Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo de Fin de Grado

Prototipo de un sistema para la evaluación de interfaces cerebro - computador basado en un motor de videojuegos

Prototype for assesing brain - computer interfaces based on video game engine

Víctor Tejera Santos

Septiembre de 2022

Tutores:
José Ignacio Estévez Damas
Sergio Elías Hernández Alonso

D. **José Ignacio Estévez Damas**, con N.I.F. 43.786.097-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como co-tutor

D. **Sergio Elías Hernández Alonso**, con N.I.F. 43.616.882-G profesor Titular de Universidad adscrito al Departamento de Ingeniería Industrial de la Universidad de La Laguna, como co-tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Prototipo de un sistema para la evaluación de interfaces cerebro - computador basado en un motor de videojuegos*"

ha sido realizada bajo su dirección por D. **Víctor Tejera Santos** con N.I.F. 45.897.544 D

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de septiembre de 2022

Agradecimientos

En primer lugar, quiero agradecer a mis tutores de TFG por permitirme embarcarme en este proyecto. En especial quiero agradecer a José Ignacio por guiarme durante toda esta travesía, por su paciencia y por no rendirse conmigo cuando la situación era desesperante.

Agradezco por supuesto a todos los profesores que me han dado clase durante el grado, por inculcarme los conocimientos necesarios para llevar a cabo este proyecto y los futuros.

Quiero también dar las gracias a mis familiares y amigos por su apoyo incondicional.

Finalmente quiero mostrar mi gratitud con todos aquellos que no he mencionado pero que han hecho posible que me encuentre donde me encuentro hoy.

Licencia

Resumen

El objeto de este proyecto es desarrollar un prototipo para la experimentación con interfaces cerebro-computador (BCI) basados en encefalografía (EEG). En concreto, se pretende utilizar para poner a prueba sistemas que puedan comunicar acciones para la navegación de vehículos robotizados. Para ello se han utilizado las herramientas diseñadas por OpenBCI para la captura del EEG junto con un motor de videojuegos: Panda3D. Este último es usado para crear una aplicación en Python que simule un entorno en 3D visitado en primera persona por el usuario. Mediante el uso del protocolo de comunicación "LSL" los datos obtenidos en OpenBCI se han podido utilizar en la aplicación Python. La aplicación cuenta con dos partes principales: un mapa 3D por el que se mueve el usuario y tres estímulos que controlan distintas acciones dentro del juego. Las acciones se realizan cuando el usuario presta atención a alguno de los tres estímulos y el sistema detecta el potencial evocado visual de estado estacionario mediante el análisis de correlación canónica. Se ha optado por la robustez, con la inclusión de un estímulo para confirmar la acción antes de realizarla. Esto reduce el número de errores en los recorridos pero aumenta el tiempo empleado para resolverlos y reduce el número de maniobras disponibles.

Palabras clave: interfaz cerebro - computador, videojuegos, EEG, potenciales evocados visuales de estado estacionario, análisis de correlación canónica

Abstract

The goal of this project is to build a prototype for the experimentation with Brain-Computer Interface (BCI) based on electroencephalography (EEG). More specifically, the intention is to use this prototype to test systems that communicate actions for the navigation for robotic vehicles. For doing so we have used tools designed by OpenBCI for acquisition of EEG data and a video game engine: Panda3D.This engine is used to build an application that simulates a 3D space visited by the user in first person. Using the "LSL" communication protocol we are able to receive and use the data obtained in OpenBci in the Python application. The user interface has two main parts: a 3D map where the user moves and three stimuli that control the movement. When the user focuses on a stimulus, the system detects the steady-state visual evoked potential (SSEVP) through the canonical correlation analysis. We opted for robustness when developing the prototype with the inclusion of a stimulus only to confirmate stored actions. This reduces the errors but increases the time taken to complete a trajectory. This decision also reduces the number of movements available.

Keywords: Brain-Computer Interface, video games, EEG, Steady state visually evoked potential, canonical correlation analysis

Índice general

Índice de Figuras

Índice de Tablas

Índice de Códigos

Capítulo 1

Introducción

En este proyecto se ha desarrollado un interfaz cerebro-computador mediante el cual somos capaces de prestar atención a un estímulo visual y con ello realizar una acción dentro de un pequeño videojuego. En este documento se va a describir todo el proceso de desarrollo de esta aplicación. Los apartados que componen la memoria son:

- Antecedentes
- Técnicas y herramientas utilizadas
- Desarrollo de la aplicación
- Evaluación del interfaz cerebro-computador
- Conclusiones y líneas futuras
- Presupuesto

El objeto de este capítulo es establecer los objetivos perseguidos y resumir los antecedentes.

1.1. Objetivos

Los objetivos del presente proyecto son:

- Muchas de los interfaces de usuario que se utilizan en la actualidad no están adaptados a personas que tienen algún tipo de impedimento, por lo que se les puede hacer difícil o directamente imposible utilizar muchos de los sistemas a los que estamos acostumbrados. Por ello, el primero de los objetivos es desarrollar un sistema para estudiar el potencial del registro de las señales cerebrales o electroencefalograma (EEG) como elemento básico de un interfaz cerebro - computador (interfaz cerebro computador conocido también por sus siglas en inglés “BCI”).
- Uno de las mayores dificultades que tienen las personas con impedimentos fsícos es la de desplazarse. Por ello, este proyecto se intentará enfocar a la construcción de un interfaz cerebro - computador (BCI) capaz de producir comandos pertinentes para un móvil destinado a realizar trayectorias en un espacio 3D con obstáculos. En este caso el móvil se limita a un avatar dentro de un videojuego, pero el objetivo a futuro es poder implementar este tipo de interfaz en un componente real, como es el caso de una silla de ruedas robotizada.
- El uso principal de este prototipo es poder evaluar un interfaz cerebro-computador (BCI) orientado a una utilización simple, robusta y segura. La búsqueda de estas caracterśticas en la interfaz desarrollada viene impulsado por lo mencionado en el punto anterior: el objetivo final es poder usar las BCI en una situación real, por lo que no puede permitirse que peligre la seguridad de la persona que vaya en el

dispositivo que utilice una aplicación de este tipo.

Se ha elegido un videojuego como sistema de pruebas debido a que el objetivo de esta línea de investigación es poder utilizar una BCI en una silla de ruedas. Un videojuego es perfecto como simulación de una situación real puesto que emula el movimiento de la silla y las interacciones con los obstáculos

1.2. Antecedentes

1.2.1. Interfaces cerebro - computador

Una interfaz cerebro-computador o Brain-Computer Interface (BCI) es esencialmente un método de comunicación directa entre la actividad eléctrica del cerebro y un dispositivo externo, el cual puede ser un ordenador o un actuador.

Una BCI consta de tres partes principales:

- Un dispositivo capaz de medir la actividad cerebral.
- Un software que se encargue de analizar y procesar la información recogida.
- Una aplicación o dispositivo externo que controlar.

Cabe resaltar también la importancia del feedback, es decir, el sistema debe hacer que el usuario conozca la decisión que el software fue capaz de interpretar con la información recabada, haciendo que de esta forma el usuario sea capaz de adaptarse al sistema BCI.

Existen dos tipos principales de BCIs. Por un lado tenemos los BCIs invasivos, que requieren cirugía para la implantación de sensores directamente en la superficie del cerebro. Por otra parte encontramos las BCIs no invasivas, en las que se opta por utilizar sensores por encima del cuero cabelludo. Idealmente se utilizarán técnicas no invasivas para el desarrollo de las BCIs, sin embargo las BCIs invasivas se siguen estudiando debido a que las señales obtenidas son más fuertes que en el caso de las no invasivas (debido a que la información no necesita atravesar el pelo, piel y hueso para ser procesada) y por tanto podrían ayudar a desarrollar otras metodologías no invasivas más eficientes.

Dentro de las BCIs invasivas podemos destacar la electrocorticografía (ECoG) en la que se utilizan electrodos directamente en la superficie expuesta del cerebro para obtener los datos pertinentes.

En rela ción a las metodologías no invasivas la mayoría de BCIs se basan en la electroencefalografía (EEG). En estas BCIs las señales se obtienen como potenciales eléctricos a través de unos electródos colocados en la superficie del cuero cabelludo. El defecto de este tipo de sistemas es que se obtiene una señal más débil, no solo debido a que esta tiene que atravesar el hueso y la piel, sino que además por la naturaleza del método se introducen perturbaciones externas (como dispositivos electrónicos) y ruido generado por el propio sujetos (movimiento de los ojos o actividad muscular). Para mejorar la calidad de la señal se suele utilizar un gel o solución salina aplicado a las puntas de los electrodos.

El interés del estudio de las metodologías no invasivas radica sobre todo en el su uso para la asistencia motriz de personas con discapacidad, como podría ser el caso de alguien en silla de ruedas o directamente en el desarrollo de exoesqueletos. No obstante, el campo de la medicina no es el único que puede aprovecharse enormemente de las BCIs, la integración en videojuegos también bebe en gran medida de este tipo de técnicas. Más abajo pasaremos a enumerar las distintas técnicas en las cuales se basan estos dos campos para construir sus BCIs.

Existen dos grandes grupos de técnicas que es importante resaltar: potenciales re-

lacionados con evento (ERPs), los cuales son potenciales eléctricos generados por el cerebro en respuesta a determinados eventos (cognitivos, sensoriales, motrices. . .) y EEG oscilatorio, que se basan en el monitoreo continuado del dominio de la frecuencia de la seãl (ondas alfa, beta, gamma, delta y theta) y tienen como ventaja que no necesitan un estímulo externo preestablecido para llevar a cabo el comando adecuado.

Dentro de los ERPs destacamos las siguientes técnicas:

- **Onda P300**: Este tipo de onda representa un pico positivo en el EEG con unas características determinadas (retraso y amplitud). Este tipo de pico en la respuesta es mucho mayor en estímulos que ocurren con poca frecuencia que en aquellos más comunes. Este tipo de técnica no es muy extendido en el campo de los videojuegos pero sí encuentra representación en el ámbito del movimiento de sillas de ruedas.
- **Potenciales visuales evocados de estado estacionario (SSVEP)**: en esta técnica al usuario se le presenta un estímulo parpadeante en la pantalla del ordenador. En cuanto el usuario centre su atención en dicho estímulo podrá observarse una respuesta a la misma frecuencia en el EEG. La detección de esta respuesta se lleva a cabo buscando la frecuencia objetivo (y sus armónicos). La respuesta es mejor en la zona occipital de la cabeza. Uno de los grandes inconvenientes de este método es que es especialmente sensible a las condiciones de iluminaciń del entorno y además la respuesta frente a determinadas frecuencias puede variar según el sujeto. A esto hay que sumarle que las frecuencias válidas para la captura de datos son bastante limitadas, por lo que el número de órdenes posibles es limitado. Este método es bastante extendido en nuestros dos campos de interés y es el que hemos empleado para la realización de este TFG.

En los últimos meses ha cobrado mucha relevancia una técnica especialmente exitosa que la empresa NextMind comercializa en forma de un dispositivo que incorpora hasta ocho electrodos en la zona occipital, y que estimula al sujeto con patrones variables (NextMind (2022)). Esta técnica utiliza una red neuronal para determinar a partir de las señales en la zona occipital a qué patrón está dirigiendo su atención el sujeto. El sistema incorpora una librería que permite desarrollar en el motor de videojuegos Unity aplicaciones controlados con este BCI.

En el grupo del EEG oscilatorio encontramos dos técnicas principales:

- **BCI para la interacción afectiva y el neurofeedback**: Se basa en medir el estado psicológico del usuario al realizar una acción. Poniendo como ejemplo el caso de los videojuegos se deberá reconocer el estado de ánimo o humor del jugador y con ello cumplir los objetivos del juego
- **Imaginería motora (MI)**: Es una caso especial y bastante extendido de la técnica anterior. En él el objetivo es que el sujeto imagine un movimiento que pueda ser detectado mediante el análisis oscilatorio y con ello facilitar el análisis y control. Esta técnica es la más extendida dentro de este segundo grupo.

