRow = -2;
        if (k == 3) m_iSelectedColumn = -2;

        l_bSelected = true;
}
if (l_bSelected && m_iSelectedRow != -1 && m_iSelectedColumn != -1)
{

        // Llamada a la función que convierte la letra seleccionada de las matrices
        blanca y negra, en la letra seleccionada de la matriz 6x6
        _convert_selection_to_original_table_();

        if (m_iSelectedRow >= 0 && m_iSelectedColumn >= 0) {

                this->getLogManager() << LogLevel_Debug << "Displays Selected Cell\n";
                this->_cache_for_each_if_(
                        m_iSelectedRow,
                        m_iSelectedColumn,
                        &CBoxAlgorithmP300SpellerVisualisation::
                        _cache_change_background_cb_,
                        &CBoxAlgorithmP300SpellerVisualisation::
                        _cache_change_null_cb_,
                        &m_oSelectedBackgroundColor,
```

```
        NULL);
this->_cache_for_each_if_(
        m_iSelectedRow,
        m_iSelectedColumn,
        &CBoxAlgorithmP300SpellerVisualisation::
        _cache_change_foreground_cb_,
        &CBoxAlgorithmP300SpellerVisualisation::
        _cache_change_null_cb_,
        &m_oSelectedForegroundColor,
        NULL);
this->_cache_for_each_if_(
        m_iSelectedRow,
        m_iSelectedColumn,
        &CBoxAlgorithmP300SpellerVisualisation::
        _cache_change_font_cb_,
        &CBoxAlgorithmP300SpellerVisualisation::
        _cache_change_null_cb_,
        m_pSelectedFontDescription,
        NULL);

std::vector < ::GtkWidget* > l_vWidgets;
this->_cache_for_each_if_(
        m_iSelectedRow,
        m_iSelectedColumn,
        &CBoxAlgorithmP300SpellerVisualisation::
        _cache_collect_child_widget_cb_,
        &CBoxAlgorithmP300SpellerVisualisation::
        _cache_collect_child_widget_cb_,
        &l_vWidgets,
        NULL);

if (l_vWidgets.size() == 1) {

        if (GTK_IS_LABEL(l_vWidgets[0])) {

                std::string l_sString;
                l_sString = gtk_label_get_text
                (GTK_LABEL(l_vWidgets[0]));

                l_sString = std::string gtk_label_get_label
                (m_pResult)) + l_sString;
                gtk_label_set_markup(m_pResult,
                l_sString.c_str());
        }
```

```
                                                        else {
                                                                this->getLogManager() << LogLevel_Warning <<
                                                                "Expected label class widget... could not
                                                                find a valid text to append\n";
                                                        }
                                                }
                                                else {
                                                        this->getLogManager() << LogLevel_Warning << "Did not find a
                                                        unique widget at row : " << (uint32)m_iSelectedRow << " column
                                                        : " << (uint32)m_iSelectedColumn << "\n";
                                                }
                                        }
                                        else {
                                                this->getLogManager() << LogLevel_Trace << "Selection Rejected !\n";
                                                std::string l_sString;

                                                l_sString = gtk_label_get_text(m_pResult);
                                                l_sString += "*";
                                                gtk_label_set_text(m_pResult, l_sString.c_str());
                                        }

                                        m_iSelectedRow = -1;
                                        m_iSelectedColumn = -1;
                                }
                        }
                }

                l_rDynamicBoxContext.markInputAsDeprecated(k, i);
            }
        }
    }
    return true;
}

/*************************************************************************************************/
/** Método _cache_for_each_if_color_(int iLine, int iColumn, _cache_callback_ fpIfCallback,     **/
/** _cache_callback_ fpElseCallback, void* pIfUserData, void* pElseUserData, bool l_bWhiteTable) **/
/** -Speller Visualisation-                                                                     **/
/** Se encarga de distribuir los caracteres en las matrices blanca y negra.                     **/
/*************************************************************************************************/
void CBoxAlgorithmP300SpellerVisualisation::_cache_for_each_if_color_(int iLine, int iColumn,   _cache_callback_   fpIfCallback,
_cache_callback_ fpElseCallback, void* pIfUserData, void*  pElseUserData, bool l_bWhiteTable) {

        std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle > >::iterator i;
        std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;
```

```
bool l_bLine = (iLine != -1);
bool l_bColumn = (iColumn != -1);

// Declaración de variables
std::string lettersColumnColor;
std::string lettersRowColor;

if (l_bWhiteTable)
{
	if (l_bLine) {

		for (int i = 0; i < m_ui64ColumnCountNew; i++) {

			lettersRowColor.push_back(m_cWhiteTable[iLine][i]);
		}
	}
	if (l_bColumn) {

		for (int i = 0; i < m_ui64RowCountNew; i++) {

			lettersColumnColor.push_back(m_cWhiteTable[i][iColumn]);
		}
	}
}
else {
	if (l_bLine) {

		for (int i = 0; i < m_ui64ColumnCountNew; i++) {

			lettersRowColor.push_back(m_cBlackTable[iLine][i]);
		}
	}
	if (l_bColumn) {

		for (int i = 0; i < m_ui64RowCountNew; i++) {

			lettersColumnColor.push_back(m_cBlackTable[i][iColumn]);
		}
	}

}

for (i = m_vCache.begin(); i != m_vCache.end(); i++) {
```

```
                        for (j = i->second.begin(); j != i->second.end(); j++) {

                                bool l_bIf;
                                std::string l_sString;
                                l_sString = gtk_label_get_text(GTK_LABEL(j->second.pChildWidget));

                                if (lettersRowColor.find(l_sString.at(0)) != std::string::npos || lettersColumnColor.find(l_sString.at(0))
                                != std::string::npos) {

                                        l_bIf = true;
                                }
                                else {
                                        l_bIf = false;
                                }

                                if (l_bIf) {

                                        (this->*fpIfCallback)(j->second, pIfUserData);
                                }
                                else {
                                        (this->*fpElseCallback)(j->second, pElseUserData);
                                }
                        }
                }
        }

/*************************************************************************************************/
/** Método _convert_original_table_to_color_table_(char letter) -Speller Visualisation-         **/
/** Se encarga de convertir la matriz original en dos matrices, una blanca y una negra.         **/
/*************************************************************************************************/
        bool CBoxAlgorithmP300SpellerVisualisation::_convert_original_table_to_color_table_(char letter) {

                for (int i = 0; i < m_ui64RowCountNew; i++) {

                        for (int j = 0; j < m_ui64ColumnCountNew; j++) {

                                // Buscamos la letra dentro de la matriz blanca
                                if (m_cWhiteTable[i][j] == letter) {

                                        // Asociamos la fila de la matriz blanca con la fila objetivo
                                        m_iLastTargetColorRow = i;
                                        // Asociamos la columna de la matriz blanca con la columna objetivo
                                        m_iLastTargetColorColumn = j;

                                        // Se devuelve true para identificar la matriz blanca
```

```
                                return true;
                        }
                }
        }
        for (int i = 0; i < m_ui64RowCountNew; i++) {

                for (int j = 0; j < m_ui64ColumnCountNew; j++) {

                        // Buscamos la letra dentro de la matriz negra
                        if (m_cBlackTable[i][j] == letter) {

                                // Asociamos la fila de la matriz negra con la fila objetivo
                                m_iLastTargetColorRow = i;
                                // Asociamos la columna de la matriz negra con la columna objetivo
                                m_iLastTargetColorColumn = j;

                                // Se devuelve true para identificar la matriz negra
                                return false;
                        }
                }
        }
        return false;
}

/*********************************************************************************************************/
/** Método _convert_selection_to_original_table_(void) -Speller Visualisation-                          **/
/** Se encarga de convertir la selección que ha realizado el usuario en las matrices blanca o negra a    **/
/** a la matriz original.                                                                               **/
/*********************************************************************************************************/
void CBoxAlgorithmP300SpellerVisualisation::_convert_selection_to_original_table_(void) {

        // Declaración de variables
        bool l_bWhiteTable;
        char letter;

        // Inicializamos las variables
        int l_iRow = m_iSelectedRow;
        int l_iColumn = m_iSelectedColumn;

        if (l_iRow < m_ui64RowCountNew) {

                // Si la fila es menor que el nuevo contador de filas (6), la selección pertenece a la matriz
                l_bWhiteTable = true;
        }
        else {
```

```
        // Si la fila es mayor que el nuevo contador de filas (6), la selección pertenece a la matriz negra
        l_bWhiteTable = false;

        // La fila será la diferencia entre la fila y el nuevo contador de filas (6)
        l_iRow = l_iRow - m_ui64RowCountNew;
}

if (l_iColumn < m_ui64ColumnCountNew) {

        // Si la columna es menor que el nuevo contador de columnas (3), la selección pertenece a la matriz negra
        l_bWhiteTable = true;
}
else {
        // Si la columna es mayor que el nuevo contador de columnas (3), la selección pertenece a la matriz negra
        l_bWhiteTable = false;

        // La columna será la diferencia entre la columna y el nuevo contador de columnas (3)
        l_iColumn = l_iColumn - m_ui64ColumnCountNew;

        if (l_iColumn >= m_ui64ColumnCountNew) {


                // La columna será la diferencia entre la columna y el nuevo contador de columnas (3)
                l_iColumn = l_iColumn - m_ui64ColumnCountNew;
        }
}

if (l_bWhiteTable) {

        // Si la matriz es blanca, la letra que buscamos será la letra perteneciente al vector de la matriz blanca cuyas
        fila y columna sean l_iRow y l_iColumn
        letter = m_cWhiteTable[l_iRow][l_iColumn];
}
else {
        // Si la matriz es negra, la letra que buscamos será la letra perteneciente al vector de la matriz negra cuyas fila
        y columna sean l_iRow y l_iColumn
        letter = m_cBlackTable[l_iRow][l_iColumn];
}

for (int i = 0; i < m_ui64RowCountNew; i++) {

        for (int j = 0; j < m_ui64ColumnCount; j++) {

                // Buscamos cual es la letra dentro de la matriz 6x6
                if (m_cTable[i][j] == letter) {
```

```
                                    // Asociamos la fila de la matriz 6x6 con la fila seleccionada
                                    m_iSelectedRow = i;
                                    // Asociamos la columna de la matriz 6x6 con la columna seleccionada
                                    m_iSelectedColumn = j;
                                }
                            }
                        }
                    }
```

2. Honeycomb-Shaped Red Dots Paradigm

```
/*-------------------------------------------------------------------------------------------------------*/
/*                          P300 SPELLER – Honeycomb-Shaped Red Dots Paradigm                            */
/*-------------------------------------------------------------------------------------------------------*/
/* Aplicación que permite al usuario escribir caracteres utilizando solamente las ondas cerebrales.      */
/* Para ello se utiliza el paradigma HSRD.                                                               */
/*                                                                                    Fecha: 20/04/2018  */
/*                                                                  Cristina Cantalapiedra Cabezas       */
/*-------------------------------------------------------------------------------------------------------*/

/*********************************************************************************************************/
/** Método initialize(void) -Speller Visualisation-                                                     **/
/** Se encarga de inicializar las variables que se van a utilizar en el proceso.                        **/
/*********************************************************************************************************/
	bool CBoxAlgorithmP300SpellerVisualisation::initialize(void) {

		TCHAR NPath[MAX_PATH];
		GetCurrentDirectory(MAX_PATH, NPath);

		const IBox& l_rStaticBoxContext=this->getStaticBoxContext();

		// Declaración e inicialización de la interfaz.
		m_pMainWidgetInterface=NULL;

		// Declaración e inicialización de la nueva interfaz.
		m_pAuxWidgetInterface = NULL;

		m_pToolbarWidgetInterface=NULL;
		m_pFlashFontDescription=NULL;
		m_pNoFlashFontDescription=NULL;
		m_pTargetFontDescription=NULL;
		m_pSelectedFontDescription=NULL;

		// Declaración e inicialización de la variable que contiene el número máximo de puntos dentro de la imagen del panal.
		m_uiNumMaxHoneyComb = 3;

		// Declaración e inicialización del directorio de imágenes.
		m_cImagesDir[0] = '\0';

		// Inicialización del directorio de imágenes.
		sprintf(m_cImagesDir, "..\\dist\\extras\\Debug\\share\\openvibe\\plugins\\simple-visualisation\\");

		m_sInterfaceFilename = FSettingValueAutoCast(*this->getBoxAlgorithmContext(), 0);
```

```
m_ui64RowStimulationBase =FSettingValueAutoCast(*this->getBoxAlgorithmContext(), 1);
m_ui64ColumnStimulationBase = FSettingValueAutoCast(*this->getBoxAlgorithmContext(), 2);

m_oFlashBackgroundColor = _AutoCast_(l_rStaticBoxContext, this->getConfigurationManager(), 3);
m_oFlashForegroundColor = _AutoCast_(l_rStaticBoxContext, this->getConfigurationManager(), 4);
m_ui64FlashFontSize = FSettingValueAutoCast(*this->getBoxAlgorithmContext(), 5);
m_oNoFlashBackgroundColor = _AutoCast_(l_rStaticBoxContext, this->getConfigurationManager(), 6);
m_oNoFlashForegroundColor = _AutoCast_(l_rStaticBoxContext, this->getConfigurationManager(), 7);
m_ui64NoFlashFontSize = FSettingValueAutoCast(*this->getBoxAlgorithmContext(), 8);
m_oTargetBackgroundColor = _AutoCast_(l_rStaticBoxContext, this->getConfigurationManager(), 9);
m_oTargetForegroundColor = _AutoCast_(l_rStaticBoxContext, this->getConfigurationManager(), 10);
m_ui64TargetFontSize = FSettingValueAutoCast(*this->getBoxAlgorithmContext(), 11);
m_oSelectedBackgroundColor = _AutoCast_(l_rStaticBoxContext, this->getConfigurationManager(), 12);
m_oSelectedForegroundColor = _AutoCast_(l_rStaticBoxContext, this->getConfigurationManager(), 13);
m_ui64SelectedFontSize = FSettingValueAutoCast(*this->getBoxAlgorithmContext(), 14);

m_pSequenceStimulationDecoder=&this->getAlgorithmManager().getAlgorithm(this->
getAlgorithmManager().createAlgorithm (OVP_GD_ClassId_Algorithm_StimulationStreamDecoder));
m_pSequenceStimulationDecoder->initialize();

m_pTargetStimulationDecoder=&this->getAlgorithmManager().getAlgorithm(this->
getAlgorithmManager().createAlgorithm(OVP_GD_ClassId_Algorithm_StimulationStreamDecoder));
m_pTargetStimulationDecoder->initialize();

m_pTargetFlaggingStimulationEncoder=&this->getAlgorithmManager().getAlgorithm(this->
getAlgorithmManager().createAlgorithm(OVP_GD_ClassId_Algorithm_StimulationStreamEncoder));
m_pTargetFlaggingStimulationEncoder->initialize();

m_pRowSelectionStimulationDecoder=&this->getAlgorithmManager().getAlgorithm(this->
getAlgorithmManager().createAlgorithm(OVP_GD_ClassId_Algorithm_StimulationStreamDecoder));
m_pRowSelectionStimulationDecoder->initialize();

m_pColumnSelectionStimulationDecoder=&this->getAlgorithmManager().getAlgorithm(this->
getAlgorithmManager().createAlgorithm(OVP_GD_ClassId_Algorithm_StimulationStreamDecoder));
m_pColumnSelectionStimulationDecoder->initialize();