Es necesario mencionar también que existe lo que se llaman técnicas híbridas que pueden dividirse en:

- Híbrido puro donde se combinan dos paradigmas de BCI.
- Híbrido fisiológico donde se combina una de las técnicas BCI con otra medida fisiológica.
- Híbrido mixto donde se combina BCI con una entrada no fisiológica.

Las técnicas híbridas (puras) son las más extendidas en el control de silla de ruedas.

1.2.2. Técnicas basadas en otras medidas fisiológicas

Como ya adelantábamos arriba existen las llamadas medidas fisiológicas, las cuales pueden servir como complemento para los BCIs pero también como alternativa para estas. Algunas de este tipo de medidas son:

- **Electrocardiografía (ECB)**: Mide el ritmo cardiaco en un intervalo de tiempo, si este ritmo se desvía de unos valores normales se llevará a cabo una acción.
- **Seguimiento de la mirada o "Eye Tracking (ET)"**: Se utiliza el movimiento de la mirada para indicar que se debe realizar una acción. El "seguimiento de la mirada" utiliza un haz de luz infrarroja que se proyecta hacia los ojos. La luz infrarroja es reflejada por la córnea. Este reflejo es detectado por un cámara con sensibilidad a la luz infraroja que determina la posición de la pupila y así infiere la posición de la misma y la orientación de la mirada.
- **Actividad electrodérmica (EDA)**: Se basa en medir la variación de conductividad de la piel mediante la sudoración.
- **Electromiografía (EMG)**: Mide la señal de acción muscular mediante el registro del potencial eléctrico en los nervios encargados de su transmisión (de cara a un interfaz, son útiles aquellos sistemas musculares que pueden moverse voluntariamente)

1.2.3. BCI en videojuegos

El uso de las BCI en videojuegos no es algo nuevo. Desde hace ya algunos años se ha estado intentando conectar las BCIs y la realidad virtual. Existen múltiples prototipos que se encargan de manipular objetos y viajar a través del mundo virtual con el uso exclusivo de los datos obtenidos del EEG. Los principales retos de esta combinación a nivel comercial van desde la necesidad de más estudios de neurociencia hasta el desarrollo de periféricos. La realidad virtual provee al usuario de un entorno seguro, controlado y que mantiene motivado al usuario que ayuda a investigar las respuestas en los procesos neuronales involucrados.

Todas las interacciones se pueden descomponer en tareas sencillas como seleccionar un objeto y manipularlo o navegar para cambiar la vista que observa el usuario. Es más, muchos estudios han demostrado que la respuesta del usuario a BCI es mucho mejor en un entorno virtual que las conseguidas en un entorno 2D clásico. En resumen, el desarrollo de videojuegos basados en BCI ayuda a mejorar este tipo de interfaces.

Algunos de los proyectos de este tipo que se han realizado son:Lcuyer et al. (2008)

- El videojuego Midbalance (Lalor et al. (2005)), desarrollado por la University College Dublin and MediaLabEurope. En este juego se mueve un personaje animado 3D por el mundo virtual.
- Un videojuego desarrollado por Graz University of Technology and University College London done un sujeto tetrapléjico navega por una calle virtual con 15 avatares (personajes) usando una silla de rueda (Leeb et al. (2007))
- La Universidad de Tokyo ha realizado varios experimentos en los que utilizan SSEVP para simular un joystick (Touyama et al. (2008))
- Recientemente la empresa NextMind ha desarrollado un BCI que se integra con el motor de videojuegos Unity NextMind (2022). La empresa Meta ha adquirido a Next-Mind, posíblemente interesándose por esta tecnología como medio de interacción en el metaverso.

Capítulo 2

Técnicas y herramientas utilizadas

2.1. Captura del EEG

A la hora de recoger datos en tiempo real se ha optado por utilizar las herramientas desarrolladas por la empresa OpenBCI, un equipo de personas que se centran en crear herramientas de código abierto para biodetección y neurociencia (OpenBCI (2022))

2.1.1. Hardware

Para realizar todas las pruebas se ha utilizado el casco UltraCortex Mark IV, disponible para su compra en la página de OpenBCI. Aunque el armazón puede adquirirse directamente, en este trabajo de fin de grado (TFG), se ha utilizado uno impreso como objeto de un TFG anterior. En la figura 2.1 puede verse el modelo utilizado.

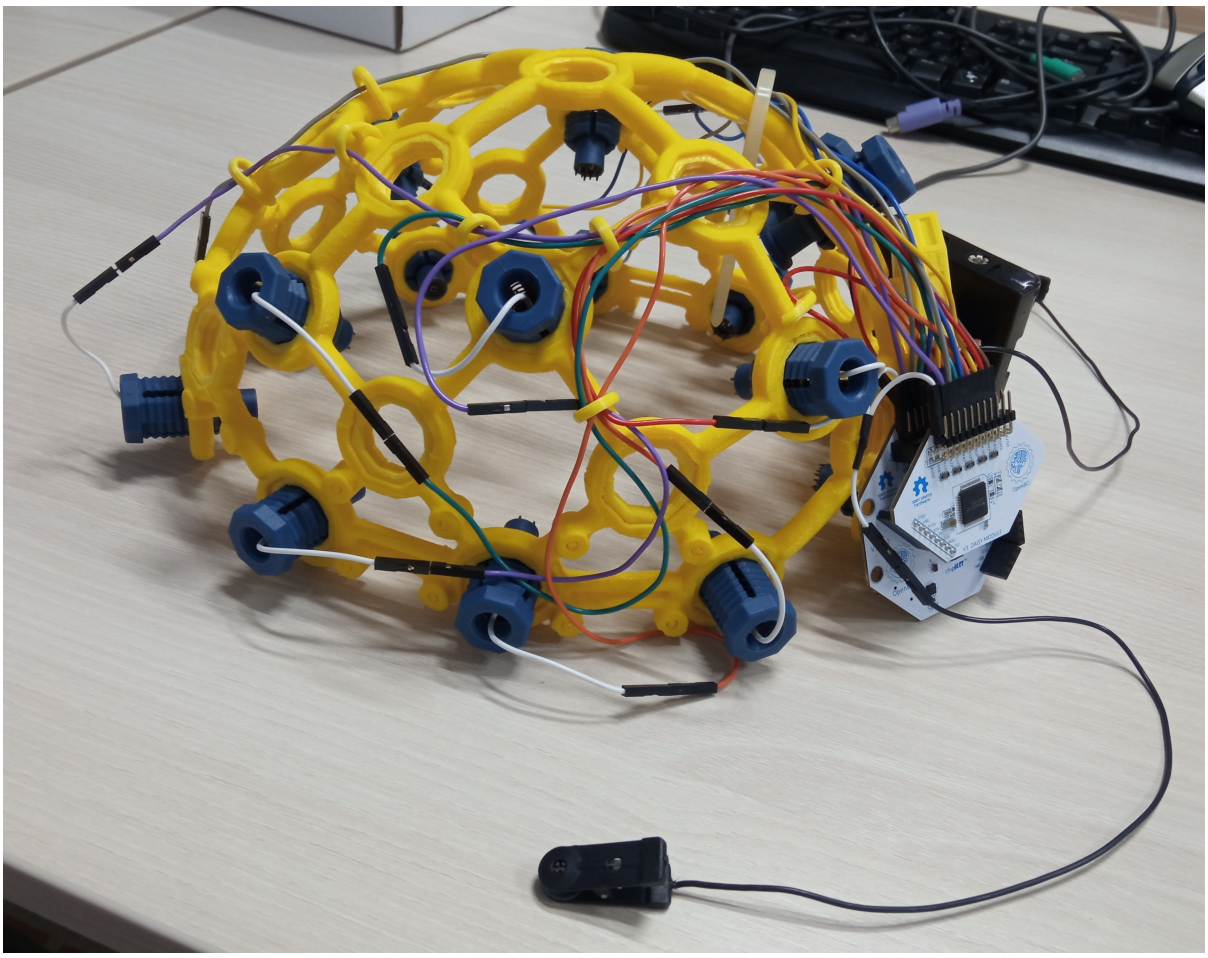

Figura 2.1: UltraCortex Mark IV preparado para ser utilizado

Además de las piezas impresas, el UltraCortex Mark IV viene con cables para conectar la placa a los electrodos, clips para las orejas y tres tipos distintos de electrodos: electrodos para zonas con cabello, electrodos para zonas sin pelo (frente) y electrodos de sujeción cuyo único objetivo es distribuir correctamente el peso del casco (OpenBCI (2021)) En la figura 2.3 se muestran los elementos mencionados por separado.

Figura 2.2: Clips para los lóbulos de las orejas

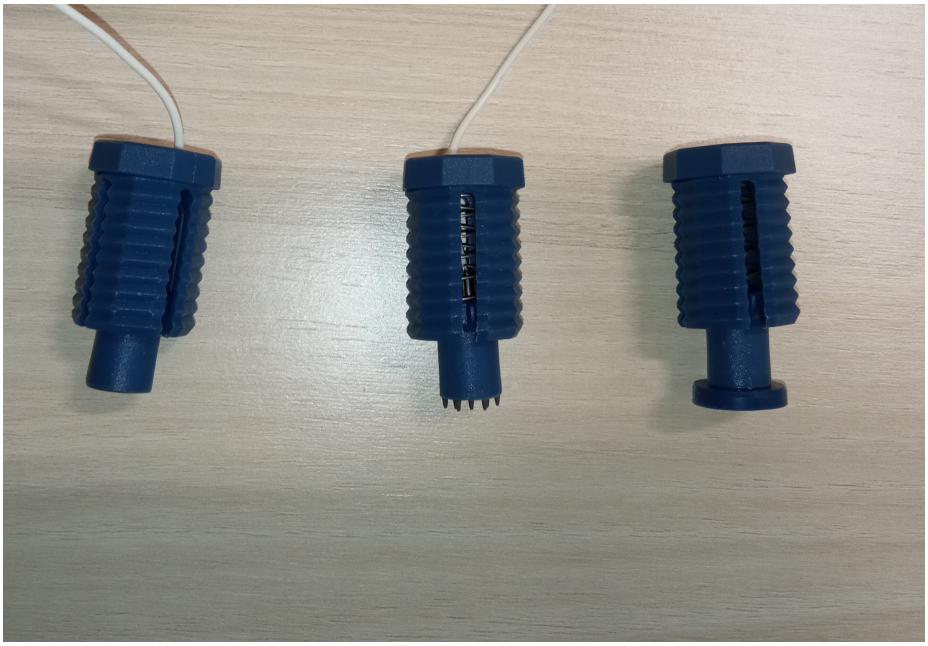

Figura 2.3: Electrodos del casco. De izquierda a derecha: electrodo plano(zonas sin pelo), puntiagudo(zonas con pelo) y de sujeción

Finalmente, tenemos que mencionar la placa de procesamiento y adquisición de datos. Dicha placa debe ser adquirida de forma separada al casco.

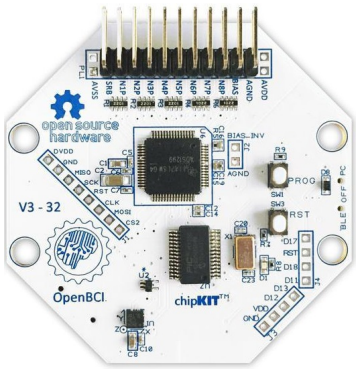

Figura 2.4: Placa Cyton

Lo que se muestra en la Figura 2.4 es una placa Cyton de 8 canales, tiene un procesador de 32 bits (microcontrolador PIC32MX250F128B) que es compatible con el sistema “Arduino”. Si solo utilizáramos esta placa estaríamos limitados a los 8 canales. Para ampliar el número de canales registrados tenemos además de la Cyton de 8 canales, un módulo Daisy que permite ampliarla hasta los 16 canales. La figura 2.5 muestra la placa Cyton con el módulo Daisy conectado.

La conexión entre este sistema y el ordenador se realiza mediante un enlace inalámbrico Bluetooth. La placa Cyton representa un extremo de la conexión, mientras que un USB dongle conectado al ordenador y que además emula un puerto serial en el PC, es el otro extremo (ver figura 2.6). Remarcar que todo el sistema debe ser alimentado con baterías AA de 1,5V (4)

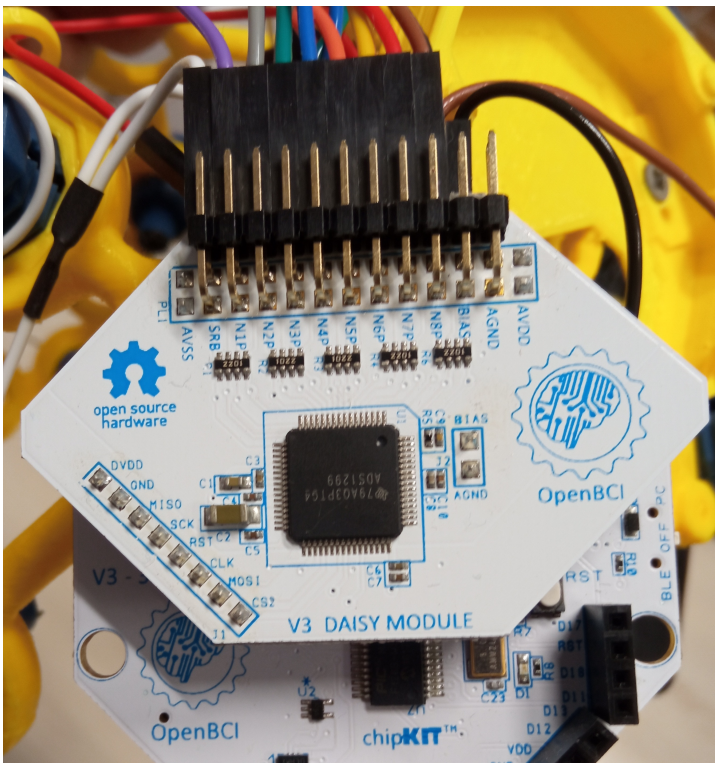

Figura 2.5: Placa Cyton + Daisy instalada en el casco

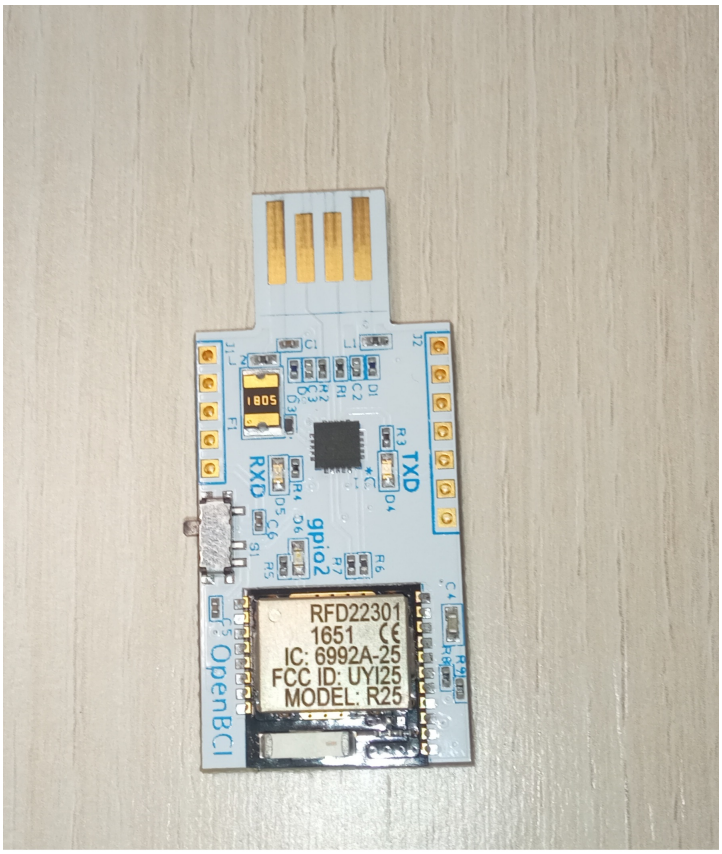

Figura 2.6: Pincho USB para conexión bluetooth entre PC y casco EEG

2.1.2. Software

Para la previsualización de los datos del EEG hemos utilizado el software desarrollado también por OpenBCI. Esencialmente nos hemos valido de:

- Su interfaz gráfica de usuario, la cual provee de información que especificaremos más abajo.
- La función de filtrado de los datos que permite el acondicionamiento de los mismos para la aplicación deseada.
- La capacidad de constituirse en un nodo emisor de datos en una red bajo el protocolo Laboratory Streaming Layer (LSL) (Kothe et al. (2019)) De esta manera, para la recogida correcta de los datos en Python hemos utilizado una librería que permite crear un flujo de datos a partir de un nodo consumidor del mismo protocolo LSL (Kothe et al. (2019))

Software "OpenBCI GUI"

Antes de continuar, debemos mencionar que la versión del software OpenBCI GUI que se utilizó inicialmente para las pruebas era la 4.2, la cual es lo suficientemente fiable si solo queremos visualizar los datos por pantalla a través de esta aplicación. Más tarde pasamos a utilizar la versión 5.1 de OpenBCI GUI, puesto que la versión anterior no podía aparentemente enviar correctamente lo datos del EEG a la red LSL, ya que la aplicación Python recogía una señal diferente al EEG que se quería transmitir. En la figura 2.8 se muestra la ventana principal de la aplicación OpenBCI.

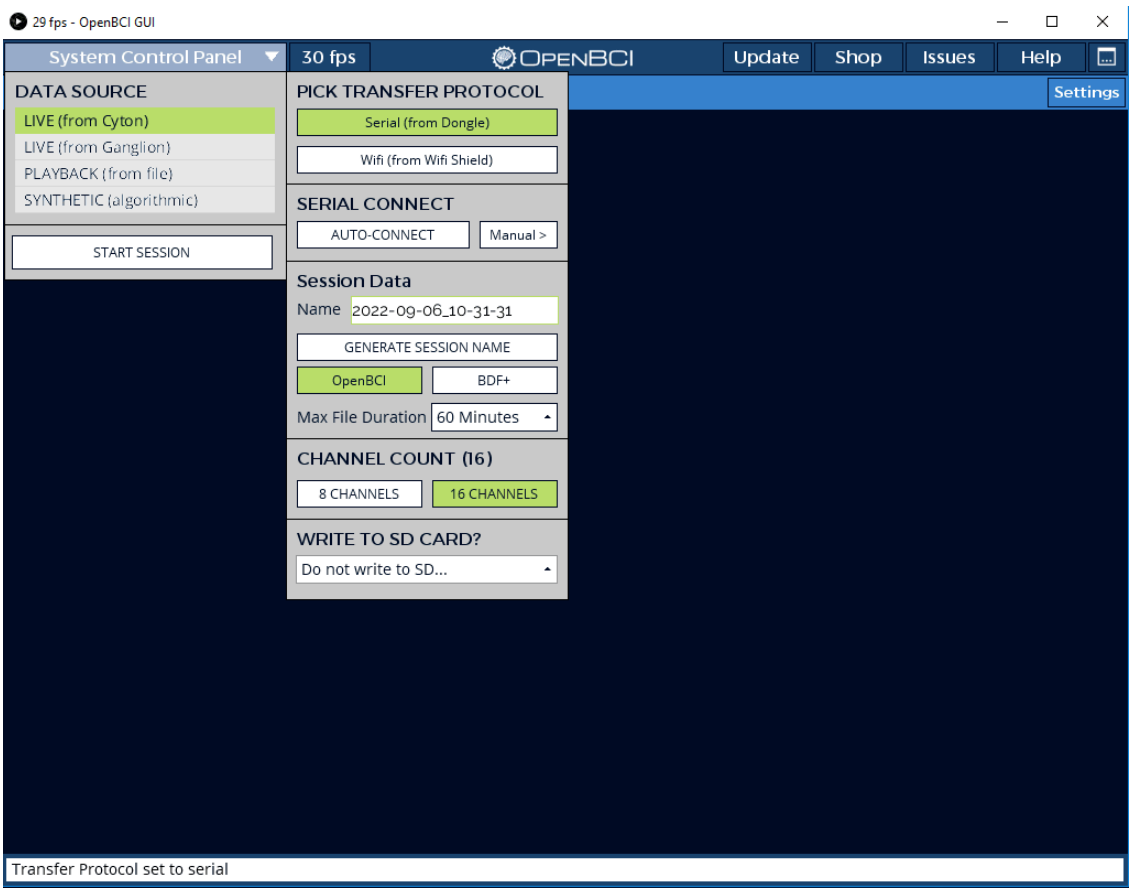

Figura 2.7: Pantalla de inicio OpenBci GUI 4.2

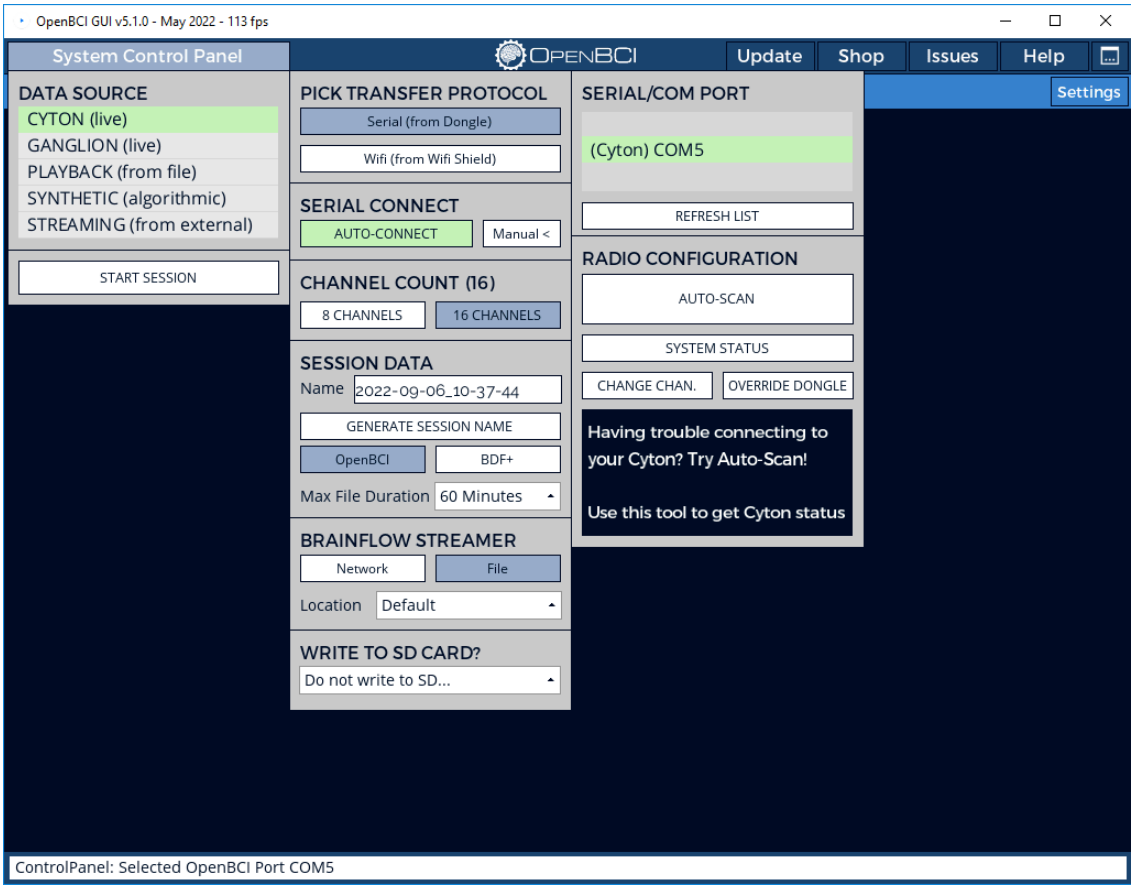

Figura 2.8: Pantalla de inicio OpenBci GUI 5.1

A la hora de utilizar esta interfaz nos servimos principalmente de tres funcionalidades: Time series, que muestra en tiempo real los potenciales del EEG; FFT, que nos da el espectro de potencias y Networking, que se utiliza para poder compartir la información de esta aplicación con otras. En particular en este trabajo se utilizó una red LSL.