// Creación de la interfaz.
m_pMainWidgetInterface=gtk_builder_new();

if (!gtk_builder_add_from_file(m_pMainWidgetInterface, m_sInterfaceFilename.toASCIIString(),NULL)) {

        this->getLogManager() << LogLevel_ImportantWarning << "Could not load interface file [" << _sInterfaceFilename <<
        "]\n";
        this->getLogManager() << LogLevel_ImportantWarning << "The file may be missing. However, the interface files now
        use gtk-builder instead of glade. Did you update your files?\n";
```

```
        return false;
}

// Como queremos otra interfaz diferente, en vez de con solo letras, con letras y panales, crearemos una nueva interfaz.
Creación de la interfaz.
m_pAuxWidgetInterface = gtk_builder_new();

if (!gtk_builder_add_from_file(m_pAuxWidgetInterface, m_sInterfaceFilename.toASCIIString(),                    NULL)) {

        this->getLogManager() << LogLevel_ImportantWarning << "Could not load interface file [" << m_sInterfaceFilename <<
        "]\n";
        this->getLogManager() << LogLevel_ImportantWarning << "The file may be missing. However, the interface files now
        use gtk-builder instead of glade. Did you update your files ?\n";

        return false;
}

m_pToolbarWidgetInterface=gtk_builder_new();
gtk_builder_add_from_file(m_pToolbarWidgetInterface, m_sInterfaceFilename.toASCIIString(), NULL);

m_pMainWindow=GTK_WIDGET(gtk_builder_get_object(m_pMainWidgetInterface, "p300-speller-main"));
m_pToolbarWidget=GTK_WIDGET(gtk_builder_get_object(m_pToolbarWidgetInterface, "p300-speller-toolbar"));

// Asigno a la variable m_pTable, la interfaz creada. Para ello, pasamos la interfaz a tipo GTK_TABLE.
m_pTable = GTK_TABLE(gtk_builder_get_object(m_pMainWidgetInterface, "p300-speller-table"));

// Asigno a la variable m_pAuxTable, la interfaz creada. Para ello, pasamos la interfaz a tipo GTK_TABLE.
m_pAuxTable = GTK_TABLE(gtk_builder_get_object(m_pAuxWidgetInterface, "p300-speller-table-aux"));

m_pResult = GTK_LABEL(gtk_builder_get_object(m_pMainWidgetInterface, "label-result"));
m_pTarget = GTK_LABEL(gtk_builder_get_object(m_pMainWidgetInterface, "label-target"));

gtk_builder_connect_signals(m_pMainWidgetInterface, NULL);
gtk_builder_connect_signals(m_pToolbarWidgetInterface, NULL);

g_signal_connect(gtk_builder_get_object(m_pToolbarWidgetInterface, "toolbutton-show_target_text"), "toggled",
G_CALLBACK(toggle_button_show_hide_cb), gtk_builder_get_object(m_pMainWidgetInterface, "label-arget"));
g_signal_connect(gtk_builder_get_object(m_pToolbarWidgetInterface, "toolbutton-show_target_text"), "toggled",
G_CALLBACK(toggle_button_show_hide_cb), gtk_builder_get_object(m_pMainWidgetInterface, "label-target-title"));
g_signal_connect(gtk_builder_get_object(m_pToolbarWidgetInterface, "toolbutton-show_result_text"), "toggled",
G_CALLBACK(toggle_button_show_hide_cb), gtk_builder_get_object(m_pMainWidgetInterface, "label-result"));
g_signal_connect(gtk_builder_get_object(m_pToolbarWidgetInterface, "toolbutton-_result_text"), "toggled",
G_CALLBACK(toggle_button_show_hide_cb), gtk_builder_get_object(m_pMainWidgetInterface, "label-result-title"));
```

```
        m_visualizationContext = dynamic_cast<OpenViBEVisualizationToolkit::IVisualizationContext*>(this->
        createPluginObject(OVP_ClassId_Plugin_VisualizationContext));
        m_visualizationContext->setWidget(*this, m_pMainWindow);
        m_visualizationContext->setToolbar(*this, m_pToolbarWidget);

        guint l_uiRowCount = 0;
        guint l_uiColumnCount = 0;
        g_object_get(m_pTable, "n-rows", &l_uiRowCount, NULL);
        g_object_get(m_pTable, "n-columns", &l_uiColumnCount, NULL);

        m_ui64RowCount = l_uiRowCount;
        m_ui64ColumnCount = l_uiColumnCount;

        // Creación de la tabla. La tabla contiene las imágenes con las letras y un fondo negro, para que la letra destaque. La
        variable m_pTable contiene la tabla que contiene las imágenes.
        this->_cache_build_from_table_(m_pTable);

        // Hacemos que el fondo de la tabla cambie a negro. No es necesario cambiar ni la fuente ni el color de la letra, ya que
        estamos tratando con imágenes.
        this->_cache_for_each_(&CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_,&m_oNoFlashBackgroundColor);

        // Creación de la nueva tabla, la tabla auxiliar.  La variable m_pAuxTable contiene la tabla que contiene los labels de
        cada celda, a través de los cuales obtenemos el valor.
        this->_cache_build_from_aux_table_(m_pAuxTable);
        this->_cache_for_each_aux_(&CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_aux_,
        &m_oNoFlashBackgroundColor);
        this->_cache_for_each_aux_(&CBoxAlgorithmP300SpellerVisualisation::_cache_change_foreground_cb_aux_,
        &m_oNoFlashForegroundColor);
        this->_cache_for_each_aux(&CBoxAlgorithmP300SpellerVisualisation::_cache_change_font_cb_aux_, l_pMaxFontDescription);

        m_iLastTargetRow = -1;
        m_iLastTargetColumn = -1;
        m_iTargetRow = -1;
        m_iTargetColumn = -1;
        m_iSelectedRow = -1;
        m_iSelectedColumn = -1;

        m_pStimulusSender = TCPTagging::createStimulusSender();

        m_bTableInitialized = false;

        return true;
}
```

```
/*******************************************************************************************************/
/** Método process(void)  -Speller Visualisation-                                                     **/
/** Se encarga de producir la estimulación necesaria para que el usuario pueda escribir el carácter   **/
/** que desea.                                                                                        **/
/*******************************************************************************************************/
	bool CBoxAlgorithmP300SpellerVisualisation::process(void) {

		IBoxIO& l_rDynamicBoxContext=this->getDynamicBoxContext();

		uint32 i, j, k;

		// --- Sequence stimulations
		for (i=0; i<l_rDynamicBoxContext.getInputChunkCount(0); i++) {

			CStimulationSet l_oFlaggingStimulationSet;

			m_pSequenceStimulationDecoder->process();
			m_ui64LastTime=l_rDynamicBoxContext.getInputChunkEndTime(0, i);

			if (m_pSequenceStimulationDecoder->isOutputTriggerActive
			(OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedHeader)) {

				m_pTargetFlaggingStimulationEncoder->process
				(OVP_GD_Algorithm_StimulationStreamEncoder_InputTriggerId_EncodeHeader);
			}

			if (m_pSequenceStimulationDecoder->isOutputTriggerActive
			(OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedBuffer)) {

				IStimulationSet* l_pStimulationSet = op_pSequenceStimulationSet;

				for (j = 0; j < l_pStimulationSet->getStimulationCount(); j++) {

					uint64 l_ui64StimulationIdentifier=l_pStimulationSet->getStimulationIdentifier(j);
					bool l_bFlash=false;
					int l_iRow=-1;
					int l_iColumn=-1;
					bool l_bIsTarget = false;

					if (l_ui64StimulationIdentifier >= m_ui64RowStimulationBase && l_ui64StimulationIdentifier <
					m_ui64RowStimulationBase+m_ui64RowCount) {

						l_iRow = (int)(l_ui64StimulationIdentifier-m_ui64RowStimulationBase);
						l_bFlash = true;
						l_bIsTarget = (l_iRow==m_iLastTargetRow);
```

```
}
if (l_ui64StimulationIdentifier >= m_ui64ColumnStimulationBase &&  l_ui64StimulationIdentifier    <
m_ui64ColumnStimulationBase+m_ui64ColumnCount) {

        l_iColumn = (int)(l_ui64StimulationIdentifier - m_ui64ColumnStimulationBase);
        l_bFlash = true;
        l_bIsTarget = (l_iColumn == m_iLastTargetColumn);
}
if (l_ui64StimulationIdentifier == OVTK_StimulationId_VisualStimulationStop) {

        this->getLogManager() << LogLevel_Debug << "Received
        OVTK_StimulationId_VisualStimulationStop - resets grid\n";

        // Llamada a la función auxiliar que hemos creado para cambiar el fondo, la fuente y el color
        de la letra de la tabla auxiliar con los labels.
        this->_cache_for_each_aux_ (&CBoxAlgorithmP300SpellerVisualisation::
        _cache_change_background_cb_aux_, &m_oNoFlashBackgroundColor);
        this->_cache_for_each_aux_(&CBoxAlgorithmP300SpellerVisualisation::
        _cache_change_foreground_cb_aux_, &m_oNoFlashForegroundColor);
        this->_cache_for_each_aux_ (&CBoxAlgorithmP300SpellerVisualisation::
        _cache_change_font_cb_aux_, m_pNoFlashFontDescription);

        // Llamada a la función para cambiar el fondo de la tabla con las imágenes.
        this->_cache_for_each_(&CBoxAlgorithmP300SpellerVisualisation::
        _cache_change_background_cb_, &m_oNoFlashBackgroundColor);

        // Llamada a la función que resetea las imágenes de la tabla.
        this->_reset_images_(&CBoxAlgorithmP300SpellerVisualisation::_cache_set_image_,
        &m_oNoFlashBackgroundColor);
}
if (l_ui64StimulationIdentifier == OVTK_StimulationId_Reset) {

        gtk_label_set_text(m_pTarget, "");
        gtk_label_set_text(m_pResult, "");
}

if (l_bFlash) {

        // Llamada a las funciones que realizan cambios en la tabla de los labels.
        this->_cache_for_each_if_aux_(
                l_iRow,
                l_iColumn,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_aux_,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_aux_,
                &m_oFlashBackgroundColor,
```

```
                        &m_oNoFlashBackgroundColor);
                this->_cache_for_each_if_aux_(
                        l_iRow,
                        l_iColumn,
                        &CBoxAlgorithmP300SpellerVisualisation::_cache_change_foreground_cb_aux_,
                        &CBoxAlgorithmP300SpellerVisualisation::_cache_change_foreground_cb_aux_,
                        &m_oFlashForegroundColor,
                        &m_oNoFlashForegroundColor);
                this->_cache_for_each_if_aux_(
                        l_iRow,
                        l_iColumn,
                        &CBoxAlgorithmP300SpellerVisualisation::_cache_change_font_cb_aux_,
                        &CBoxAlgorithmP300SpellerVisualisation::_cache_change_font_cb_aux_,
                        m_pFlashFontDescription,
                        m_pNoFlashFontDescription);

                // Llamada a la función que realiza cambios en la imagen.
                this->_cache_for_each_if_image_(
                        l_iRow,
                        l_iColumn,
                        &CBoxAlgorithmP300SpellerVisualisation::_cache_set_image_,
                        &CBoxAlgorithmP300SpellerVisualisation::_cache_set_image_,
                        &m_oFlashBackgroundColor,
                        &m_oNoFlashBackgroundColor);

                if(l_bIsTarget) {

                        m_vStimuliQueue.push_back(OVTK_StimulationId_Target);
                        l_oFlaggingStimulationSet.appendStimulation(OVTK_StimulationId_Target,
                        l_pStimulationSet->getStimulationDate(j), 0);
                }
                else {
                        m_vStimuliQueue.push_back(OVTK_StimulationId_NonTarget);
                        l_oFlaggingStimulationSet.appendStimulation(OVTK_StimulationId_NonTarget,
                        l_pStimulationSet->getStimulationDate(j), 0);
                }
        }

        m_vStimuliQueue.push_back(l_ui64StimulationIdentifier);
    }

    m_pTargetFlaggingStimulationEncoder->
    process(OVP_GD_Algorithm_StimulationStreamEncoder_InputTriggerId_EncodeBuffer);
}
```

```
        if(m_pSequenceStimulationDecoder->isOutputTriggerActive
        (OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedEnd)) {

                m_pTargetFlaggingStimulationEncoder->
                process(OVP_GD_Algorithm_StimulationStreamEncoder_InputTriggerId_EncodeEnd);
        }

        l_rDynamicBoxContext.markInputAsDeprecated(0, i);
        l_rDynamicBoxContext.markOutputAsReadyToSend(0, l_rDynamicBoxContext.getInputChunkStartTime(0, i),
        l_rDynamicBoxContext.getInputChunkEndTime(0, i));
}

// --- Target stimulations
for (i = 0; i<l_rDynamicBoxContext.getInputChunkCount(1); i++) {

        if (m_ui64LastTime >= l_rDynamicBoxContext.getInputChunkStartTime(1, i)) {

                ip_pTargetMemoryBuffer=l_rDynamicBoxContext.getInputChunk(1, i);
                m_pTargetStimulationDecoder->process();

                if(m_pTargetStimulationDecoder->isOutputTriggerActive
                (OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedBuffer)) {