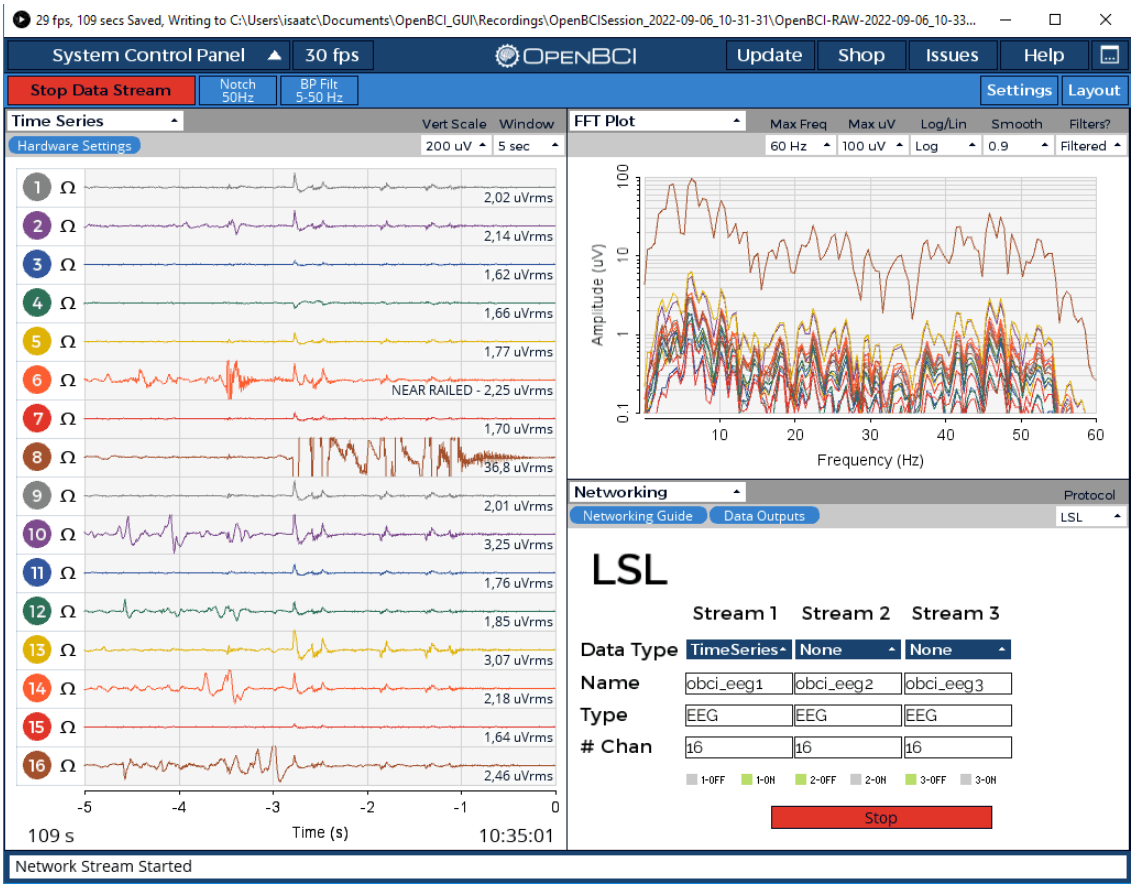

Figura 2.9: Funcionalidades utilizadas OpenBci GUI 4.2

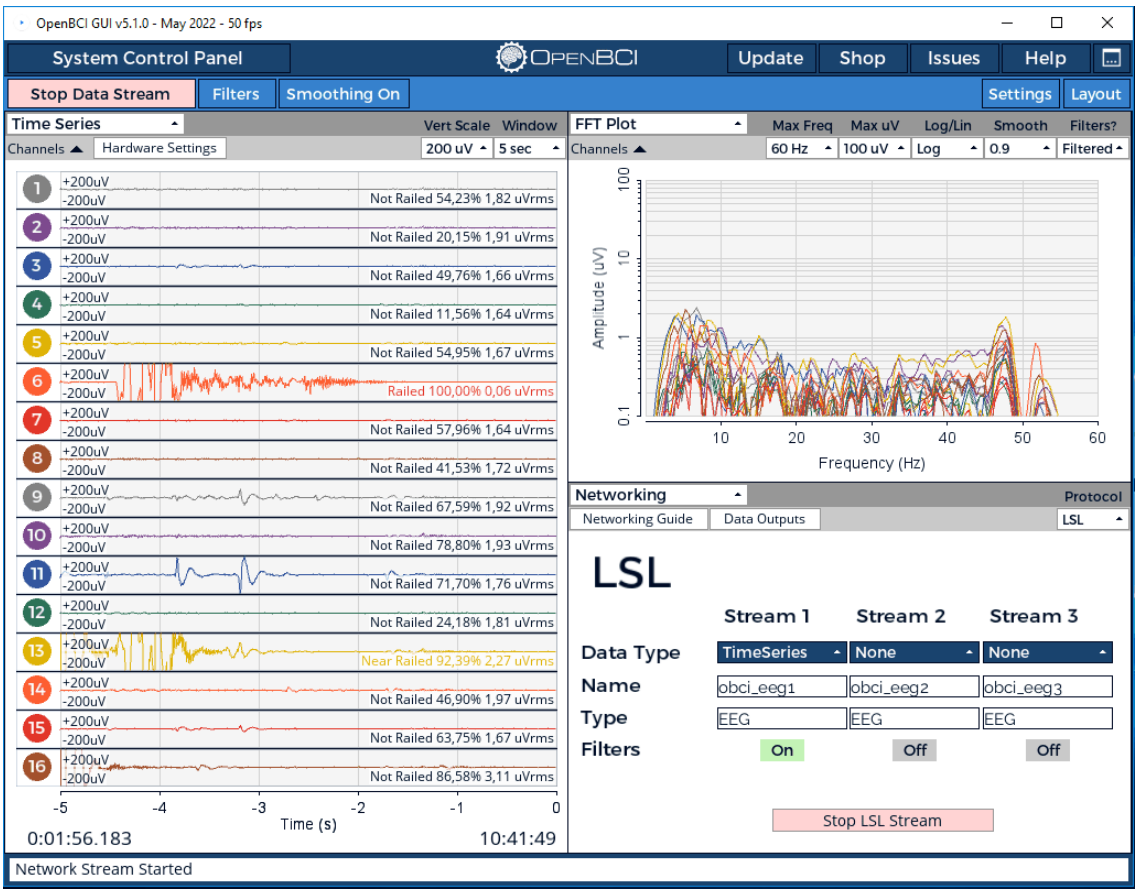

Figura 2.10: Funcionalidades utilizadas OpenBci GUI 5.1

Debemos tener en cuenta varias cosas antes de compartir estos datos con la función de networking. Lo primero es que debemos estar aplicando en todo momento los filtros que se muestran en la Figura 2.11. Lo segundo es que en el panel de configuración “Netowrking”, se deben establecer los parámetros “protocolo” y “Data Type” a LSL y Time Series respectivamente.

En el momento de enviar los datos a la red LSL hay que tener en cuenta que la opción “Filters” debe estar activada en la pestaña “Netowrking”, tal y como se aprecia en la Figura 2.10. Con esto los datos que se envíen habrán previamente pasado por dos filtros Notch en 50 y 60 Hz, así como un filtro pasabanda entre 5 Hz y 50 Hz (Figura 2.11).

El siguiente paso es reorganizar los datos de manera que puedan ser fácilmente utilizados en el videojuego. Por ello los datos requieren una etapa de procesamiento en Python. El funcionamiento de la aplicación considera dos estados del usuario:

- El usuario no está atendiendo a algún estímulo y no se requiere detectar ningún comando a partir de la señal del EEG.
- El usuario está atendiendo a un estímulo particular que induce una respuesta específica en la señal del EEG.

Por eso, los datos almacenados incluirán solo los correspondientes a las fases de atención.

La aplicación desarrollada se utiliza en dos escenarios diferentes:

- En el primero, tratamos de realizar un experimento sistemático donde de forma repetitiva el usuario atiende a una secuencia de estímulos determinada. Cada conjunto de intervalos de atención así se denomina sesión.
- En el segundo caso, la aplicación se centra en solo un periodo de atención, el último

realizado, para realizar una detección del estímulo atendido por el usuario en tiempo real.

Si estamos en el primer caso, los datos se organizan conforme a un formato particular similar al usado en (Wang et al. (2016)) En este formato, lo que se obtienen son grupos de muestras, uno por cada vez que se el usuario presta atención, en la serie sistemática de exposiciones a los estímulos. Para ello, se deben reorganizar los datos de forma que estén clasificados por el electrodo del que se toman los datos, estímulo al que estaba atendiendo el usuario (el cual es previamente conocido a la hora de realizar estas pruebas sistemáticas) y sesión. El tamaño de cada serie temporal de datos en este formato es el mismo y será igual al menor múltiplo de un tamaño de ventana establecido (las muestras que excedan ese número en una serie temporal son eliminadas).

En caso de estar en el segundo caso, con la aplicación detectando en tiempo real, la reorganización de los datos se limita a eliminar los datos que corresponden a periodos de no atención y clasificar las muestras por electrodo (el target es único y desconocido y solo hay una sesión).

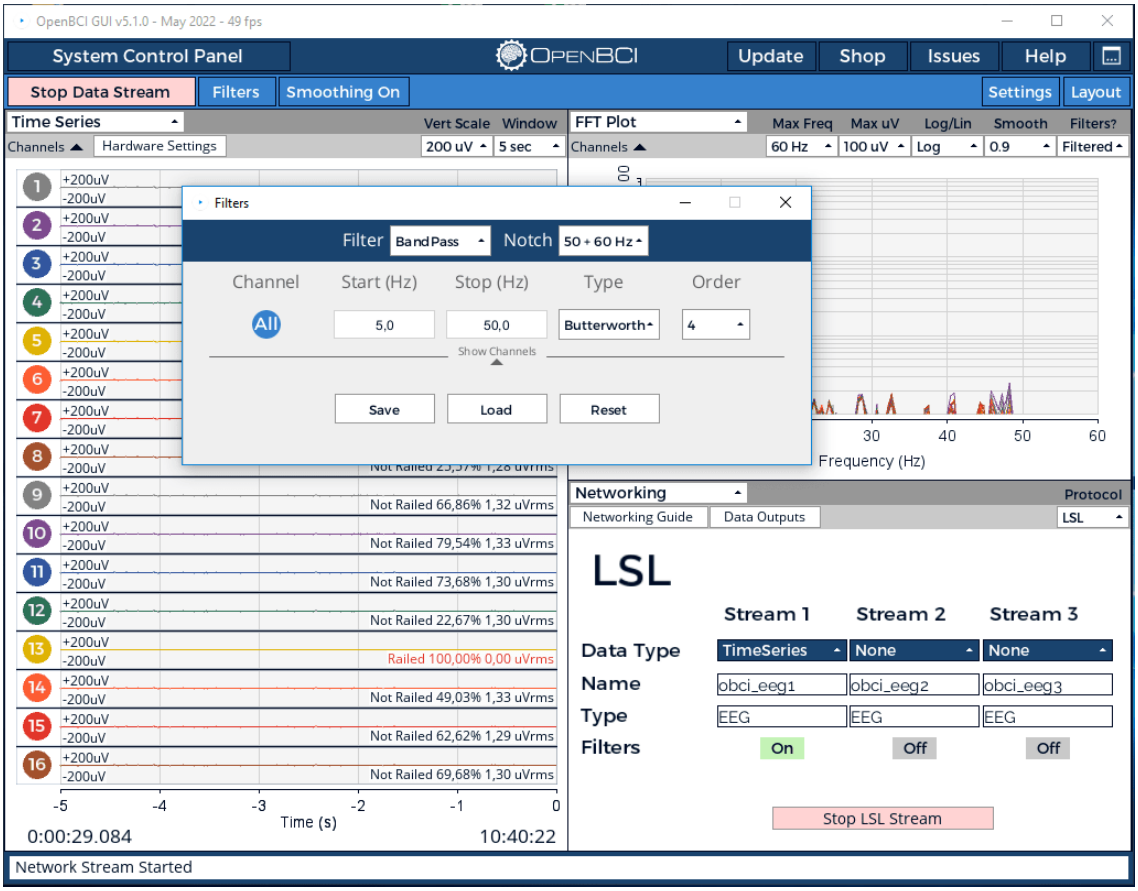

Figura 2.11: Configuración de filtros utilizada en la transmisión de datos

2.1.3. Ventajas y desventajas

Tras utilizar el hardware y software de OpenBCI podemos concluir que sus principales ventajas y desventajas son las siguientes:

El software, como indica su nombre, es de código abierto por lo que puede desarrollarse y mantiene las ventajas de este tipo de código.

A título de curiosidad, había un bug con los archivos de sonido en la versión 5.1 que

impedía lanzar correctamente el programa en algunos ordenadores, pero gracias a que el software es de código abierto pudimos encontrar una solución a ese problema.

Otra de las ventajas, esta vez centrándonos en el hardware, es la versatilidad del casco gracias a los distintos tipos de electrodos que se ponen a nuestra disposición. Asimismo, los electrodos puntiagudos para las zonas con pelo facilitan el contacto y hacen que el uso de una solución salina para mejorar la conductividad no sea necesaria, cosa que agiliza y facilita mucho la realización de las distintas pruebas.

En cuanto a las desventajas podemos resaltar la limitación en la frecuencia de muestreo: esta frecuencia siempre es de 125 Hz, lo que limita las frecuencias que podemos utilizar en los estímulos e impide realizar pruebas a alta frecuencia que serían muy interesantes en este campo.

Otro de los inconvenientes es la falta de documentación de la interfaz gráfica de usuario, sobre todo en versiones más antiguas de esta, donde las funcionalidades y etiquetas eran menos autoexplicativas. En relación al hardware, si bien mencionábamos que los distintos electrodos le añadían versatilidad, el armazón del casco se la quita puesto que no es adaptable al tamaño de la cabeza del usuario, limitando así el número de personas que pueden llevar puesto un modelo concreto.

Sin embargo, en la web de OpenBCI se pueden comprar otros dos modelos (uno más pequeño y otro más grande) o bien, obtener los archivos necesarios para imprimir el armazón en 3D.

Laboratory Streaming Layer

Laboratory Streaming Layer (LSL) es un protocolo de comunicaciones orientado a distribuir datos de series temporales entre productores y consumidores de los mismos. Los datos son etiquetados por marcas de tiempo, estableciéndose un mecanismo de sincronización automática entre el reloj utilizado para marcar los datos en el lado del consumidor y el reloj que sirve para marcar el momento de la producción del dato en el lado del productor.

LSL cuenta con una librería principal y un conjunto de herramientas asociadas a ella, las cuales son compatibles con distintos lenguajes (Python, C++, Java...)(Kothe et al. (2019))

Al principio no conocíamos la interfaz de Networking del OpenBCI GUI, por lo que utilizamos LSL directamente sobre la consola de Python e iniciábamos el stream de datos desde ahí. Esto tenía dos problemas principales. El primero de ellos era que los datos que nos llegaban al videojuego estaban sin filtrar, por lo que antes de utilizarlos teníamos que hacerlos pasar por los filtros pertinentes que se implementaron. El segundo problema era que no se podían visualizar los datos en la interfaz de usuario a la vez que los enviábamos al videojuego, por lo que errores en los datos o malfuncionamiento de alguno de los electrodos no podía ser detectado inmediatamente.

Con suerte, después de descubrir la pestaña de Networking en la interfaz gráfica no es necesario tener que volver a utilizar LSL directamente sobre Python, lo que aligeró en gran medida la carga de trabajo.

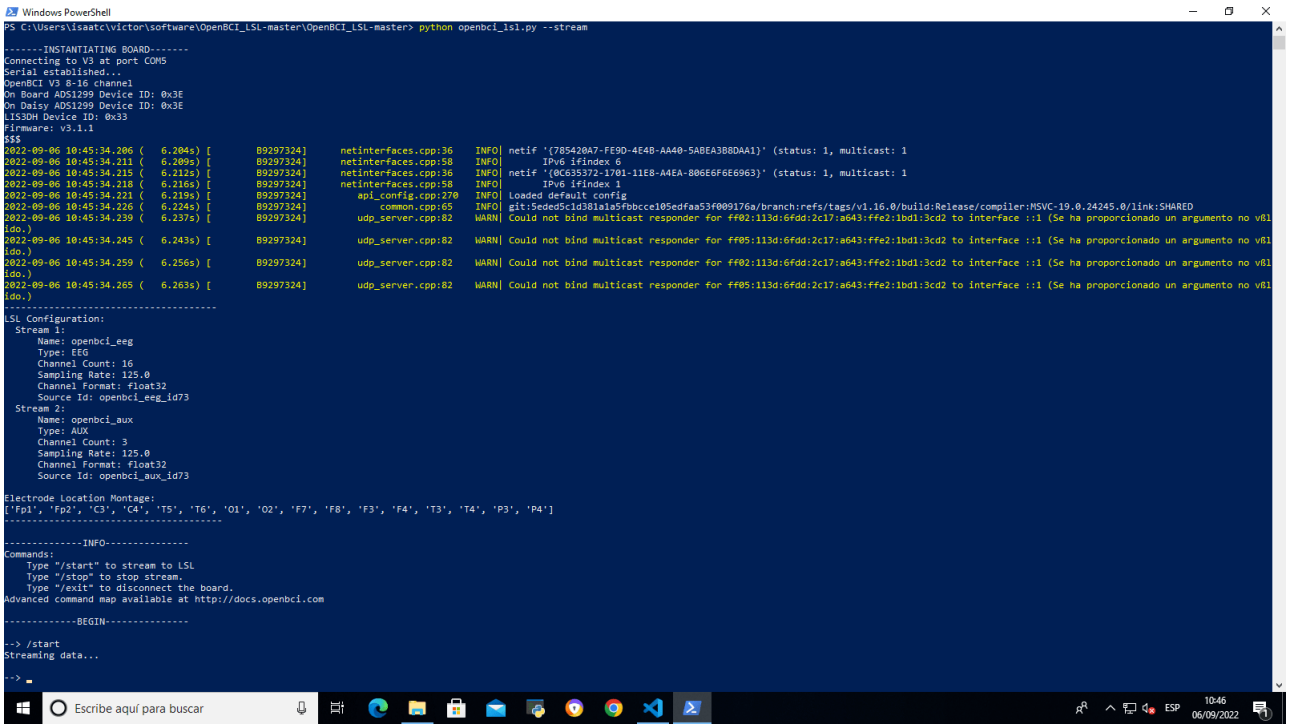

Figura 2.12: LSL funcionando a través de la ventana de comandos

2.2. Análisis de correlación canónica

Una vez procesados y tratados los datos en Python, el último paso antes de realizar una acción dentro del videojuego es analizar esos datos y averiguar si realmente el sujeto ha estado atendiendo a los estímulos que se le presentan en la pantalla. Para ello hacemos uso del método análisis de correlación canónica (CCA) (Korstanje (2021))

El CCA es un método estadístico que se utiliza para analizar la correlación existente entre dos grupos de datos. En el caso de esta aplicación con un BCI basado en potenciales evocados de estado estacionario, se trata de encontrar una relación de dependencia entre los datos obtenidos en los diferentes canales del EEG y un conjunto de señales sinusoidales de referencia, construidas con senos y cosenos a las frecuencias que se espera se induzcan respuestas neuronales por la exposición a los estímulos. Estas frecuencias son por lo general las frecuencias base de los estímulos y sus armónicos.

La idea intuitiva del método es encontrar una combinación lineal en ambos conjuntos de señales que maximice la correlación, y aprovecha la multiplicidad de señales para disminuir el efecto del ruido.

2.2.1. Explicación del método matemático

En esta sección se va a explicar brevemente las matemáticas detrás del método. Debemos empezar teniendo dos grupos de variables X e Y, tal y como se muestra en 2.1 y 2.2. Cada fila representará las muestras de las distintas variables a estudiar (Penn State's Department of Statistics (2016))

$$X = \begin{pmatrix} X_1 \\ X_2 \\ ... \\ X_n \end{pmatrix} \tag{2.1}$$

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ ... \\ Y_m \end{pmatrix} \tag{2.2}$$

Debemos seleccionar X e Y de forma que el nḿero de variables dentro del grupo Y (variables de referencia) sea mayor que el del grupo 'X'.

El siguiente paso es definir combinaciones lineales de cada uno de los dos grupos. El nḿero de combinaciones lineales de cada grupo será igual al número total de variables en X.Cada una de estas combinaciones lineales recibe el nombre de variables canónicas

$$\begin{aligned} P_1 &= a_{11}X_1 + X_2a_{12} + X_3a_{13} + ... + X_na_{1n} \\ P_2 &= a_{21}X_1 + X_2a_{22} + X_3a_{23} + ... + X_na_{2n} \\ P_3 &= a_{31}X_1 + X_2a_{32} + X_3a_{33} + ... + X_na_{3n} \\ &... \\ P_n &= a_{n1}X_1 + X_2a_{n2} + X_3a_{n3} + ... + X_na_{nn} \end{aligned} \tag{2.3}$$

$$\begin{aligned} Q_1 &= b_{11}Y_1 + Y_2b_{12} + Y_3b_{13} + ... + Y_mb_{1m} \\ Q_2 &= b_{21}Y_1 + Y_2b_{22} + Y_3b_{23} + ... + Y_mb_{2m} \\ Q_3 &= b_{31}Y_1 + Y_2b_{32} + Y_3b_{33} + ... + Y_mb_{3m} \\ &... \\ Q_n &= b_{n1}Y_1 + Y_2b_{n2} + Y_3b_{n3} + ... + Y_mb_{nm} \end{aligned} \tag{2.4}$$