                        IStimulationSet* l_pStimulationSet=op_pTargetStimulationSet;

                        for (j = 0; j < l_pStimulationSet->getStimulationCount(); j++) {

                                uint64 l_ui64StimulationIdentifier=l_pStimulationSet->getStimulationIdentifier(j);
                                bool l_bTarget = false;

                                if (l_ui64StimulationIdentifier >= m_ui64RowStimulationBase && l_ui64StimulationIdentifier
                                < m_ui64RowStimulationBase + m_ui64RowCount) {

                                        this->getLogManager() << LogLevel_Debug << "Received Target Row " <<
                                        l_ui64StimulationIdentifier << "\n";
                                        m_iTargetRow = (int)(l_ui64StimulationIdentifier -  m_ui64RowStimulationBase);

                                        l_bTarget = true;
                                }
                                if (l_ui64StimulationIdentifier >= m_ui64ColumnStimulationBase &&
                                l_ui64StimulationIdentifier < m_ui64ColumnStimulationBase + m_ui64ColumnCount) {

                                        this->getLogManager() << LogLevel_Debug << "Received Target Column" <<
                                        l_ui64StimulationIdentifier << "\n";
                                        m_iTargetColumn = (int)(l_ui64StimulationIdentifier - m_ui64ColumnStimulationBase);
```

```
        l_bTarget=true;
}

if (l_bTarget && m_iTargetRow != -1 && m_iTargetColumn != -1) {

        this->getLogManager() << LogLevel_Debug << "Displays Target Cell\n";

        // Llamadas a la función que realiza cambios en la tabla de los labels
        this->_cache_for_each_if_aux_(
                m_iTargetRow,
                m_iTargetColumn,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_aux_,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_null_cb_,
                &m_oTargetBackgroundColor,
                NULL);
        this->_cache_for_each_if_aux_(
                m_iTargetRow,
                m_iTargetColumn,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_foreground_cb_aux_,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_null_cb_,
                &m_oTargetForegroundColor,
                NULL);
        this->_cache_for_each_if_aux_(
                m_iTargetRow,
                m_iTargetColumn,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_font_cb_aux_,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_null_cb_,
                m_pTargetFontDescription,
                NULL);

        // Llamada a la funcion que realiza cambios en el fondo de las celdas de la tabla de
        imágenes.
        this->_cache_for_each_if_(
                m_iTargetRow,
                m_iTargetColumn,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_,
                &CBoxAlgorithmP300SpellerVisualisation::_cache_change_null_cb_,
                &m_oTargetBackgroundColor,
                NULL);

        std::vector < ::GtkWidget* > l_vWidgets;

        // Llamadas a la función que realiza cambios en la tabla de los labels
        this->_cache_for_each_if_aux_(
```

```
                                        m_iTargetRow,
                                        m_iTargetColumn,
                                        &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
                                        &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
                                        &l_vWidgets,
                                        NULL);

                                {
                                        m_vStimuliQueue.push_back(m_iTargetRow + m_ui64RowStimulationBase);
                                        m_vStimuliQueue.push_back(m_iTargetColumn + m_ui64ColumnStimulationBase);
                                }

                                if (l_vWidgets.size() == 1) {


                                        if (GTK_IS_LABEL(l_vWidgets[0])) {

                                                std::string l_sString;
                                                l_sString = gtk_label_get_text(m_pTarget);
                                                l_sString += gtk_label_get_text(GTK_LABEL(l_vWidgets[0]));
                                                gtk_label_set_text(m_pTarget, l_sString.c_str());
                                        }
                                        else {
                                                this->getLogManager() << LogLevel_Warning << "Expected label class
                                                widget... could not find a valid text to append\n";
                                        }
                                }
                                else {
                                        this->getLogManager() << LogLevel_Warning << "Did not find a unique widget at
                                        row:" << (uint32)m_iTargetRow << " column:" << (uint32) m_iTargetColumn <<
                                        "\n";
                                }

                                m_vTargetHistory.push_back(std::make_pair(m_iTargetRow, m_iTargetColumn));
                                m_iLastTargetRow=m_iTargetRow;
                                m_iLastTargetColumn=m_iTargetColumn;
                                m_iTargetRow=-1;
                                m_iTargetColumn=-1;
                        }
                }
        }

        l_rDynamicBoxContext.markInputAsDeprecated(1, i);
    }
}
```

```
// --- Selection stimulations
for (k = 2; k < 4; k++) {

        IAlgorithmProxy* l_pSelectionStimulationDecoder = (k == 2 ? m_pRowSelectionStimulationDecoder:
        m_pColumnSelectionStimulationDecoder);
        TParameterHandler < const IMemoryBuffer* > ip_pSelectionMemoryBuffer(l_pSelectionStimulationDecoder->
        getInputParameter(OVP_GD_Algorithm_StimulationStreamDecoder_InputParameterId_MemoryBufferToDecode));
        TParameterHandler < IStimulationSet* > op_pSelectionStimulationSet(l_pSelectionStimulationDecoder->
        getOutputParameter(OVP_GD_Algorithm_StimulationStreamDecoder_OutputParameterId_StimulationSet));

        for (i = 0; I < l_rDynamicBoxContext.getInputChunkCount(k); i++) {

                if (m_ui64LastTime >= l_rDynamicBoxContext.getInputChunkStartTime(k, i)) {

                        ip_pSelectionMemoryBuffer = l_rDynamicBoxContext.getInputChunk(k, i);
                        l_pSelectionStimulationDecoder->process();

                        if (l_pSelectionStimulationDecoder->isOutputTriggerActive
                        (OVP_GD_Algorithm_StimulationStreamDecoder_OutputTriggerId_ReceivedBuffer)) {

                                IStimulationSet* l_pStimulationSet=op_pSelectionStimulationSet;

                                for (j = 0; j < l_pStimulationSet->getStimulationCount(); j++) {

                                        uint64 l_ui64StimulationIdentifier = l_pStimulationSet->getStimulationIdentifier(j);
                                        bool l_bSelected = false;

                                        if (l_ui64StimulationIdentifier >= m_ui64RowStimulationBase &&
                                        l_ui64StimulationIdentifier < m_ui64RowStimulationBase + m_ui64RowCount) {

                                                this->getLogManager() << LogLevel_Debug << "Received Selected Row " <<
                                                l_ui64StimulationIdentifier << "\n";

                                                m_iSelectedRow = (int)(l_ui64StimulationIdentifier -
                                                m_ui64RowStimulationBase);
                                                l_bSelected = true;
                                        }
                                        if (l_ui64StimulationIdentifier >= m_ui64ColumnStimulationBase &&
                                        l_ui64StimulationIdentifier < m_ui64ColumnStimulationBase + m_ui64RowCount) {

                                                this->getLogManager() << LogLevel_Debug << "Received Selected Column " <<
                                                l_ui64StimulationIdentifier << "\n";
```

```
        m_iSelectedColumn = (int)(l_ui64StimulationIdentifier -
        m_ui64ColumnStimulationBase);
        l_bSelected = true;
}
if (l_ui64StimulationIdentifier == OVTK_StimulationId_Label_00) {

        if (k == 2) m_iSelectedRow = -2;
        if (k == 3) m_iSelectedColumn = -2;
        l_bSelected = true;
}
if (l_bSelected && m_iSelectedRow != -1 && m_iSelectedColumn != -1) {

        if(m_iSelectedRow>=0 && m_iSelectedColumn>=0) {

                this->getLogManager() << LogLevel_Debug << "Displays Selected Cell\n";

                // Llamadas a la funcion que realiza cambios en la tabla de los labels.
                this->_cache_for_each_if_aux_(
                        m_iSelectedRow,
                        m_iSelectedColumn,
                        &CBoxAlgorithmP300SpellerVisualisation::
                        _cache_change_background_cb_aux_,
                        &CBoxAlgorithmP300SpellerVisualisation::
                        _cache_change_null_cb_,
                        &m_oSelectedBackgroundColor,
                        NULL);
                this->_cache_for_each_if_aux_(
                        m_iSelectedRow,
                        m_iSelectedColumn,
                        &CBoxAlgorithmP300SpellerVisualisation::
                        _cache_change_foreground_cb_aux_,
                        &CBoxAlgorithmP300SpellerVisualisation::
                        _cache_change_null_cb_,
                        &m_oSelectedForegroundColor,
                        NULL);
                this->_cache_for_each_if_aux_(
                        m_iSelectedRow,
                        m_iSelectedColumn,
                        &CBoxAlgorithmP300SpellerVisualisation::
                        _cache_change_font_cb_aux_,
                        &CBoxAlgorithmP300SpellerVisualisation::
                        _cache_change_null_cb_,
                        m_pSelectedFontDescription,
                        NULL);
```

```
// Llamada a la funcion que realiza cambios en el fondo de las celdas
de la tabla de imagenes.
this->_cache_for_each_if_(
       m_iSelectedRow,
       m_iSelectedColumn,
       &CBoxAlgorithmP300SpellerVisualisation::
       _cache_change_background_cb_,
       &CBoxAlgorithmP300SpellerVisualisation::
       _cache_change_null_cb_,
       &m_oSelectedBackgroundColor,
       NULL);

std::vector < ::GtkWidget* > l_vWidgets;

// Llamadas a la funcion que realiza cambios en la tabla de los labels.
this->_cache_for_each_if_aux_(
       m_iSelectedRow,
       m_iSelectedColumn,
       &CBoxAlgorithmP300SpellerVisualisation::
       _cache_collect_child_widget_cb_,
       &CBoxAlgorithmP300SpellerVisualisation::
       _cache_collect_child_widget_cb_,
       &l_vWidgets,
       NULL);

if (l_vWidgets.size() == 1) {

       if (GTK_IS_LABEL(l_vWidgets[0])) {

              std::string l_sString;
              l_sString = gtk_label_get_text(GTK_LABEL
              (l_vWidgets[0]));

              if(m_vTargetHistory.size()) {

                     std::list < std::pair < int, int >>::
                     const_iterator it = m_vTargetHistory.begin();

                     bool l_bCorrect = (it->first ==
                     m_iSelectedRow &&     it-> second ==
                     m_iSelectedColumn);
                     bool l_bHalfCorrect = (it->first ==
                     m_iSelectedRow || it-> second ==
                     m_iSelectedColumn);
                     m_vTargetHistory.pop_front();
```

```
                                                                }

                                                                l_sString = std::string(gtk_label_get_label
                                                                (m_pResult)) + l_sString;
                                                                gtk_label_set_markup(m_pResult, l_sString.c_str());
                                                        }
                                                        else {
                                                                this->getLogManager() << LogLevel_Warning <<
                                                                "Expected label class widget... could not find a valid
                                                                text to append\n";
                                                        }
                                                }
                                                else {
                                                        this->getLogManager() << LogLevel_Warning << "Did not find a
                                                        widget at row : " << (uint32)m_iSelectedRow << " column : " <<
                                                        (uint32)m_iSelectedColumn << "\n";
                                                }
                                        }
                                        else {
                                                this->getLogManager() << LogLevel_Trace << "Selection Rejected !\n";

                                                std::string l_sString;
                                                l_sString = gtk_label_get_text(m_pResult);
                                                l_sString += "*";
                                                gtk_label_set_text(m_pResult, l_sString.c_str());
                                        }

                                        m_iSelectedRow = -1;
                                        m_iSelectedColumn = -1;
                                }
                        }
                }

                l_rDynamicBoxContext.markInputAsDeprecated(k, i);
        }
    }

    return true;
}
```

```
/*****************************************************************************************************/
/** Método _cache_build_from_table_(::GtkTable* pTable) -Speller Visualisation-                     **/
/** Se encarga de formar la tabla de imágenes de letras.                                            **/
/*****************************************************************************************************/
	void CBoxAlgorithmP300SpellerVisualisation::_cache_build_from_table_(::GtkTable* pTable) {

		// Construcción de la tabla de imágenes.
		if(pTable) {

			::GtkTableChild* l_pTableChild = NULL;
			::GList* l_pList = NULL;

			for (l_pList = pTable->children; l_pList; l_pList = l_pList->next) {

				l_pTableChild=(::GtkTableChild*)l_pList->data;

				for(unsigned long i = l_pTableChild->top_attach; i<l_pTableChild->bottom_attach; i++){

					for(unsigned long j = l_pTableChild->left_attach; j<l_pTableChild->right_attach; j++) {

						CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle& l_rWidgetStyle = m_vCache[i][j];

						l_rWidgetStyle.pParent = l_pTableChild->widget;
						l_rWidgetStyle.pWidget = gtk_bin_get_child(GTK_BIN(l_rWidgetStyle.pParent));

						l_rWidgetStyle.pImage = GTK_IMAGE(l_rWidgetStyle.pWidget);

						l_rWidgetStyle.oBackgroundColor=l_oWhite;
						l_rWidgetStyle.oForegroundColor=l_oWhite;
						l_rWidgetStyle.pFontDescription=NULL;
					}
				}
			}
		}
	}

/*****************************************************************************************************/
/** Método _cache_build_from_aux_table_(::GtkTable* pTable)-Speller Visualisation-                  **/
/** Se encarga de formar la tabla de imágenes de panales.                                           **/
/*****************************************************************************************************/
	void CBoxAlgorithmP300SpellerVisualisation::_cache_build_from_aux_table_(::GtkTable* pTable) {

		// Construcción de la tabla auxiliar de labels.
		if (pTable) {
```

```
                        ::GtkTableChild* l_pTableChild = NULL;
                        ::GList* l_pList = NULL;

                        for (l_pList = pTable->children; l_pList; l_pList = l_pList->next) {

                                l_pTableChild = (::GtkTableChild*)l_pList->data;

                                for (unsigned long i = l_pTableChild->top_attach; i<l_pTableChild->bottom_attach; i++){

                                        for (unsigned long j = l_pTableChild->left_attach; j<l_pTableChild->right_attach; j++) {

                                                CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle& l_rWidgetStyle = m_vCacheAux[i][j];
                                                l_rWidgetStyle.pWidget = l_pTableChild->widget;
                                                l_rWidgetStyle.pChildWidget = gtk_bin_get_child(GTK_BIN(l_pTableChild->widget));
                                                l_rWidgetStyle.oBackgroundColor = l_oWhite;
                                                l_rWidgetStyle.oForegroundColor = l_oWhite;
                                                l_rWidgetStyle.pFontDescription = NULL;
                                        }
                                }
                        }
                }
        }

/*************************************************************************************************/
/** Método _cache_for_each_(_cache_callback_ fpCallback, void* pUserData) -Speller Visualisation-        **/
/** Se encarga de realizar cambios en la tabla de imágenes de letras.                                    **/
/*************************************************************************************************/
        void CBoxAlgorithmP300SpellerVisualisation::_cache_for_each_(_cache_callback_ fpCallback, void* pUserData) {

                std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle > >::iterator i;
                std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;

                for (i = m_vCache.begin(); i != m_vCache.end(); i++) {

                        for(j = i->second.begin(); j!=i->second.end(); j++) {

                                (this->*fpCallback)(j->second, pUserData);
                        }
                }
        }
```

```
/**************************************************************************************************/
/** Método _cache_for_each_aux_(_cache_callback_ fpCallback, void* pUserData) -Speller          **/
/** Visualisation-                                                                              **/
/** Se encarga de realizar cambios en la tabla de imágenes de panales.                          **/
/**************************************************************************************************/
	void CBoxAlgorithmP300SpellerVisualisation::_cache_for_each_aux_(_cache_callback_ fpCallback, void* pUserData) {

		std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle > >::iterator i;
		std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;

		for (i = m_vCacheAux.begin(); i != m_vCacheAux.end(); i++) {

			for (j = i->second.begin(); j != i->second.end(); j++) {

				(this->*fpCallback)(j->second, pUserData);
			}
		}
	}


/**************************************************************************************************/
/** Método _cache_for_each_if_(int iLine, int iColumn, _cache_callback_ fpIfCallback,           **/
/** _cache_callback_ fpElseCallback, void* pIfUserData, void* pElseUserData) -Speller Visualisation-  **/
/** Se encarga de realizar cambios en la tabla de imágenes de letras.                           **/
/**************************************************************************************************/
	void CBoxAlgorithmP300SpellerVisualisation::_cache_for_each_if_(int iLine, int iColumn, _cache_callback_fpIfCallback,
	_cache_callback_ fpElseCallback, void* pIfUserData, void*  pElseUserData) {

		std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle > >::iterator i;
		std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;

		for (i = m_vCache.begin(); i != m_vCache.end(); i++) {
			for(j=i->second.begin(); j!=i->second.end(); j++) {

				bool l_bLine = (iLine != -1);
				bool l_bColumn = (iColumn != -1);
				bool l_bInLine = false;
				bool l_bInColumn = false;
				bool l_bIf;

				if (l_bLine && (unsigned long)iLine == i->first) {

					l_bInLine = true;
				}
				if (l_bColumn && (unsigned long)iColumn == j->first) {
```

```
                                l_bInColumn = true;
                        }

                        if (l_bLine && l_bColumn) {

                                l_bIf = l_bInLine && l_bInColumn;
                        }
                        else {
                                l_bIf = l_bInLine || l_bInColumn;
                        }

                        if (l_bIf) {
                                (this->*fpIfCallback)(j->second, pIfUserData);
                        }
                        else {
                                (this->*fpElseCallback)(j->second, pElseUserData);
                        }
                }
        }
}

/*****************************************************************************************************/
/** Método _cache_for_each_if_aux_(int iLine, int iColumn, _cache_callback_ fpIfCallback,           **/
/** _cache_callback_ fpElseCallback, void* pIfUserData, void* pElseUserData) -Speller Visualisation- **/
/** Se encarga de realizar cambios en la tabla de imágenes de panales.                              **/
/*****************************************************************************************************/
void CBoxAlgorithmP300SpellerVisualisation::_cache_for_each_if_aux_(int iLine, int iColumn, _cache_callback_ fpIfCallback,
_cache_callback_ fpElseCallback, void* pIfUserData, void* pElseUserData) {

        std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle > >::iterator i;
        std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;

        for (i = m_vCacheAux.begin(); i != m_vCacheAux.end(); i++) {

                for (j = i->second.begin(); j != i->second.end(); j++) {

                        bool l_bLine = (iLine != -1);
                        bool l_bColumn = (iColumn != -1);
                        bool l_bInLine = false;
                        bool l_bInColumn = false;
                        bool l_bIf;

                        if (l_bLine && (unsigned long)iLine == i->first) {
```

```
                                l_bInLine = true;
                        }
                        if (l_bColumn && (unsigned long)iColumn == j->first) {

                                l_bInColumn = true;
                        }

                        if (l_bLine && l_bColumn) {

                                l_bIf = l_bInLine && l_bInColumn;
                        }
                        else {
                                l_bIf = l_bInLine || l_bInColumn;
                        }

                        if (l_bIf) {

                                (this->*fpIfCallback)(j->second, pIfUserData);
                        }
                        else {
                                (this->*fpElseCallback)(j->second, pElseUserData);
                        }
                }
        }
}

/*****************************************************************************************************/
/** Método _cache_for_each_if_image_(int iLine, int iColumn, _cache_callback_image_ fpIfCallback,   **/
/** _cache_callback_image_ fpElseCallback, void* pIfUserData, void* pElseUserData) -Speller Visualisation- **/
/** Se encarga de realizar cambios en las imágenes.                                                 **/
/*****************************************************************************************************/
void CBoxAlgorithmP300SpellerVisualisation::_cache_for_each_if_image_(int iLine, int iColumn, _cache_callback_image_
fpIfCallback, _cache_callback_image_ fpElseCallback, void* pIfUserData, void* pElseUserData) {

        std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle > >::iterator i;
        std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;

        // Declaración de variables.
        int x = 0, y = 0;

        char image[600];
        char honeyComb_num_redDots[600];
        char honeyComb[600];
        char final_honeyComb[600];
```

```
// Llamada a la función que devuelve cuantos puntos tiene el panal.
int num_redDots = _rand_red_dots_honeyComb_();

// Concatenamos el número de puntos del panal con la palabra panal.
sprintf(honeyComb_num_redDots, "panal%d", num_redDots);

// Llamada a la función que devuelve el número de imágenes que hay para el número de puntos obtenido.
int num_images = _read_directory_(std::string(honeyComb_num_redDots));

// Llamada a la función que devuelve el número de imágenes que hay para el número de puntos obtenido.
int num_honeyComb = _rand_honeyComb_(num_images);

// Concatenamos la palabra con el número de puntos y la imagen elegida del directorio de imágenes.
sprintf(honeyComb, "%s-%.3d", honeyComb_num_redDots, num_honeyComb);

// Concatenamos la dirección del directorio de imágenes con el resto de la palabra y le añadimos la extensión de la imagen.
sprintf(final_honeyComb, "%s%s.png", m_cImagesDir, honeyComb);

for (i = m_vCache.begin(); i != m_vCache.end(); i++) {

	y = 0;

	for (j = i->second.begin(); j != i->second.end(); j++) {

		bool l_bLine = (iLine != -1);
		bool l_bColumn = (iColumn != -1);
		bool l_bInLine = false;
		bool l_bInColumn = false;
		bool l_bIf;

		if (l_bLine && (unsigned long)iLine == i->first) {

			l_bInLine = true;
		}
		if (l_bColumn && (unsigned long)iColumn == j->first) {

			l_bInColumn = true;
		}

		if (l_bLine && l_bColumn) {
			l_bIf = l_bInLine && l_bInColumn;
		}
		else {
			l_bIf = l_bInLine || l_bInColumn;
```

```
                        }

                        if (l_bIf) {

                                (this->*fpIfCallback)(j->second, pIfUserData, std::string(final_honeyComb));
                        }
                        else {
                                std::vector < ::GtkWidget* > l_vWidgets;

                                // Llamadas a la función que realiza cambios en la tabla de los labels.
                                this->_cache_for_each_if_aux_(
                                        x,
                                        y,
                                        &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
                                        &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
                                        &l_vWidgets,
                                        NULL);

                                std::string l_sString;
                                l_sString = gtk_label_get_text(GTK_LABEL(l_vWidgets[0]));

                                sprintf(image, "%sletra-%s.png", m_cImagesDir, l_sString.c_str());

                                (this->*fpElseCallback)(j->second, pElseUserData, std::string(image));
                        }
                        y++;
                }
                x++;
        }
}

/*************************************************************************************************/
/** Método _cache_change_background_cb_ (CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle&   **/
/** rWidgetStyle, void* pUserData) -Speller Visualisation-                                       **/
/** Se encarga de cambiar el fondo de la tabla de imágenes de letras.                            **/
/*************************************************************************************************/
void CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_ (CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle&
rWidgetStyle, void* pUserData) {

        ::GdkColor oColor=*(::GdkColor*)pUserData;

        if (!System::Memory::compare(&rWidgetStyle.oBackgroundColor, &oColor, sizeof(::GdkColor))) {

                gtk_widget_modify_bg(rWidgetStyle.pParent, GTK_STATE_NORMAL, &oColor);
                rWidgetStyle.oBackgroundColor=oColor;
```

```
		}
	}

/*************************************************************************************************/
/** Método _cache_change_background_cb_aux_(CBoxAlgorithmP300SpellerVisualisation::             **/
/** SWidgetStyle& rWidgetStyle, void* pUserData) -Speller Visualisation-                        **/
/** Se encarga de cambiar el fondo de la tabla de imágenes de panales.                          **/
/*************************************************************************************************/
	void		CBoxAlgorithmP300SpellerVisualisation::_cache_change_background_cb_aux_(CBoxAlgorithmP300SpellerVisualisation::
	SWidgetStyle& rWidgetStyle, void* pUserData) {

		::GdkColor oColor = *(::GdkColor*)pUserData;

		if (!System::Memory::compare(&rWidgetStyle.oBackgroundColor, &oColor, sizeof(::GdkColor))) {

			gtk_widget_modify_bg(rWidgetStyle.pWidget, GTK_STATE_NORMAL, &oColor);
			rWidgetStyle.oBackgroundColor = oColor;
		}
	}

/*************************************************************************************************/
/** Método _reset_images_(_cache_callback_image_ fpCallback, void* pUserData) -Speller Visualisation-  **/
/** Se encarga de eliminar la imagen que se ha mostrado.                                        **/
/*************************************************************************************************/
	void CBoxAlgorithmP300SpellerVisualisation::_reset_images_(_cache_callback_image_ fpCallback, void* pUserData) {

		std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle > >::iterator i;
		std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;

		// Declaración de variables.
		int x = 0, y = 0;
		char image[600];

		for (i = m_vCache.begin(); i != m_vCache.end(); i++) {

			y = 0;

			for (j = i->second.begin(); j != i->second.end(); j++) {

				std::vector < ::GtkWidget* > l_vWidgets;

				// Llamadas a la función que realiza cambios en la tabla de los labels.
				this->_cache_for_each_if_aux_(
					x,
					y,
```

```
                                &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
                                &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
                                &l_vWidgets,
                                NULL);

                        std::string l_sString;
                        l_sString = gtk_label_get_text(GTK_LABEL(l_vWidgets[0]));

                        sprintf(image, "%sletra-%s.png", m_cImagesDir, l_sString.c_str());
                        (this->*fpCallback)(j->second, pUserData, std::string(image));

                        y++;
                }
                x++;
        }
}

/*****************************************************************************************************/
/** Método _cache_set_image_(CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle& rWidgetStyle,     **/
/** void*pUserData, std::string l_sString) -Speller Visualisation-                                   **/
/** Se encarga de seleccionar la imagen que se va a mostrar.                                        **/
/*****************************************************************************************************/
void CBoxAlgorithmP300SpellerVisualisation::_cache_set_image_(CBoxAlgorithmP300SpellerVisualisation::  SWidgetStyle&
rWidgetStyle, void*pUserData, std::string l_sString) {

        gtk_image_set_from_file(rWidgetStyle.pImage, l_sString.c_str());
}

/*****************************************************************************************************/
/** Método _read_directory_(const std::string& name) -Speller Visualisation-                        **/
/** Se encarga de obtener las imágenes del directorio de imágenes.                                  **/
/*****************************************************************************************************/
int CBoxAlgorithmP300SpellerVisualisation::_read_directory_(const std::string& name) {

        WIN32_FIND_DATA data;
        HANDLE hFind;

        int count = 0;

        sprintf(m_cImagesDir, "%s", "..\\dist\\extras\\Debug\\share\\openvibe\\plugins\\simple-visualisation\\");

        std::string pattern(m_cImagesDir);
        pattern.append(name);
        pattern.append("*");
```

```
		if ((hFind = FindFirstFile(pattern.c_str(), &data)) != INVALID_HANDLE_VALUE) {
			do {
				count++;

			}while (FindNextFile(hFind, &data) != 0);

			FindClose(hFind);
		}

		return count;
	}

/**********************************************************************************************************/
/** Método _rand_red_dots_honeyComb_(void) -Speller Visualisation-                                       **/
/** Se encarga de obtener el número aleatorio de puntos rojos que hay distribuidos en el panal.          **/
/**********************************************************************************************************/
	int CBoxAlgorithmP300SpellerVisualisation:: _rand_red_dots_honeyComb_(void) {

		srand(unsigned int(time(NULL)));
		return (1 + rand() % m_uiNumMaxHoneyComb);
	}

/**********************************************************************************************************/
/** Método _rand_honeyComb_(int num_honeyComb) -Speller Visualisation-                                   **/
/** Se encarga de obtener la imagen del panal con los puntos aleatoriamente distribuidos.                **/
/**********************************************************************************************************/
	int CBoxAlgorithmP300SpellerVisualisation::_rand_honeyComb_(int num_honeyComb) {

		srand(unsigned int(time(NULL)));
		return (1 + rand() % num_honeyComb);
	}
```

3. Honeycomb-Shaped Red Dots – Checkerboard Paradigm