Llegados a este punto quedan definidas las parejas de variables canónicas 2.5

$$(P_i, Q_i) \tag{2.5}$$

Una vez definidas estas parejas, se deben definir las varianzas individuales de cada variable canónica como se ve en las ecuaciones 2.6 y 2.7, así como la covarianza de cada pareja, como aparece en la ecuación 2.8

$$var\,(P_i) = \sum_{k=1}^{n}\sum_{l=1}^{n} a_{ik}a_{il}cov(X_k, X_l) \tag{2.6}$$

$$var\,(Q_j) = \sum_{k=1}^{n}\sum_{l=1}^{m} b_{jk}b_{jl}cov(Y_k, Y_l) \tag{2.7}$$

$$cov\,(P_i, Q_j) = \sum_{k=1}^{n}\sum_{l=1}^{m} a_{ik}b_{jl}cov(X_k, Y_l) \tag{2.8}$$

Con todo esto se puede ya definir el coeficiente de correlación como se muestra en la ecuación 2.9

$$\rho^* = \frac{cov(P_i, Q_i)}{\sqrt{var(P_i)var(Q_i)}} \tag{2.9}$$

Este valor es el que se busca maximizar y el cual se debe tener en cuenta al momento de formar cada variable canónica. Además de esta condición debemos imponer otras para poder hallar las variables canónicas que se buscan, en la ecuación 2.10 se resumen estas condiciones.

$$
\begin{aligned}
&var(P_i) = var(Q_i) = 1 \\
&cov(P_1, P_i) = cov(Q_1, Q_i) = 0 \\
&cov(P_2, P_i) = cov(Q_2, Q_i) = 0 \\
&... \\
&cov(P_{i-1}, P_i) = cov(Q_{i-1}, Q_i) = 0 \\
&cov(P_1, Q_i) = cov(P_i, Q_1) = 0 \\
&cov(P_2, Q_i) = cov(P_i, Q_2) = 0 \\
&... \\
&cov(P_{i-1}, Q_i) = cov(P_i, Q_{i-1}) = 0
\end{aligned}
\tag{2.10}
$$

La implementación utilizada en el proyecto para el cálculo de la correlación canónica es “scikit-learn” (ScikitLearn (2022)).

2.2.2. Utilización de CCA para BCI

El uso del análisis de correlación canónica no es ajeno a las interfaces cerebro-computador. En el artćulo ((Wang et al., 2016)) se describe un experimento cuyas características se asemejan a lo que se quiere conseguir dentro del videojuego. En dicho experimento se someten a distintos usuarios (entrenados y no entrenados en este tipo de experimentos) a 40 estímulos de frecuencias distintas durante 5 segundos para comprobar cómo reaccionaban. Una vez se obtenían los datos eran analizados mediante el uso de CCA con resultados muy positivos. En este otro artículo (Lin et al. (2006)), de la misma institución se prueba la eficacia del método tiempo antes de realizar el experimento que se menciona en este mismo apartado.

En el caso de este experimento de BCI, el uso del análisis de correlación canónica para el análisis de los SSVEP se vería como en la figura 2.13. La idea general del método se basa en utilizar CCA para encontrar los coeficientes de correlación entre las señales del EEG seleccionadas y una serie de conjuntos de señales de referencia,la frecuencia de las cuales será igual a la de los tres estímulos utilizados. Cada vez que se inicie un proceso de detección, el método se aplicará una vez por cada señal de referencia, por lo que acabaremos con un valor de correlación por conjunto.

Cada conjunto de referencia se forma a partir de ondas sinusoidales a la frecuencia del estímulo y sus armónicos (tres en estos experimentos). Tras obtener todos los coeficientes de la correlación se busca el mayor, y la frecuencia asociada a dicho valor se considera que es la detección.

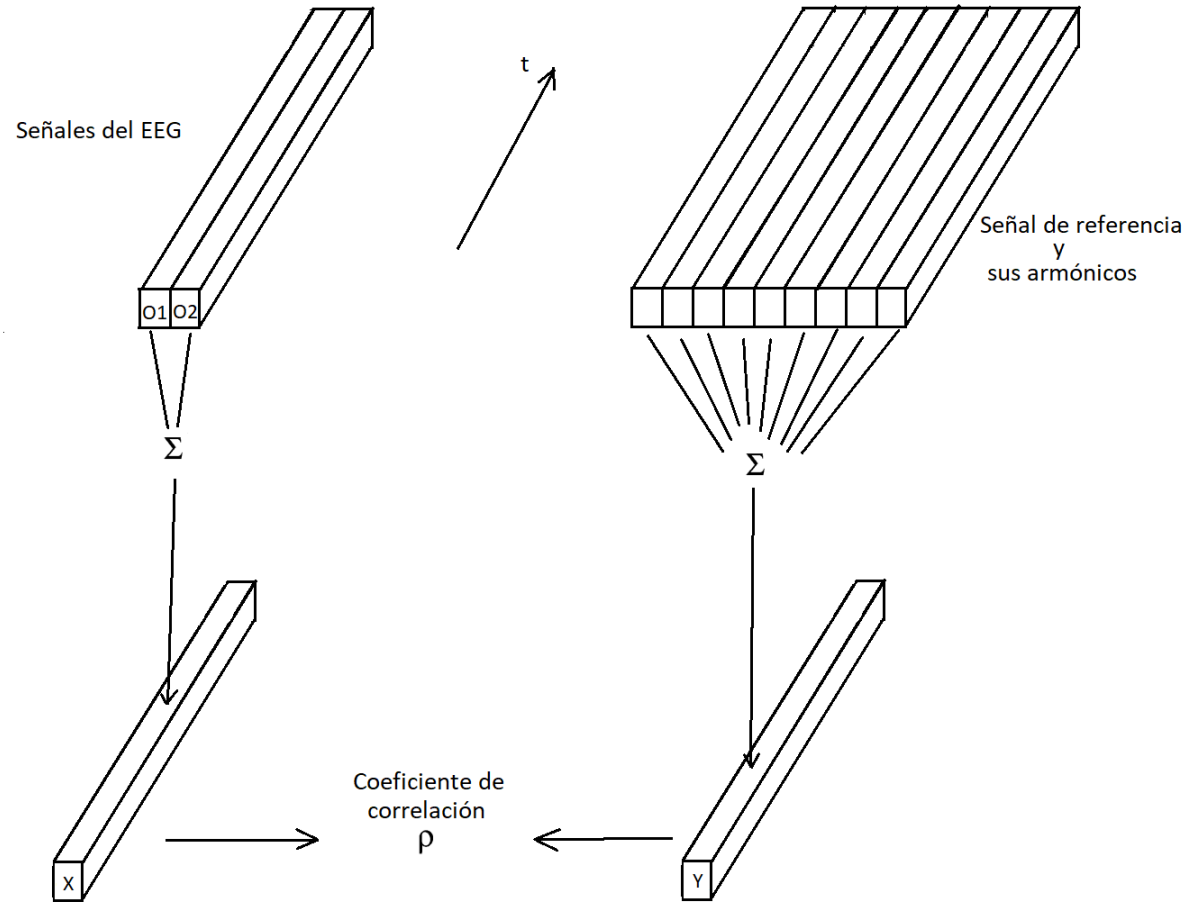

Figura 2.13: Diagrama del análisis de correlación canónica para el caso de nuestra para BCI

2.3. Motores de videojuegos

El otro aspecto importante del proyecto, además de la recogida de datos del EEG es la programación de un videojuego sencillo en que poder utilizar los datos recogidos en tiempo real por el casco de OpenBCI. El objetivo de este videojuego es crear un entorno adecuado para probar el interfaz BCI para el control de un vehículo en un espacio tridimensional.

2.3.1. Introducción

En un principio se presentaron dos opciones distintas. La primera era utilizar el motor de videojuegos de Unreal Engine, programado en el lenguaje C++. Esta elección facilitaría en gran medida la creación del videojuego sofisticado y realista, pero dificultará la inclusión de los datos tomados puesto que al momento de iniciar con el proyecto no había librerías para C++ que ayudaran a integrar los datos tomados y realizar la detección de forma sencilla. En su lugar se priorizó la obtención de un prototipo simple pero funcional sobre el que analizar los problemas derivados del propio BCI sin añadir

complejidad en el lado de la parte gráfica. En el futuro, una vez se hubiera estudiado suficientemente la problemática de este tipo de BCI, se puede optar a trasladar el sistema a un motor sofisticado.

Por otra parte, tenemos la opción de un motor de videojuegos como Panda3D, que sin la sofisticación de Unreal permite crear videojuegos 3D sencillos utilizando un lenguaje como Python. La elección de Python como lenguaje para desarrollar los prototipos se debe en parte a su simplicidad como lenguaje y sobre todo a la gran cantidad de librerías que se encuentran disponibles para acometer de análisis y obtención de datos. Además, su extendido uso garantiza el soporte y la mejora continua del lenguaje y sus librerías.

2.3.2. Panda3d sobre Python

Panda3d es un motor de videojuegos de código abierto y sin coste alguno para su uso. Consiste en una librería formada por implementaciones de clases que pueden utilizarse en los dos lenguajes que el motor soporta (C++ y Python). Desarrollar un videojuego en Panda3d consiste en escribir un programa en Python que cree clases derivadas de las proporcionadas por la libreı'a, las extienda creando nuevos métodos para mejorar la funcionalidad, y finalmente sean instanciadas.

A la hora de programar con Panda3d se debe tener en cuenta que se dispone de una clase básica que representa a un juego con su bucle principal ya implementado. De esta manera, cada vez que se lance una aplicación esta comenzará automáticamente a ejecutar el bucle principal del juego.

Esto se hará mediante el método run(). Esta funciø'n no retorna nada, por lo que nos quedaremos dentro de ese bucle hasta que el juego sea interrumpido (por ejemplo, completando un objetivo propuesto o cerrando la propia ventana del juego).

Para afectar el estado del juego hay que emplear las llamadas tareas. Hay diferentes tipos de tareas, para unas de las más utilizadas son las "tareas de ejecución", que son funciones que se ejecutan automáticamente en cada fotograma del juego. Todas las tareas que definamos se ejecutarán concurrentemente dentro del hilo principal.

Otra de las características importantes a destacar es que mediante unas sencillas líneas de código se puede cambiar el modo de funcionamiento del juego y que el motor trate de trabajar a la mayor tasa de frames posible hasta un límite que define el usuario. Esta característica es la clave de la generación de los estímulos en el juego, ya que su frecuencia va a depender de la tasa de frames. Es importante destacar que aunque se establezca el límite de los 60 fotogramas por segundo dentro del juego, si el procesador del ordenador debe encargarse de otros procesos prioritarios, el juego podría dejar de correr a una tasa estable de frames y la frecuencia de los estímulos se vería afectada.

La librería proporcionada por Panda3d no es suficiente para resolver todos los problemas de esta aplicación, por ejemplo la obtención del flujo de datos del sistema LSL. En esta versión del proyecto se está utilizando un módulo de Threading de Python para poder recoger los datos que envía el GUI de OpenBCI por LSL de forma ajena al bucle principal del videojuego, es decir, ambos procesos se desarrollan en el mismo núcleo del procesador pero de forma separada (esto es posible debido a que una frecuencia de muestreo de 125 Hz apenas carga el procesador). Si esta recogida de datos se hiciera en el mismo bucle del juego la frecuencia de muestreo se reduciría, puesto que solo se podría obtener una muestra en cada fotorama que el juego esté en funcionamiento.

Finalmente se debe mencionar que gracias a la libreía "blen2bam" pueden convertirse fácilmente la malla con los vértices y polígonos de una geometría creada en el programa

de modelado gráfico Blender, a un formato propio de Panda3d (.bam). Toda esto permite que el modelos se cargue de forma rápida y facilita además la implementación de los sólidos de colisión en el mapa del juego puesto que poseemos las coordenadas en forma binaria de los puntos de interés del modelo.

2.4. Herramientas de modelado

Tras haber podido analizar los datos y determinar correctamente el estímulo al que el sujeto le presta atención en cada momento lo único que resta es asignar cada acción a un movimiento dentro del juego. Sin embargo, antes de eso se debe diseñar un mapa para poder recorrer cuando atendemos a los estímulos. Para esta parte del proyecto nos hemos valido de Blender.