```
/*-------------------------------------------------------------------------------------------------------------*/
/*                      P300 SPELLER - Honeycomb-Shaped Red Dots - Checkerboard Paradigm                        */
/*-------------------------------------------------------------------------------------------------------------*/
/* Aplicación que permite al usuario escribir caracteres utilizando solamente las ondas cerebrales.            */
/* Para ello se utiliza el paradigma HSRDCBP.                                                                  */
/*                                                                                          Fecha: 6/07/2018 */
/*                                                                                Cristina Cantalapiedra Cabezas */
/*-------------------------------------------------------------------------------------------------------------*/

/***************************************************************************************************************/
/** Método _cache_build_from_aux_table_(::GtkTable* pTable) -Speller Visualisation-                           **/
/** Se encarga de obtener la tabla de imagenes de panales.                                                    **/
/***************************************************************************************************************/
      void CBoxAlgorithmP300SpellerVisualisation::_cache_build_from_aux_table_(::GtkTable* pTable) {

            // Declaración e inicialización de las variables que voy a utilizar en los bucles.
            int x = 0, y = 0, z = 0, t = 0;

            // Declaración de las variables de tipo string que contienen las matrices blanca y negra.
            std::string whiteTable, blackTable;

            // Construcción de la tabla auxiliar de labels.
            if (pTable) {

                  // Declaración e inicialización de la tabla hijo que hereda de la tabla padre (matriz pTable). La tabla hijo es
                  para acceder a cada celda de la matriz (label y demás parámetros).
                  ::GtkTableChild* l_pTableChild = NULL;

                  // Declaración e inicialización de una lista.
                  ::GList* l_pList = NULL;

                  // Bucle que recorre la lista, desde el valor que tenga el parámetro child de pTable hasta que no haya más valores.
                  for (l_pList = pTable->children; l_pList; l_pList = l_pList->next) {

                        // La tabla hijo va a contener los datos de la lista que estamos recorriendo. Se pasa la variable a tipo
                        GtkTableChild ya que queremos que sea parte de la tabla hijo creada.
                        l_pTableChild = (::GtkTableChild*)l_pList->data;

                        // Bucle que recorre la tabla hijo de arriba hacia abajo.
                        for (unsigned long i = l_pTableChild->top_attach; i<l_pTableChild->bottom_attach; i++) {
```

```
// Bucle que recorre la tabla de izquierda a derecha. De esta forma, se están cogiendo todas las
celdas pertenecientes a la tabla hijo.
for (unsigned long j = l_pTableChild->left_attach; j<l_pTableChild->right_attach; j++) {

        // Se obtiene cada celda de la tabla padre mediante m_vCache y se guarda en la variable
        l_rWidgetStyle.
        CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle& l_rWidgetStyle = m_vCacheAux[i][j];

        // Se asigna al parámetro pWidget de la variable en la que se están guardando las celdas de
        la tabla (l_rWidgetStyle) el valor que tenga el widget en la tabla hijo.
        l_rWidgetStyle.pWidget = l_pTableChild->widget;

        // Se asigna al parámetro pChildWidget (label), el valor binario obtenido de acceder al
        widget de la tabla hijo. Para ello, se pasa el valor obtenido a tipo GTK_BIN.
        l_rWidgetStyle.pChildWidget = gtk_bin_get_child(GTK_BIN(l_pTableChild->widget));

        // Declaración de una variable de tipo string.
        std::string l_sString;

        // Se asigna a la variable declarada l_sString, el valor binario que se ha guardado en la
        variable pChildWidget. Para ello, se pasa el valor obtenido a tipo GTK_LABEL. Se obtiene el
        label de la letra que se está iluminando.
        l_sString = gtk_label_get_text(GTK_LABEL(l_rWidgetStyle.pChildWidget));
        m_cTable[i][j] = l_sString.at(0);

        // Se crean los vectores bidimensionales para las matrices blanca y negra. Para las celdas
        de la matriz   blanca:
        if ((i + j) % 2 == 0) {

                // Se añade a la matriz blanca, la letra de la tabla guardada en la variable de tipo
                string.
                whiteTable.append(l_sString);

                // Necesito que la matriz que estoy creando sea de 6x3.
                if (y == 3) {

                        // Pongo la columna a 0 y añado una fila mas.
                        y = 0;
                        x++;
                }
        }
        // Para las celdas de la matriz negra:
        else {
```

```
                                        // Se añade a la matriz negra, la letra de la tabla guardada en la variable de tipo
                                        string.
                                        blackTable.append(l_sString);

                                        // Necesito que la matriz que estoy creando sea de 6x3.
                                        if (t == 3) {

                                                // Pongo la columna a 0 y añado una fila mas.
                                                t = 0;
                                                z++;
                                        }
                                }

                                l_rWidgetStyle.oBackgroundColor = l_oWhite;
                                l_rWidgetStyle.oForegroundColor = l_oWhite;
                                l_rWidgetStyle.pFontDescription = NULL;
                        }
                }
        }

        // Declaración de variables para distribuir las letras dentro de las matrices blanca y negra de forma aleatoria.
        std::random_shuffle(whiteTable.begin(), whiteTable.end());
        std::random_shuffle(blackTable.begin(), blackTable.end());

        // Bucle que recorre tanto la matriz negra como la matriz blanca, para distribuir de forma aleatoria las letras.
        for (x = 0; x < 6; x++) {

                for (y = 0; y < 3; y++) {

                        // Se asigna a la matriz el label que se ha elegido de forma aleatoria. Para ello, se utiliza .at
                        que da como resultado un tipo char.
                        m_cBlackTable[x][y] = blackTable.at(0);
                        m_cWhiteTable[x][y] = whiteTable.at(0);

                        // Se borra el label de la matriz para seguir distribuyendo dentro de las matrices blanca y negra.
                        blackTable.erase(0, 1);
                        whiteTable.erase(0, 1);
                }
        }
    }
}
```

```
/*********************************************************************************************************/
/** Método __cache_for_each_if_image_color_(int iLine, int iColumn, _cache_callback_image_              **/
/** fpIfCallback, _cache_callback_image_ fpElseCallback, void* pIfUserData, void* pElseUserData,        **/
/** bool l_bWhiteTable) -Speller Visualisation-                                                          **/
/** Se encarga de realizar cambios en las imágenes de las matrices blanca y negra.                      **/
/*********************************************************************************************************/
	void CBoxAlgorithmP300SpellerVisualisation::_cache_for_each_if_image_color_(int iLine, int iColumn,     _cache_callback_image_
	fpIfCallback, _cache_callback_image_ fpElseCallback, void* pIfUserData, void*    pElseUserData, bool l_bWhiteTable) {

		std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle > >::iterator i;
		std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;

		bool l_bLine = (iLine != -1);
		bool l_bColumn = (iColumn != -1);

		std::string lettersColumnColor;
		std::string lettersRowColor;

		if (l_bWhiteTable) {

			if (l_bLine) {

				for (int i = 0; i < m_ui64ColumnCountNew; i++) {

					lettersRowColor.push_back(m_cWhiteTable[iLine][i]);
				}
			}
			if (l_bColumn) {

				for (int i = 0; i < m_ui64RowCountNew; i++) {

					lettersColumnColor.push_back(m_cWhiteTable[i][iColumn]);
				}
			}
		}
		else {
			if (l_bLine) {

				for (int i = 0; i < m_ui64ColumnCountNew; i++) {

					lettersRowColor.push_back(m_cBlackTable[iLine][i]);
				}
			}
			if (l_bColumn) {
```

```
                for (int i = 0; i < m_ui64RowCountNew; i++) {

                        lettersColumnColor.push_back(m_cBlackTable[i][iColumn]);
                }
        }

}

// Declaración de variables.
int x = 0, y = 0;

char image[600];
char honeyComb_num_redDots[600];
char honeyComb[600];
char final_honeyComb[600];

// Llamada a la función que devuelve cuantos puntos tiene el panal.
int num_redDots = _rand_red_dots_honeyComb_();

// Concatenamos el número de puntos del panal con la palabra panal.
sprintf(honeyComb_num_redDots, "panal%d", num_redDots);

// Llamada a la función que devuelve el número de imágenes que hay para el número de puntos obtenido.
int num_images = _read_directory_(std::string(honeyComb_num_redDots));

// Llamada a la función que devuelve el número de imágenes que hay para el número de puntos obtenido.
int num_honeyComb = _rand_honeyComb_(num_images);

// Concatenamos la palabra con el número de puntos y la imagen elegida del directorio de imágenes.
sprintf(honeyComb, "%s-%.3d", honeyComb_num_redDots, num_honeyComb);

// Concatenamos la dirección del directorio de imágenes con el resto de la palabra y le añadimos la extensión de la imagen.
sprintf(final_honeyComb, "%s%s.png", m_cImagesDir, honeyComb);

for (i = m_vCache.begin(); i != m_vCache.end(); i++) {

        y = 0;

        for (j = i->second.begin(); j != i->second.end(); j++) {

                bool l_bIf;

                if (lettersRowColor.find(m_cTable[(int)x][(int)y]) != std::string::npos || lettersColumnColor.find
                (m_cTable[(int)x][(int)y]) != std::string::npos) {
```

```
                        l_bIf = true;
                    }
                    else {
                        l_bIf = false;
                    }

                    if (l_bIf) {

                        (this->*fpIfCallback)(j->second, pIfUserData, std::string(final_honeyComb));
                    }
                    else {
                        std::vector < ::GtkWidget* > l_vWidgets;

                        // Llamadas a la funcion que realiza cambios en la tabla de los labels.
                        this->_cache_for_each_if_aux_(
                            x,
                            y,
                            &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
                            &CBoxAlgorithmP300SpellerVisualisation::_cache_collect_child_widget_cb_,
                            &l_vWidgets,
                            NULL);

                        std::string l_sString;
                        l_sString = gtk_label_get_text(GTK_LABEL(l_vWidgets[0]));

                        sprintf(image, "%sletra-%s.png", m_cImagesDir, l_sString.c_str());

                        (this->*fpElseCallback)(j->second, pElseUserData, std::string(image));
                    }
                    y++;
                }
                x++;
            }
        }
```

```
/**************************************************************************************************/
/** Método _cache_for_each_if_aux_color_(int iLine, int iColumn, _cache_callback_ fpIfCallback,  **/
/** fpElseCallback, void* pIfUserData, void* pElseUserData, bool l_bWhiteTable)                   **/
/** -Speller Visualisation-                                                                       **/
/** Se encarga de realizar cambios en las matrices blanca y negra.                                **/
/**************************************************************************************************/
	void CBoxAlgorithmP300SpellerVisualisation::_cache_for_each_if_aux_color_(int iLine, int iColumn, _cache_callback_ fpIfCallback,
	_cache_callback_ fpElseCallback, void* pIfUserData, void*  pElseUserData, bool l_bWhiteTable) {

		std::map < unsigned long, std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation:: SWidgetStyle > >::iterator i;
		std::map < unsigned long, CBoxAlgorithmP300SpellerVisualisation::SWidgetStyle >::iterator j;

		bool l_bLine = (iLine != -1);
		bool l_bColumn = (iColumn != -1);

		std::string lettersColumnColor;
		std::string lettersRowColor;

		if (l_bWhiteTable) {

			if (l_bLine) {

				for (int i = 0; i < m_ui64ColumnCountNew; i++) {

					lettersRowColor.push_back(m_cWhiteTable[iLine][i]);
				}
			}
			if (l_bColumn) {
				for (int i = 0; i < m_ui64RowCountNew; i++) {

					lettersColumnColor.push_back(m_cWhiteTable[i][iColumn]);
				}
			}
		}
		else {
			if (l_bLine) {

				for (int i = 0; i < m_ui64ColumnCountNew; i++) {

					lettersRowColor.push_back(m_cBlackTable[iLine][i]);
				}
			}
			if (l_bColumn) {

				for (int i = 0; i < m_ui64RowCountNew; i++) {
```

```
                        lettersColumnColor.push_back(m_cBlackTable[i][iColumn]);
                    }
                }
            }

            for (i = m_vCacheAux.begin(); i != m_vCacheAux.end(); i++) {

                for (j = i->second.begin(); j != i->second.end(); j++) {

                    bool l_bIf;

                    std::string l_sString;
                    l_sString = gtk_label_get_text(GTK_LABEL(j->second.pChildWidget));

                    if (lettersRowColor.find(l_sString.at(0)) != std::string::npos || lettersColumnColor.find
                    (l_sString.at(0)) != std::string::npos) {

                        l_bIf = true;
                    }
                    else {
                        l_bIf = false;
                    }

                    if (l_bIf) {

                        (this->*fpIfCallback)(j->second, pIfUserData);
                    }
                    else {
                        (this->*fpElseCallback)(j->second, pElseUserData);
                    }
                }
            }
        }
```

4. Tratamiento con Matlab

```
/*************************************************************************************************************/
/** Método feature_extraction_RCP_HSRD(file, window, windowb, subfs, fir_path, word)                        **/
/** Se encarga de extraer las características de los archivos RCP y HSRD.                                   **/
/*************************************************************************************************************/
       function data_feat = feature_extraction_RCP_HSRD(file, window, windowb, subfs, fir_path, word)

              data = load(file);
              fir = load(fir_path);

              code = [];
              trials = [];
              signal = [];

              w_samples = round(window * (data.samplingFreq / 1000));
              w_baseline = round(windowb * (data.samplingFreq / 1000));

              temp_CAR = pre_car(data);
              temp_FIR = filtfilt(fir.Num, 1, temp_CAR.samples);

              index_start = find(data.stims(:, 2) == 32773);
              index_stop = find(data.stims(:, 2) == 32774);

              j = 1;

              for i = 1:length(index_stop)

              if i == 2
              index_aux(j) == index_stop(i);
              j = j + 1;
                     end
              end

              index_stop = index_aux';

              wo = 50 / (data.samplingFreq / 2);
              bw = wo / 100;
              [b,a] = iirnotch(wo, bw);

              data.samples = filtfilt(b, a, data.samples);

              totall = length(index_start)*12*10;
              signal = NaN(totall, w_samples, 8);
              feat = [];
```

```
        for trial = 1:length(index_start)

                temp = data.stims(index_start(trial):index_stop(trial), :);

                temp_code = temp(temp(:, 2) < 33037 & temp(:, 2) > 33024, 2) - 33025;
                temp_times = temp(temp(:, 2) < 33037 & temp(:, 2) > 33024, 1);

                code = [code; temp_code];
                trials = [trials; trial.*ones(length(temp_code),1)];

                for flash = 1:length(temp_code)