2.4.1. Blender

Blender (Blender (2022)) es un software de modelado 3-D de código abierto. Blender incluye modelado, renderizado, animación, edición de vídeo, texturizado, etc. En nuestro caso hemos utilizado esta herramienta para diseñar un laberinto que el jugador tendrá que recorrer utilizando las acciones e movimiento hacia adelante y girar para poder recorrer un plano. Los comandos utilizados para poder realizar el modelo 3D han sido los de añadir figura, escalado y extrusión, además de dotar a las paredes del mapa de distintos materiales para mejorar la sensación de movimiento y posicionamiento.

Para exportar el modelos hay dos opciones, utilizar la herramienta export del propio Blender y guardarlo como GLTF o utilizar en la consola, la utilidad ‘’blend2bam”. Esta librería se beneficia de que el sistema de ejes de Blender sea de mano derecha al igual que el de Panda3d, haciendo más fácil la exportación. Durante esta exportación se le pasa a Panda3d la información de los vértices y polígonos del modelo 3D, pero además también las texturas y colores. Por este motivo podemos pintar y texturizar el modelo dentro de Blender, que es mucho más intuitivo, en lugar de en Panda3d. Una vez hecho esto cargamos el modelos durante la tarea de inicialización del juego. Lo único que falta hacer con el mapa es activar colisiones con las paredes, lo cual se hace dentro del código del juego en Python.

2.5. Librerías

Por último hay que hablar de las distintas librerías utilizadas en el programa de Python. Se han utilizado librerías comunes para realizar cálculos matemáticos como ”numpy” o ”math”y también librerías para poder trabajar con archivos ”.csv”.

De entre las librerías debemos destacar las librerías de panda3d “direct”, que nos permite trabajar en el espacio 2D del interfaz de usuario, y la librería “panda3d.core” que habilita funcionalidades tan importantes como el uso de “Clock objects” para controlar la tasa de fotogramas, la definición de colisiones y el uso de “CardMaker” para crear los polígonos que se utilizan para los estímulos.

Es necesario nombrar también la librería “pylsl” que es la que contiene todas las funcionalidades necesarias para trabajar con los streming de datos del protocolo LSL. Finalmente la librería “sklearn.cross_decomposition” contiene el módulo “CCA” que como

su nombre indica se encarga de realizar todos lo cálculos pertinentes al análisis de correlación canónica.

Capítulo 3

Desarrollo de la aplicación

La meta de este proyecto es poder diseñar un videojuego sencillo en primera persona en el que el movimiento del personaje se realice haciendo uso de una BCI, más concretamente empleando la técnica SSEVP que mencionamos en el capítulo introductorio.

3.1. Funcionalidad de la aplicación desarrollada

En este apartado se explica la funcionalidad de la aplicación desarrollada. Como ayuda a la explicación se incluye el diagrama de la figura 3.1, donde pueden observarse los diferentes elementos hardware y software necesarios.

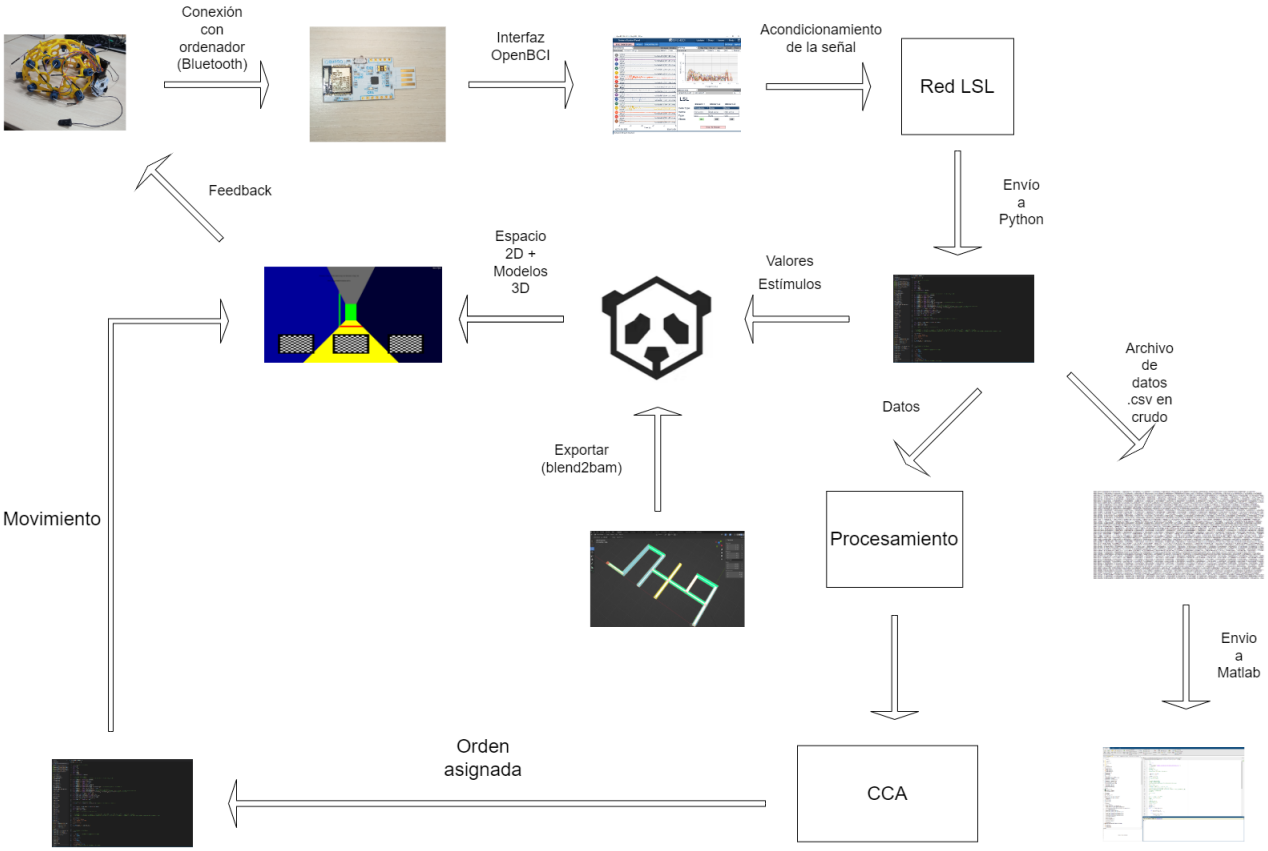

Figura 3.1: Diagrama de componentes del sistema

En este prototipo al usuario se le presentan tres estímulos diferentes en pantalla. Cada estímulo tendrá una frecuencia distinta. De izquierda a derecha dichos estímulos serán

de 10, 12 y 7,5 Hz idealmente. Los estímulos estarán siempre visibles para el jugador en primera persona, como parte de su interfaz de usuario. El jugador podrá recorrer un laberinto con pasillos y giros de 90º. El laberinto, como se aprecia en el diagrama 3.1 se debe exportar desde blender a panda3D para poder renderizarlo en el espacio 3D y los valores ideales de los estímulos tambieén deben ser conocidos por el motor en cada momento. El objetivo del usuario es ser capaz de maniobrar por ese laberinto utilizando los estímulos dados. Para ello, cada estímulo representa una acción que luego se describirá. Cuando el usuario centre su atención en uno de los estímulos la acción seleccionada quedará guardada como acción activa a la espera de confirmación. Tal y como vemos en el diagrama 3.1 el usuario debe llevar el casco con los electrodos y la placa Cyton para la adquisici'on de las señales. La placa Cyton debe estar conectada al ordenador mediante el pincho USB Bluetooth.

De esta manera el software OpenBCI GUI recoge los datos del puerto serial emulado por el pincho USB, y puede enviar los datos a traveś de la red con el protocolo "LSL". El software OpenBCI GUI también sirve para comprobar la calidad de la señal. Esa información recibida deberá pasar por un tratamiento y análisis (figura 3.1) para poder generar una ación.

Las funciones de los estímulos son las siguientes:

- Izquierdo: activa la orden de avanzar / retroceder. Si la orden activa es "avanzar" o "retroceder", esta se cambia por la contraria, si el sujeto centra la atención en este estímulo. Si la orden activa no es avanzar o retroceder, la orden de "avanzar" pasa a ser la orden activa.
- Central: activa la orden de girar a la derecha o la orden de girar a la izquierda. El funcionamiento es análogo al del estímulo izquierdo. La orden que se activa cuando ninguna de las dos órdenes mencionadas está como activa es "girar a la derecha"
- Derecho: confirma la acción activada y permite su aplicación. Esta acción no borrará el último tipo de movimiento activado, por lo que es posible realizar dicho movimiento de forma continuada sin la necesidad de voler a atender los otros estímulos, basta volver a atender al estímulo de confirmación.

La acción activada será mostrada en todo momento en pantalla para el conocimiento del usuario. Ese feedback (3.1) le será indicativo l ususario e donde se encuentra e influirá en la decisión de su siguiente acción. El objetivo final de la aplicación es que el usuario sea capaz de atravesar el laberinto usando combinaciones de las acciones disponibles. En la figura 3.2 muestra el aspecto del interfaz.

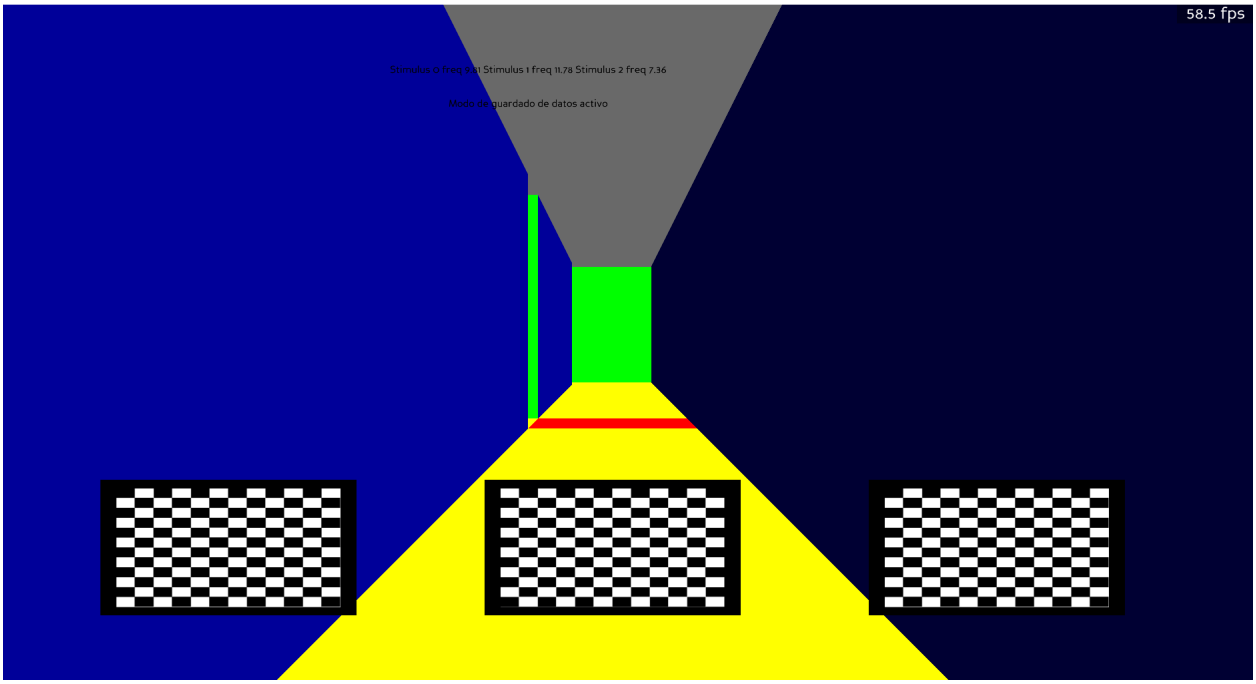

Figura 3.2: Interfaz del juego

La aplicación está orientada a realizar experimentos para comprobar el funcionamiento de interfaces cerebro computador. Por eso hay un modo manual y se realiza un almacenamiento de los datos, incluyendo detecciones y correlaciones. También se ha implenetado un modo automático de funcionamiento de la interfaz. Las principales características de estos modos se detallan abajo.