                [c, index] = min(abs(data.sampleTime - temp_times(flash)));
                temp_baseline = data.samples((index - w_baseline):(index - 1), :);

                mb = mean(temp_baseline, 1);
                sb = std(temp_baseline);

                temp_samples = data.samples(index : (index + w_samples - 1), :);
                        temp_samples = (temp_samples - repmat(mb, w_samples, 1))./repmat(sb, w_samples, 1);

                signal(flash, :, :) = temp_samples;

                temp_feat = [];

                for ch = 1:8
                                temp_feat = [temp_feat, resample(temp_samples(:,ch), subfs,
                data.samplingFreq)'];
                end

                feat = [feat; temp_feat];
                end

                [word_code] = Obtain_word_code_RCP_HSRD(word);

                labels = zeros(length(code), 1);
                labels(code == word_code.row(trials)' | code == word_code.col(trials)') = 1;

        end

        data_feat.signal = signal;
        data_feat.feat = feat;
        data_feat.code = code;
        data_feat.trial = trials;
        data_feat.labels = labels;
        data_feat.data.fs = data.samplingFreq;
        data_feat.data.channels = data.channelNames;
end
```

```
/*****************************************************************************************************/
/** Método feature_extraction_RCP_HSRD(file, window, windowb, subfs, fir_path, word)                **/
/** Se encarga de extraer las características de los archivos CBP y HSRDCBP                         **/
/*****************************************************************************************************/
       function data_feat = feature_extraction_CBP_HSRDCBP(file, window, windowb, subfs, fir_path, word, table)

              data = load(file);
              fir = load(fir_path);

              code = [];
              trials = [];
              signal = [];

              w_samples = round(window * (data.samplingFreq / 1000));
              w_baseline = round(windowb * (data.samplingFreq / 1000));

              temp_CAR = pre_car(data);
              temp_FIR = filtfilt(fir.Num, 1, temp_CAR.samples);

              index_start = find(data.stims(:, 2) == 32773);
              index_stop = find(data.stims(:, 2) == 32774);

              wo = 50 / (data.samplingFreq / 2);
              bw = wo / 100;
              [b,a] = iirnotch(wo, bw);

              data.samples = filtfilt(b, a, data.samples);

              totall = length(index_start)*18*10;
              signal = NaN(totall, w_samples, 8);
              feat = [];

              for trial = 1:length(index_start)

                     temp = data.stims(index_start(trial):index_stop(trial), :);
                     temp_code = temp(temp(:, 2) < 33043 & temp(:, 2) > 33024, 2) - 33025;
                     temp_times = temp(temp(:, 2) < 33043 & temp(:, 2) > 33024, 1);

                     code = [code; temp_code];
                     trials = [trials; trial.*ones(length(temp_code),1)];

                     for flash = 1:length(temp_code)

                     [c, index] = min(abs(data.sampleTime - temp_times(flash)));
                     temp_baseline = data.samples((index - w_baseline):(index - 1), :);

                     mb = mean(temp_baseline, 1);
                     sb = std(temp_baseline);
```

```
                temp_samples = data.samples(index : (index + w_samples - 1), :);
                temp_samples = (temp_samples - repmat(mb, w_samples, 1))./repmat(sb, w_samples, 1);

                signal(flash, :, :) = temp_samples;

                temp_feat = [];



                for ch = 1:8
                                temp_feat = [temp_feat, resample(temp_samples(:,ch), subfs,
                data.samplingFreq)'];
                end

                feat = [feat; temp_feat];
        end

                [word_code] = Obtain_word_code_CBP_HSRDCBP(word, table);

                labels = zeros(length(code), 1);
                labels(code == word_code.row(trials)' | code == word_code.col(trials)') = 1;
        end

        data_feat.signal = signal;
        data_feat.feat = feat;
        data_feat.code = code;
        data_feat.trial = trials;
        data_feat.labels = labels;
        data_feat.data.fs = data.samplingFreq;
        data_feat.data.channels = data.channelNames;
end


/*************************************************************************************************/
/** Método Obtain_plot_p300_RCP_HSRD_5words(file1, file2, file3, file4, file5, fir_filter)      **/
/** Se encarga de obtener el P300 de los archivos RCP y HSRD                                    **/
/*************************************************************************************************/
function P300_data_RCP_HSRD = Obtain_plot_p300_RCP_HSRD_5words(file1, file2, file3, file4, file5, fir_filter)

        % Tabla normal 6x6
        % - 33025 Row 0
        % - 33030 Row 5
        % - 33031 Column 0
        % - 33036 Column 5
```

```
% Identificadores
% - 32773 Trial start
% - 32774 Trial stop

% Palabras de cada sesión de evaluación
words = ['cabras' ; 'hinojo' ; 'sabana' ; 'gacela' ; 'agosto'];

% Carga de datos
datos_EEG_1 = load(file1);
datos_EEG_2 = load(file2);
datos_EEG_3 = load(file3);
datos_EEG_4 = load(file4);
datos_EEG_5 = load(file5);

% Se guardan los datos en una estructura
datos_EEG = [datos_EEG_1 datos_EEG_2 datos_EEG_3 datos_EEG_4 datos_EEG_5];

% Se inicializa la ventana de muestras que vamos a coger (1000 ms)
window = 1000;
windowb = 200;

% Carga del filtro
fir = load(fir_filter);

for i = 1:size(words, 1)

        temp_CAR(i).samples = pre_car1(datos_EEG(i).samples);
        temp_FIR(i).filtered = filtfilt(fir.Num, 1, temp_CAR(i).samples);
        datos_EEG(i).samples = temp_FIR(i).filtered;

        % Se busca dentro de los estímulos guardados, aquellos índices que sean un TrialStart (32773) y un TrialStop
        (32774). De esta forma, eliminamos los estímulos que nos sobran entre trials y tenemos acotadas las letras
        index(i).start = find(datos_EEG(i).stims(:, 2) == 32773);
        index(i).stop = find(datos_EEG(i).stims(:, 2) == 32774);

        % Se inicializan las variables
        data_Signal(i).code = [];
        data_Signal(i).trials = [];
        data_Signal(i).signal = {};
end

% Si por cada segundo (1000 ms = window), voy a coger 256 muestras (fs), el número de muestras de la ventana será la
ventana por fs entre la ventana
w_samples = round(window * (datos_EEG(1).samplingFreq / 1000));
w_baseline = round(windowb * (datos_EEG(1).samplingFreq / 1000));

% Se busca cada trial de cada palabra
for i = 1:size(words, 1)
```

```
        for trial = 1:length(index(i).start)

                % Se busca dentro de los estímulos aquellos que estén delimitados por TrialStart y TrialStop. Están
                buscados utilizando los índices. Los stims son los valores teóricos
                temp(i).tmp = datos_EEG(i).stims(index(i).start(trial):index(i).stop(trial), :);

                % Una vez se han filtrado los comienzos de los estímulos, buscamos los estímulos que interesan sabiendo
                que solo son válidos aquellos que sean 33025 (row 0) o mayores y 33036 (column 6) o menores
                temp(i).code = temp(i).tmp(temp(i).tmp(:, 2) < 33037 & temp(i).tmp(:, 2) > 33024, 2)- 33025;
                temp(i).times = temp(i).tmp(temp(i).tmp(:, 2) < 33037 & temp(i).tmp(:, 2) > 33024, 1);

                % Se guarda en la variable code los valores que toman filas y columnas en la iluminación
                data_Signal(i).code = [data_Signal(i).code; temp(i).code];

                % Se inicializa la variable trials utilizando una matriz de unos del tamaño de la variable code
                data_Signal(i).trials = [data_Signal(i).trials; trial.*ones(length(temp(i).code),1)];

                % Se busca cada iluminación
                for flash = 1:length(temp(i).code)

                        % Sabiendo cual es el tiempo en el que se produce cada iluminación, ya que lo hemos guardado en
                        temp_times, restamos a cada elemento del vector de tiempos general este tiempo y nos quedamos con
                        la menor distancia. De esta forma, obtenemos el índice de la muestra en el que se está produciendo
                        la iluminación
                        [c, indexAux] = min(abs(datos_EEG(i).sampleTime - temp(i).times(flash)));

                        % Se guardan en la variable temp_samples, todos los tiempos de las muestras a partir del índice de
                        la iluminación, hasta ocupar la ventana. De esta forma, se están haciendo cachos de 1 segundo de
                        las muestras que tienen iluminación
                        temp(i).baseline = datos_EEG(i).samples((indexAux - w_baseline):(indexAux - 1), :);

                        % Media y desviación típica
                        mb = mean(temp(i).baseline, 1);
                        sb = std(temp(i).baseline);

                        temp(i).samples = datos_EEG(i).samples(indexAux : (indexAux + w_samples - 1), :);
                        temp(i).samples = (temp(i).samples - repmat(mb, w_samples, 1))./repmat(sb, w_samples, 1);

                        % Se guarda en celdas (cell) en la variable signal, los tiempos de toda la señal cuando se produce
                        una iluminación
                        data_Signal(i).signal = [data_Signal(i).signal; {temp(i).samples}];
                end
        end
end
```

```
% Se inicializan las variables matrix_basal y matrix_P300
for i = 1:8

        canal(i).matrix_basal = [];
        canal(i).matrix_P300 = [];
end



for w = 1:size(words, 1)

        % Llamada a la función que devuelve las filas y columnas en las que se encuentra la palabra que se ha escrito
        [word_code(w).code] = Obtain_word_code_RCP_HSRD(words(w,:));

        % Se inicializa la variable labels con una matriz de ceros del tamaño de la variable code
        labels = zeros(length(data_Signal(w).code), 1);

        labels(data_Signal(w).code == word_code(w).code.row(data_Signal(w).trials)' | data_Signal(w).code ==
        word_code(w).code.col(data_Signal(w).trials)') = 1;

        % Se buscan las partes de la señal en las que el label sea 1 para representar la curva del P300
        P300 = data_Signal(w).signal(labels == 1);

        % Se buscan las partes de la señal en las que el label sea 0 para representar la curva basal
        basal = data_Signal(w).signal(labels == 0);

        % Filtro de ranura a 50 Hz
        wo = 50 / (datos_EEG(w).samplingFreq / 2);
        bw = wo / 100;
        [b,a] = iirnotch(wo, bw);

        % Se obtiene el número de canales y el número de iluminaciones
        num_channels = size(P300{1}, 2);
        num_flash = size(P300, 1);
        num_nonflash = size(basal, 1);

        for i = 1:num_channels
                for j = 1:num_flash

                        samples_filt1{j, 1}(:, i) = filtfilt(b, a, P300{j}(:, i));

                        % Se convierte la matriz de celdas en una matriz normal
                        matrix_P300(:, j) = samples_filt1{j, 1}(:, i);
                end

                for k = 1:num_nonflash

                        samples_filt2{k, 1}(:, i) = filtfilt(b, a, basal{k}(:, i));
```

```
                        % Se convierte la matriz de celdas en una matriz normal
                        matrix_basal(:, k) = samples_filt2{k, 1}(:, i);
                end

                % Guardo en cada matriz los datos pertenecientes a cada palabra en cada canal
                canal(i).matrix_basal = [canal(i).matrix_basal matrix_basal];
                canal(i).matrix_P300 = [canal(i).matrix_P300 matrix_P300];
        end
end

for i = 1:8

        % Se realiza el promedio de las épocas tanto para la curva del P300 como para la curva basal para los 8 canales
        promedio_P300(:, i) = nanmean(canal(i).matrix_P300, 2);
        promedio_basal(:, i) = nanmean(canal(i).matrix_basal, 2);
end


% Media de PO7 + PO8
P300_PO = [promedio_P300(:, 6) promedio_P300(:, 7)];
promedio_P300_PO = mean(P300_PO, 2);

basal_PO = [promedio_basal(:, 6) promedio_basal(:, 7)];
promedio_basal_PO = mean(basal_PO, 2);

% Vector de tiempos
t = linspace(0, 1000, datos_EEG(1).samplingFreq);

% Visualización
figure;

hold on;
plot(t, promedio_P300(:, 2), '-k', 'linewidth', 2);
xlabel('Tiempo (ms)');
ylabel('Amplitud (\muV)');
plot(t, promedio_basal(:, 2), ':k');
plot(t, promedio_P300_PO, '-m', 'linewidth', 2);
plot(t, promedio_basal_PO, ':m');
hold off;
grid on;

legend('Cz target', 'Cz non target', 'PO7+PO8 target', 'PO7+PO8 non target');

% Se borran las variables que no son necesarias
clearvars -except words word_code data_Signal w_samples samplingFreq datos_EEG canal mb sb basal P300 promedio_P300
promedio_basal
```

```
	end

/*************************************************************************************************************/
/** Método Obtain_plot_p300_CBP_HSRDCBP _5words(file1, file2, file3, file4, file5, fir_filter)           **/
/** Se encarga de obtener el P300 de los archivos CBP y HSRDCBP                                          **/
/*************************************************************************************************************/
	function P300_data_CBP_HSRDCBP = Obtain_plot_p300_CBP_HSRDCBP_5words(file1, file2, file3, file4, file5, table1, table2, table3,
	table4, table5, fir_filter)

		% Tabla blanca o negra
		% - 33025 Row 0 White
		% - 33030 Row 5 White
		% - 33031 Row 0 Black
		% - 33036 Row 5 Black
		% - 33037 Column 0 White
		% - 33039 Column 2 White
		% - 33040 Column 0 Black
		% - 33042 Column 2 Black

		% Identificadores
		% - 32773 Trial start
		% - 32774 Trial stop

		% Palabras de cada sesión de evaluación
		words = ['cabras' ; 'hinojo' ; 'sabana' ; 'gacela' ; 'agosto'];

		table = [table1 ; table2 ; table3 ; table4 ; table5];

		% Carga de datos
		datos_EEG_1 = load(file1);
		datos_EEG_2 = load(file2);
		datos_EEG_3 = load(file3);
		datos_EEG_4 = load(file4);
		datos_EEG_5 = load(file5);

		% Se guardan los datos en una estructura
		datos_EEG = [datos_EEG_1 datos_EEG_2 datos_EEG_3 datos_EEG_4 datos_EEG_5];

		% Se inicializa la ventana de muestras que vamos a coger (1000 ms)
		window = 1000;
		windowb = 200;

		% Carga del filtro
		fir = load(fir_filter);

		for i = 1:size(words, 1)

			temp_CAR(i).samples = pre_car1(datos_EEG(i).samples);
```

```
        temp_FIR(i).filtered = filtfilt(fir.Num, 1, temp_CAR(i).samples);
        datos_EEG(i).samples = temp_FIR(i).filtered;

        % Se busca dentro de los estímulos guardados, aquellos índices que sean un TrialStart (32773) y un TrialStop
        (32774). De esta forma, eliminamos los estímulos que nos sobran entre trials y tenemos acotadas las letras
        index(i).start = find(datos_EEG(i).stims(:, 2) == 32773);
        index(i).stop = find(datos_EEG(i).stims(:, 2) == 32774);

        % Se inicializan las variables
        data_Signal(i).code = [];
        data_Signal(i).trials = [];
        data_Signal(i).signal = {};
end

% Si por cada segundo (1000 ms = window), voy a coger 256 muestras (fs), el número de muestras de la ventana será la
ventana por fs entre la ventana
w_samples = round(window * (datos_EEG(1).samplingFreq / 1000));
w_baseline = round(windowb * (datos_EEG(1).samplingFreq / 1000));

% Se busca cada trial de cada palabra
for i = 1:size(words, 1)
        for trial = 1:length(index(i).start)

        % Se busca dentro de los estímulos aquellos que estén delimitados por TrialStart y TrialStop. Están buscados
        utilizando los índices. Los stims son los valores teóricos
        temp(i).tmp = datos_EEG(i).stims(index(i).start(trial):index(i).stop(trial), :);

        % Una vez se han filtrado los comienzos de los estímulos, buscamos los estímulos que interesan sabiendo que solo
        son válidos aquellos que sean 33025 (row 0) o mayores y 33042 (column 6) o menores
        temp(i).code = temp(i).tmp(temp(i).tmp(:, 2) < 33043 & temp(i).tmp(:, 2) > 33024, 2) - 33025;
        temp(i).times = temp(i).tmp(temp(i).tmp(:, 2) < 33043 & temp(i).tmp(:, 2) > 33024, 1);

        % Se guarda en la variable code los valores que toman filas y columnas en la iluminación
        data_Signal(i).code = [data_Signal(i).code; temp(i).code];

        % Se inicializa la variable trials utilizando una matriz de unos del tamaño de la variable code
        data_Signal(i).trials = [data_Signal(i).trials; trial.*ones(length(temp(i).code),1)];

        % Se busca cada iluminación
        for flash = 1:length(temp(i).code)

                % Sabiendo cual es el tiempo en el que se produce cada iluminación, ya que lo hemos guardado en
                temp_times, restamos a cada elemento del vector de tiempos general este tiempo y nos quedamos con
                la menor distancia. De esta forma, obtenemos el índice de la muestra en el que se está produciendo
                la iluminación
                [c, indexAux] = min(abs(datos_EEG(i).sampleTime - temp(i).times(flash)));
```

```
                        % Se guardan en la variable temp_samples, todos los tiempos de las muestras a partir del índice de
                        la iluminación, hasta ocupar la ventana. De esta forma, se están haciendo cachos de 1 segundo de
                        las muestras que tienen iluminación
                        temp(i).baseline = datos_EEG(i).samples((indexAux - w_baseline):(indexAux - 1), :);

                        % Media y desviación típica
                        mb = mean(temp(i).baseline, 1);
                        sb = std(temp(i).baseline);

                        temp(i).samples = datos_EEG(i).samples(indexAux : (indexAux + w_samples - 1), :);
                        temp(i).samples = (temp(i).samples - repmat(mb, w_samples, 1))./repmat(sb, w_samples, 1);

                        % Se guarda en celdas (cell) en la variable signal, los tiempos de toda la señal cuando se produce
                        una iluminación
                        data_Signal(i).signal = [data_Signal(i).signal; {temp(i).samples}];
                end
        end
end

% Se inicializan las variables matrix_basal y matrix_P300
for i = 1:8

        canal(i).matrix_basal = [];
        canal(i).matrix_P300 = [];
end

for w = 1:size(words, 1)

        % Llamada a la función que devuelve las filas y columnas en las que se encuentra la palabra que se ha escrito
        [word_code(w).code] = Obtain_word_code_CBP_HSRDCBP(words(w, :), table(w, :));

        % Se inicializa la variable labels con una matriz de ceros del tamaño de la variable code
        labels = zeros(length(data_Signal(w).code), 1);

        labels(data_Signal(w).code == word_code(w).code.row(data_Signal(w).trials)' | data_Signal(w).code ==
        word_code(w).code.col(data_Signal(w).trials)') = 1;

        % Se buscan las partes de la señal en las que el label sea 1 para representar la curva del P300
        P300 = data_Signal(w).signal(labels == 1);

        % Se buscan las partes de la señal en las que el label sea 0 para representar la curva basal
        basal = data_Signal(w).signal(labels == 0);

        % Filtro de ranura a 50 Hz
        wo = 50 / (datos_EEG(w).samplingFreq / 2);
        bw = wo / 100;
        [b,a] = iirnotch(wo, bw);
```

```
        % Se obtiene el número de canales y el número de iluminaciones
        num_channels = size(P300{1}, 2);
        num_flash = size(P300, 1);
        num_nonflash = size(basal, 1);

        for i = 1:num_channels
                for j = 1:num_flash

                        samples_filt1{j, 1}(:, i) = filtfilt(b, a, P300{j}(:, i));

                        % Se convierte la matriz de celdas en una matriz normal
                        matrix_P300(:, j) = samples_filt1{j, 1}(:, i);
                end

                for k = 1:num_nonflash

                        samples_filt2{k, 1}(:, i) = filtfilt(b, a, basal{k}(:, i));

                        % Se convierte la matriz de celdas en una matriz normal
                        matrix_basal(:, k) = samples_filt2{k, 1}(:, i);
                end

                % Guardo en cada matriz los datos pertenecientes a cada palabra en cada canal
                canal(i).matrix_basal = [canal(i).matrix_basal matrix_basal];
                canal(i).matrix_P300 = [canal(i).matrix_P300 matrix_P300];
        end
end

for i = 1:8

        % Se realiza el promedio de las épocas tanto para la curva del P300 como para la curva basal para los 8 canales
        promedio_P300(:, i) = nanmean(canal(i).matrix_P300, 2);
        promedio_basal(:, i) = nanmean(canal(i).matrix_basal, 2);
end

% Media de PO7 + PO8
P300_PO = [promedio_P300(:, 6) promedio_P300(:, 7)];
promedio_P300_PO = mean(P300_PO, 2);

basal_PO = [promedio_basal(:, 6) promedio_basal(:, 7)];
promedio_basal_PO = mean(basal_PO, 2);

% Vector de tiempos
t = linspace(0, 1000, datos_EEG(1).samplingFreq);

% Visualización
figure;
```

```
            hold on;
            plot(t, promedio_P300(:, 2), '-k', 'linewidth', 2);
            xlabel('Tiempo (ms)');
            ylabel('Amplitud (\muV)');
            plot(t, promedio_basal(:, 2), ':k');
            plot(t, promedio_P300_PO, '-m', 'linewidth', 2);
            plot(t, promedio_basal_PO, ':m');
            hold off;
            grid on;

            legend('Cz target', 'Cz non target', 'PO7+PO8 target', 'PO7+PO8 non target');

            % Se borran las variables que no son necesarias
            clearvars -except words word_code data_Signal w_samples samplingFreq datos_EEG canal mb sb basal P300 promedio_P300
            promedio_basal

      end

/*************************************************************************************************/
/** Método Obtain_word_RCP_HSRD(code)                                                           **/
/** Se encarga de obtener la palabra a partir del código de los archivos RCP y HSRD             **/
/*************************************************************************************************/
      function [word] = Obtain_word_RCP_HSRD(code)

            % Se inicializa el contador
            j = 1;
            word = [];

            row = code(:, 1);
            col = code(:, 2);

            % Se inicializa la matriz
            matrix = ['abcdef'; 'ghijkl'; 'mnopqr'; 'stuvwx'; 'yz1234'; '567890'];

            for i = 1:length(row)

                  word = [word , matrix(row(i) + 1, col(i) - 5)];

                  if i >= (j * 6)
                        j = (j + 1);
                  end
            end

      end
```

```
/*************************************************************************************************************/
/** Método Obtain_word_CBP_HSRDCBP(code, table_path)                                                        **/
/** Se encarga de obtener la palabra a partir del código de los archivos CBP y HSRDCBP                      **/
/*************************************************************************************************************/
	function [word] = Obtain_word_CBP_HSRDCBP(code, table_path)

		% Se inicializa el contador
		word = [];

		for i = 1:5

			% Se abre el archivo que contiene la matriz
			fid = fopen(table_path(i, :));

			% Se asigna a cada variable una línea del archivo
			tline1 = fgetl(fid);
			tline2 = fgetl(fid);

			% Se cierra el archivo que contiene la matriz
			fclose(fid);

			w = i - 1;

			% Se asigna a cada variable el contenido entre ; que haya en cada línea del archivo. El strsplit devuelve un cell
			whiteTable = upper(strsplit(tline1, ';'));
			blackTable = upper(strsplit(tline2, ';'));

			for j = 1:6

				newWhiteTable(j, :) = whiteTable{j};
				newBlackTable(j, :) = blackTable{j};
			end

			row = code(:, 1);
			col = code(:, 2);

			for j = (w*6 + 1):(w*6 + 6)

				if row(j) >=6 && col(j) >= 15 && col(j) <= 17

					word_aux = newBlackTable(row(j) - 5, col(j) - 14);

				elseif row(j) <= 5 && col(j) <= 14 && col(j) >= 12

					word_aux = newWhiteTable(row(j) + 1, col(j) - 11);
```

```
                        else
                                r = round(5 * rand());
                                c = round(2 * rand());
                                m = round(rand());

                                if m == 0
                                        word_aux = newWhiteTable(r + 1, c + 1);
                                else
                                        word_aux = newBlackTable(r + 1, c + 1);
                                end
                        end

                        word = strcat(word, lower(word_aux));
                end
        end
end


/*************************************************************************************************/
/** Método Obtain_word_code_RCP_HSRD(code, table_path)                                          **/
/** Se encarga de obtener el código a partir de la palabra de los archivos RCP y HSRD          **/
/*************************************************************************************************/
        function [code_word] = Obtain_word_code_RCP_HSRD(word)


                % Se inicializa el contador
                j = 1;

                % Se inicializa la matriz
                matrix = ['abcdef';'ghijkl';'mnopqr';'stuvwx';'yz1234';'567890'];

                % Se asigna a la variable el contenido de la matriz
                matrix = upper(matrix);

                % Se asigna a la variable, la palabra que se quiera buscar en la matriz
                word = upper(word);

                % Se buscan todas las letras de la palabra
                for  i = 1:length(word)

                        % Se busca la fila desde 1 hasta el tamaño de la matriz para recorrerla entera
                        for row = 1:length(matrix)

                                % Se busca la columna de la matriz  en la que se encuentra la letra de la palabra. Para ello, se busca en
                                la matriz, sabiendo la fila en la que estamos
                                col = find((matrix(row, :) == word(i)), 1);
```

```
                        % Cuando encuentra la letra (col tiene un valor), entra en el if
                        if (col)

                        % Se guardan en dos vectores, las filas y columnas que las letras de la palabra buscada
                        code_word.row(j) = row - 1;
                        code_word.col(j) = col + 5;

                        % Se suma uno al contador para pasar a la siguiente fila o columna
                        j = j + 1;
                        end
                end
        end
end

/*************************************************************************************************************/
/** Método Obtain_word_code_CBP_HSRDCBP(code, table_path)                                                   **/
/** Se encarga de obtener el código a partir de la palabra de los archivos CBP y HSRDCBP                    **/
/*************************************************************************************************************/
        function [word_code_cbp] = Obtain_word_code_CBP_HSRDCBP(word, file_matrix)

                % Se inicializa el contador
                j = 1;

                % Se abre el archivo que contiene la matriz
                fid = fopen(file_matrix);

                % Se asigna a cada variable una línea del archivo
                tline1 = fgetl(fid);
                tline2 = fgetl(fid);

                % Se cierra el archivo que contiene la matriz
                fclose(fid);

                % Se asigna a cada variable el contenido entre ; que haya en cada línea del archivo. El strsplit devuelve un cell
                whiteTable = upper(strsplit(tline1, ';'));
                blackTable = upper(strsplit(tline2, ';'));

                % Se asigna a la variable, la palabra que se quiera buscar en la matriz
                word = upper(word);

                % Se buscan todas las letras de la palabra
                for  i = 1:length(word)

                        % Se busca la fila desde 1 hasta el tamaño de la matriz blanca para recorrerla entera
                        for row = 1:length(whiteTable)
```

```
                        % Se busca la columna de la matriz blanca en la que se encuentra la de la palabra. Para ello, como la
                        matriz blanca es un cell, accedemos a él utilizando corchetes
                        col = find((whiteTable{row} == word(i)), 1);

                        % Cuando encuentra la letra (col tiene un valor), entra en el if
                        if (col)

                        % Se guardan en dos vectores, las filas y columnas que contengan las letras de la palabra buscada
                        word_code_cbp.row(j) = row - 1;
                        word_code_cbp.col(j) = col + 11;

                        % Se suma uno al contador para pasar a la siguiente fila o columna
                        j = j + 1;
                end
        end

        % Se busca la fila desde 1 hasta el tamaño de la matriz negra para recorrerla entera
        for row = 1:length(blackTable)

                % Se busca la columna de la matriz blanca en la que se encuentra la letra de la palabra. Para ello, como la
                matriz negra es un cell, accedemos a él utilizando corchetes
                col = find((blackTable{row} == word(i)), 1);

                % Cuando encuentra la letra (col tiene un valor), entra en el if
                if (col)

                        % Se guardan en dos vectores, las filas y columnas que contengan las letras de la palabra buscada
                        word_code_cbp.row(j) = row + 5;
                        word_code_cbp.col(j) = col + 14;

                        % Se suma uno al contador para pasar a la siguiente fila o columna
                        j = j + 1;
                end
        end
end

/*****************************************************************************************************/
/** Método Obtain_precision_RCP_HSRD(train_path, test_path, maxnsec)                                **/
/** Se encarga de obtener la precisión de los archivos RCP y HSRD                                   **/
/*****************************************************************************************************/
function [precision] = Obtain_precision_RCP_HSRD(train_path, test_path, maxnsec)

        if nargin < 3, maxnsec = 10;
        end

        training = load(train_path);
```

```
testing.feature.feat =[];
testing.feature.code =[];
testing.feature.labels =[];
testing.feature.trial =[];

word_teorica = ['cabras' , 'hinojo' , 'sabana' , 'gacela' , 'agosto'];
precision = [];

for word = 1:length(test_path)

        temp = load(test_path{word});

        testing.feature.feat = [testing.feature.feat; temp.feature.feat];
        testing.feature.code = [testing.feature.code; temp.feature.code];
        testing.feature.labels = [testing.feature.labels; temp.feature.labels];
        testing.feature.trial = [testing.feature.trial; (word-1) * 6 + temp.feature.trial];
end

for n = 1:maxnsec

        clearvars -except training testing word_teorica maxnsec n precision;

        newtraining = set_seq(training,n);
        newtesting = set_seq(testing,n);

        rLDA = fitcdiscr(newtraining.feature.feat, newtraining.feature.labels);

        [yp, x0, sys_pred] = predict(rLDA, newtesting.feature.feat);
        scores = x0(:, 2);

        l_trial = (12 * n);
        ntrials = length(scores) / l_trial;
        predicted_trials = NaN(ntrials, 2);

        for trial = 1:ntrials

        scores_trial = scores(((trial - 1) * l_trial + 1) : (trial * l_trial));
        codes_trial = newtesting.feature.code(((trial-1) * l_trial + 1) : (trial * l_trial));

        row_scores = NaN(6, 1);
        col_scores = NaN(6, 1);

        for row = 0:1:5

                row_scores(row + 1) = sum(scores_trial(codes_trial == row));
        end

        [~, temp] = max(row_scores);
```

```
                predicted_trials(trial, 1) = temp - 1;

                for col = 6:1:11

                                col_scores(col - 5)= sum(scores_trial(codes_trial == col));
                end

                [~, temp] = max(col_scores);
                predicted_trials(trial, 2) = temp + 5;
                end

                word = Obtain_word_RCP_HSRD(predicted_trials);

                bien = sum(word == word_teorica);

                precision(n) = 100 * (bien / length(word));

                fprintf('- nsec: %i \t %.4f%%\n', n, precision(n));
        end
end

function newdata = set_seq(data, nsec)

        ntrials = length(unique(data.feature.trial));
        l_trial = length(data.feature.labels) / ntrials;

        newdata.feature.code = [];
        newdata.feature.trial = [];
        newdata.feature.labels = [];
        newdata.feature.feat = [];

        for t = 1:ntrials
                temp_code = data.feature.code(((t - 1) * l_trial + 1) : (t * l_trial));
                newdata.feature.code = [newdata.feature.code; temp_code(1 : (nsec * 12))];

                temp_trial = data.feature.trial(((t - 1) * l_trial + 1) : (t * l_trial));
                newdata.feature.trial = [newdata.feature.trial; temp_trial(1 : (nsec * 12))];

                temp_labels = data.feature.labels(((t - 1) * l_trial + 1) : (t * l_trial));
                newdata.feature.labels = [newdata.feature.labels; temp_labels(1 : (nsec * 12))];

                temp_feat = data.feature.feat(((t - 1) * l_trial + 1) : (t * l_trial), :);
                newdata.feature.feat = [newdata.feature.feat; temp_feat(1 : (nsec * 12), :)];
        end
end
```

```
/*************************************************************************************************/
/** Método Obtain_precision_CBP_HSRDCBP(train_path, test_path, maxnsec)                         **/
/** Se encarga de obtener la precisión de los archivos CBP y HSRDCBP                            **/
/*************************************************************************************************/
       function [precision] = Obtain_precision_CBP_HSRDCBP(train_path, table_train_path, test_path, table_test_path, maxnsec)

              if nargin < 5, maxnsec = 10;
              end

              training = load(train_path);

              testing.feature.feat =[];
              testing.feature.code =[];
              testing.feature.labels =[];
              testing.feature.trial =[];

              word_teorica = ['cabras' , 'hinojo' , 'sabana' , 'gacela' , 'agosto'];
              precision = [];

              for word = 1:length(test_path)

                     temp = load(test_path{word});

                     testing.feature.feat = [testing.feature.feat; temp.feature.feat];
                     testing.feature.code = [testing.feature.code; temp.feature.code];
                     testing.feature.labels = [testing.feature.labels; temp.feature.labels];
                     testing.feature.trial = [testing.feature.trial; (word - 1) * 6 + temp.feature.trial];
              end

              for n = 1:maxnsec

                     clearvars -except training testing word_teorica maxnsec n precision table_test_path precision;

                     newtraining = set_seq(training, n);
                     newtesting = set_seq(testing, n);

                     rLDA = fitcdiscr(newtraining.feature.feat, newtraining.feature.labels);

                     [yp, x0, sys_pred] = predict(rLDA, newtesting.feature.feat);
                     scores = x0(:, 2);

                     l_trial = (18 * n);
                     ntrials = length(scores) / l_trial;
                     predicted_trials = NaN(ntrials, 2);

                     for trial = 1:ntrials

                            scores_trial = scores(((trial - 1) * l_trial + 1) : (trial * l_trial));
```

```
                codes_trial = newtesting.feature.code(((trial-1) * l_trial + 1) : (trial * l_trial));

                row_scores = NaN(12, 1);
                col_scores = NaN(6, 1);

                for row = 0:1:11

                        row_scores(row + 1) = sum(scores_trial(codes_trial == row));
                end

                [~, temp] = max(row_scores);
                predicted_trials(trial, 1) = temp - 1;

                for col = 12:1:17

                        col_scores(col - 11)= sum(scores_trial(codes_trial == col));
                end

                [~, temp] = max(col_scores);
                predicted_trials(trial, 2) = temp + 11;
        end

        word = Obtain_word_CBP_HSRDCBP(predicted_trials, table_test_path);

        bien = sum(word == word_teorica);

        precision(n) = 100 * (bien / length(word));

        fprintf('- nsec: %i \t %.4f%%\n', n, precision(n));
    end
end

function newdata = set_seq(data, nsec)

    ntrials = length(unique(data.feature.trial));
    l_trial = length(data.feature.labels) / ntrials;

    newdata.feature.code = [];
    newdata.feature.trial = [];
    newdata.feature.labels = [];
    newdata.feature.feat = [];

    for t = 1:ntrials

        temp_code = data.feature.code(((t - 1) * l_trial + 1) : (t * l_trial));
        newdata.feature.code = [newdata.feature.code; temp_code(1 : (nsec * 18))];

        temp_trial = data.feature.trial(((t - 1) * l_trial + 1) : (t * l_trial));
```

```
                newdata.feature.trial = [newdata.feature.trial; temp_trial(1 : (nsec * 18))];

                temp_labels = data.feature.labels(((t - 1) * l_trial + 1) : (t * l_trial));
                newdata.feature.labels = [newdata.feature.labels; temp_labels(1 : (nsec * 18))];

                temp_feat = data.feature.feat(((t - 1) * l_trial + 1) : (t * l_trial), :);
                newdata.feature.feat = [newdata.feature.feat; temp_feat(1 : (nsec * 18), :)];
        end
end
```

PLIEGO DE CONDICIONES

En este anexo, se recogen las condiciones legales que guiarán a la realización del Trabajo Fin de Grado titulado como "Escribiendo mediante las ondas cerebrales: desarrollo y evaluación de paradigmas utilizando OpenViBE".

Para realizar este apartado, se supondrá que el trabajo ha sido encargado por una empresa cliente a una empresa consultora cuya finalidad es la utilización del sistema BCI. La empresa consultora ha debido desarrollar una línea de investigación con el objeto de llevar a cabo el trabajo. Esta línea de investigación junto con el posterior desarrollo de la aplicación está amparada por las condiciones particulares del siguiente pliego. Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes condiciones:

Condición 1

La modalidad de contratación será el concurso. La adjudicación se hará por tanto a la proposición más favorable, sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho de declararlo desierto.

Condición 2.

El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

Condición 3.

En la oferta se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

Condición 4.

La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos que sea preciso para el desarrollo de la misma.

Condición 5.

Aparte del director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no está obligado a aceptarla.

Condición 6.

El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

Condición 7.

Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le haya comunicado por escrito el Ingeniero Director de obras siempre que dicha obra se haya ejecutado a los preceptos de los pliegos de condiciones, de acuerdo a los cuales se harán modificaciones y la valoración de las diversas unidades, sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto no podrán servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

Condición 8.

Tanto en las certificaciones de la obra como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

Condición 9.

Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata, pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero considere justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

Condición 10.

Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere, y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento se sujetarán siempre al establecido en el punto anterior.

Condición 11.

Cuando el contratista, con la autorización del Ingeniero Director de obras, emplee material de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio, ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sino a lo que le correspondería si se hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

Condición 12.

Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ella se formen, o en su defecto, por lo que resulte de su medición final.

Condición 13.

El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras, así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

Condición 14.

Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

Condición 15.

La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

Condición 16.

La forma de pago será por certificaciones mensuales de la obra ejecutada de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

Condición 17.

La fecha de comienzo de las obras será a partir de los quince días naturales del replanteo oficial de las mismas, y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna a la reclamación de la fianza.

Condición 18.

Si el contratista, al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

Condición 19.

El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras o con el delegado que este designe, para todo lo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

Condición 20.

Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista la conservación de la obra ejecutada hasta la recepción de la misma, por lo que el deterioro total o parcial de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o construido por su cuenta.

Condición 21.

El contratista deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa por retraso en la ejecución siempre que este no sea debido a causas de fuerza

mayor. A la terminación de la obra se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

Condición 22.

Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas establecidas.

Condición 23.

Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución por contrata", y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

La empresa constructora, que ha desarrollado este proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares.

Condición particular 1.

La propiedad intelectual de los procesos descritos y analizados en el presente trabajo pertenece por entero a la empresa constructora representada por el Ingeniero Director del Proyecto.

Condición particular 2.

La empresa constructora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos posteriores, para la misma empresa cliente o para otra.

Condición particular 3.

Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con la autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

Condición particular 4.

En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones, así como su cantidad.

Condición particular 5.

En todas las reproducciones se indicará su procedencia, explicando el nombre del proyecto, nombre del Ingeniero Director y empresa consultora.

Condición particular 6.

Si el proyecto pasa a la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de este, la empresa consultora decidirá aceptar o no la modificación propuesta.

Condición particular 7.

Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta al añadirla.

Condición particular 8.

Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

Condición particular 9.

Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

Condición particular 10

La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

Condición particular 11.

La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia de este hecho. En este caso deberá autorizar expresamente los proyectos presentados por otros.

Condición particular 12.

El Ingeniero Director del proyecto será responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona asignada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.

Los requisitos, clasificados como documentación, hardware y software, que han sido necesarios para la elaboración del presente trabajo fin de grado son los siguientes.

DOCUMENTACIÓN

- Servicio de la Biblioteca de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid.

HARDWARE

- Equipo Intel® Core™ 2 Duo – CPU 8550U, con 2.0GHz, 64 bits y 8GB de RAM.
- Equipo BCI:
  - Amplificador g.USBamp de 16 canales.
  - Gorro EEG de 64 canales.
  - Electrodos EEG de oro.
  - Electrodos de referencia.
  - Gel conductor.

SOFTWARE

- Microsoft® Windows 10 Home.

  Copyright© 2015, Microsoft Corporation.
- Microsoft® Word (16.0.10325.20118).

  Microsoft® Office 365 ProPlus -es-es.

  Copyright© 2016, Microsoft Corporation.
- MATLAB® Student R2017a (9.2).

  Copyright© 2017, The MathWorks Inc.
- Adobe Acrobat Reader DC (Versión 18.011.20055).

  Copyright© 1984–2012, Adobe Systems Incorporated.
- Microsoft® Visual Studio Ultimate 2013 (Versión 12.0.21005.1).

  Copyright© 2013, Microsoft Corporation.
- CMake (2.8.12.1) con Qt (4.6.2).

  Creative Commons (CC), Kitware Inc.
- OpenViBE (Versión 2.1.0).
- g.USBamp Simulink High-Speed Online Processing.

Anexo D

PRESUPUESTO

El presupuesto estimado de acuerdo a las condiciones expuestas en el anexo de pliego de condiciones es el siguiente.

EJECUCIÓN MATERIAL

Equipo Intel® Core™ i7-8550U Quad Core, 1.8 GHz a 4.0 GHz, 64 bits y 8GB de RAM	900€
Equipo BCI	13.300€
Sistema Operativo: Microsoft® Windows 10 Home	120€
Software: Microsoft® Office 365 ProPlus -es-es	70€
Software: MATLAB® Student R2017a	500€
TOTAL	14.890€

GASTOS GENERALES

16% sobe la ejecución material	2.382,4€

BENEFICIO INDUSTRIAL

6% sobe la ejecución material	893,4€

MATERIAL FUNGIBLE

Gastos de Impresión (tinta y papel)	200€
Encuadernación	120€
DVDs	2€
TOTAL	322€

HONORARIOS DEL PROYECTO

450 horas a 30€/hora	13.500€

SUBTOTAL DEL PRESUPUESTO

Subtotal del presupuesto	31.987,8€

I.V.A. APLICABLE

21% sobre el subtotal de presupuesto	6.717,44€

TOTAL DEL PRESUPUESTO

Total del presupuesto	**38.705,24€**

El total del presupuesto, en euros, asciende a:

TREINTA Y OCHO MIL SETECIENTOS CINCO EUROS CON VEINTICUATRO CÉNTIMOS DE EURO.

En Valladolid, Septiembre de 2018.

Fdo. Cristina Cantalapiedra Cabezas