- **Modo manual**: Se iniciará la aplicación de forma normal. Cuando el usuario esté preparado pulsará la tecla “T” para indicar que está atendiendo al estímulo, con lo que los datos se empezarán a guardar en el archivo de datos en crudo y en un buffer circular en memoria. Cuando el usuario decida que ha pasado suficiente tiempo (se calculan 5 segundos normalmente) pulsará la tecla “Y” para indicar que ya no se está prestando atención. Con esta orden también se aplicará el procesamiento sobre el buffer de datos y el análisis de correlación, por lo que en pantalla saldrán los resultados de ese análisis y la ación correspondiente a la mayor correlación se guardará (o realizará si la corelación da como resultado el estímulo derecho). Este proceso se repite hasta que se llegue al final de laberinto.
- **Modo automático**: Al pulsar la tecla “K” se entra en este modo. Despueés de entrar en este modo el usuario debe atender a un estímulo. Durante un tiempo predeterminado (5 segundos) lo datos se irán guardando. Una vez pasado el lapso de tiempo se aplicará todo el proceso de detección y se moverá el avatar, es decir, al acabarse el tiempo es como si se pulsar la tecla ´´Y” en el modo manual. Una vez realizada la detección. El cuadrado negro que contiene el estímulo detectado se tornará rojo, indicando que ha sido el estímulo detectado y avisando al usuario que la detección ha concluido y que deberá atender al siguiente objetivo. Tres segundos después el cuadrado volverá a ser negro y se iniciará otra detección (como si se volviera a pulsar “T”).

3.1.1. Comprobaciones previas

Es importante señalar que estamos asumiendo en el uso de la aplicación que previamente se ha comprobado la señal del EEG con el interfaz OpenBCI GUI. Por eso, el primer paso será siempre comprobar el correcto funcionamiento del hardware y asegurarse que los datos que se muestran en la interfaz de usuario son correctos. Para ello conectamos el casco por BlueTooth y en la aplicación OpenBCI GUI, nos deberemos fijar en las funciones de “Time Series” y “FFT Plot”. Hay que asegurarse que los electrodos están todos en “Not Railed”, como se ve en la figura 3.3, donde todos los electros excepto los números 6 y 13 están en este estado. Estos dos electrodos necesitan ser reajustados hasta que su estado sea “Not Railed”.

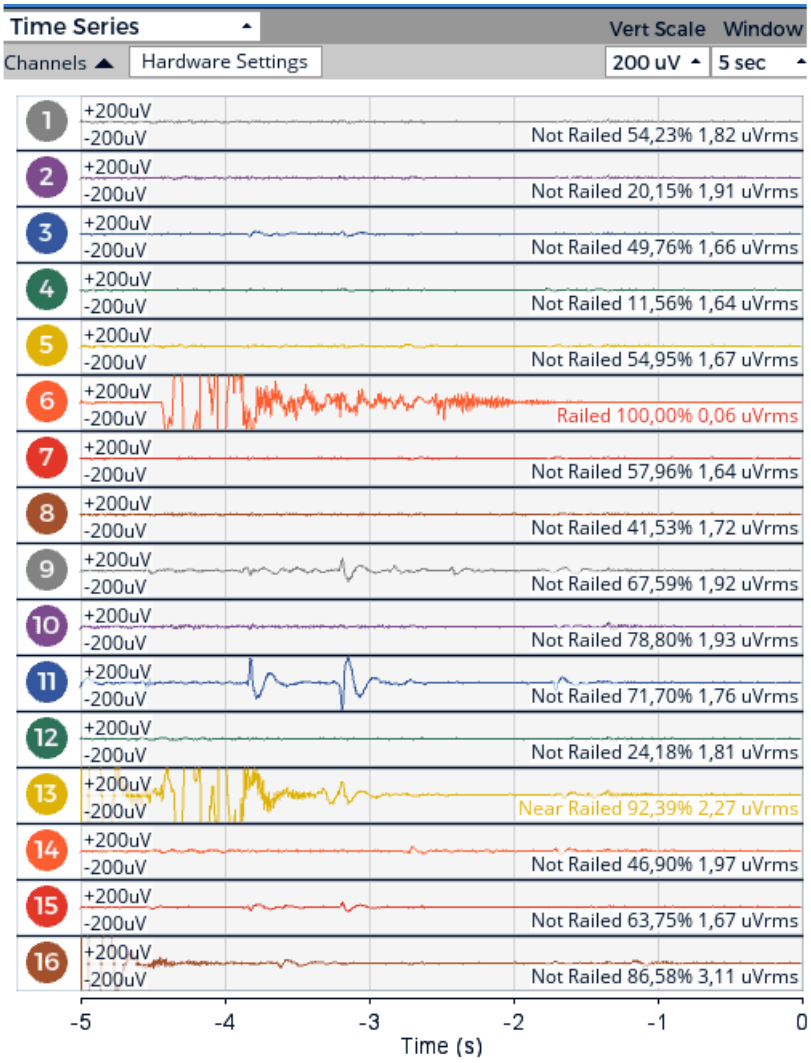

Figura 3.3: Distintos estados posibles de los electrodos y como los indica el GUI de OpenBci

Tras esta comprobación se pasa a realizar una prueba de calibración. Para ello le pedimos al usuario que mantenga los ojos cerrados. Tras unos segundos se debería observar un pico de potencia en la frecuencia en las cercanías de 10 Hz (en “FFT Plot”) correspondiente a las ondas alfa generadas al cerrar los ojos. Estas dos comprobaciones pueden realizarse en primera instancia tanto con la versión 4.2 como con la 5.1. Sin embargo, estas comprobaciones deberán hacerse antes, cada vez que el usuario se ponga el casco y vaya a hacer pruebas por lo que es mejor optar por la versión más actualizada.

Es igualmente importante destacar que todas las pruebas es mejor hacerlas haciendo uso de todos los canales disponibles, pues cuantos más canales se utilicen mejor se compensa el ruido externo en la técnica de análisis de correlación canónica.

3.2. Estructura básica de la aplicación

El software se ha desarrollado en Python, haciendo uso especialmente de las funciones provistas por la librería Panda3d, que como se ha explicado es un motor de videojuegos, y cuyo funcionamiento se basa en algunas ideas que se deben comprender para seguir la explicación del software desarrollado.

El motor Panda3d asume que el juego desarrollado es una instancia de la clase Showbase, que proporciona en Windows una ventana gráfica sobre la que se va a representar toda la parte gráfica y que asume la interacción con el usuario (lectura de teclado, por ejemplo).

La aplicación está formada por dos clases principales:

- La clase MyApp que representa a la aplicación gráfica y la interacción con el usuario.
- Una clase denominada Control_Data_Thread que representa un hilo de ejecución para la recepción, procesamiento y puesta a disposición del resto de la aplicación de los datos de EEG obtenidos de la red LSL.
- Un conjunto de funciones auxiliares entre las que se encuentran las necesarias para determinar de la señal capturada qué estímulo se está atendiendo por el usuario.

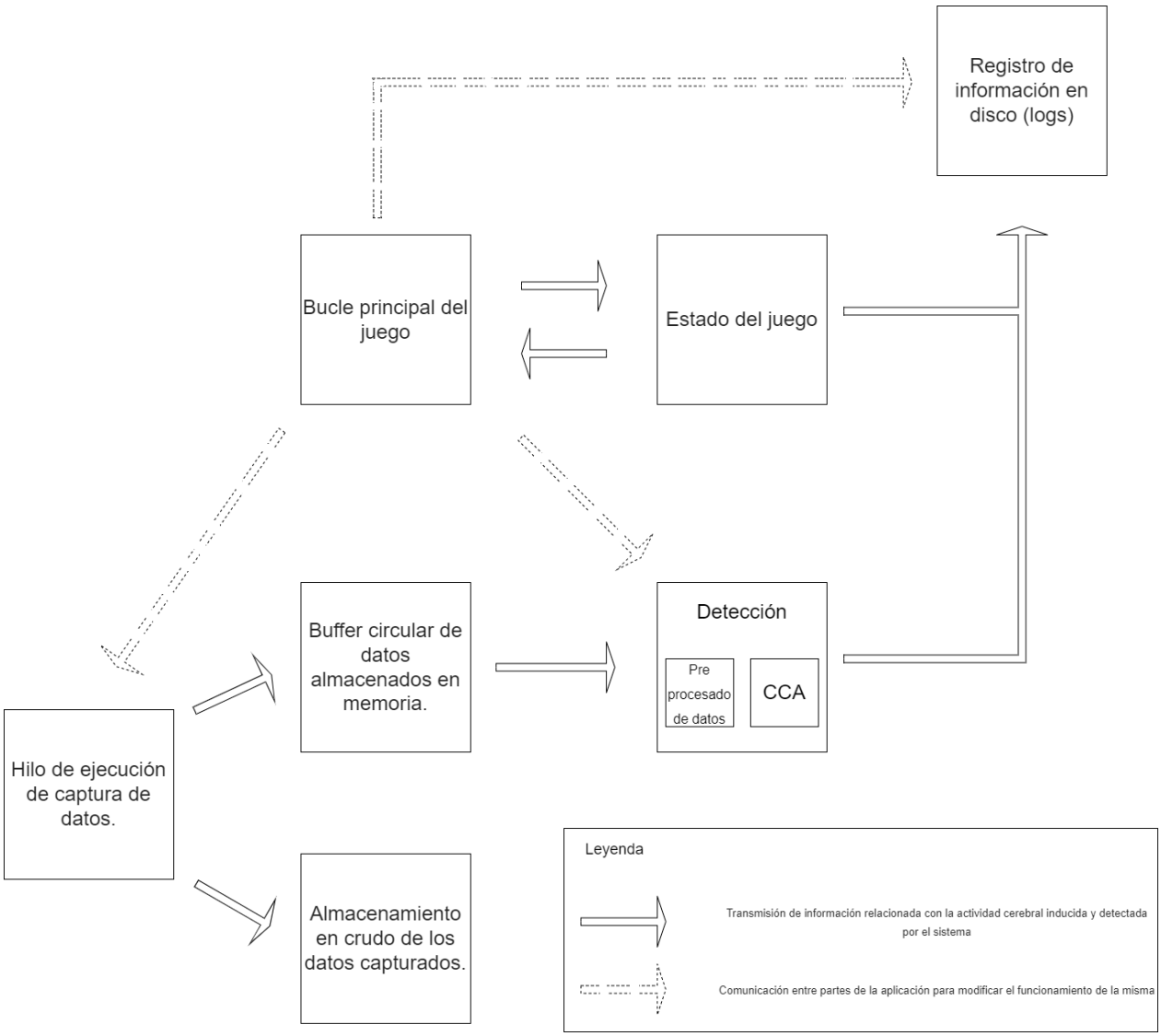

Figura 3.4: Flujo de información dentro del programa en Python

El diagrama 3.4 pretende explicar de manera sencilla el funcionamiento del prototipo durante la aplicación. Las flechas continuas indican la transmisión de información relacionada con la actividad cerebral y las discontinuas indican la transmisión de información entre las partes del proyecto para su correcto funcionamiento. Como pasos previos el usuario debe llevar puesto el dispositivo de captura del EEG (el casco) y la aplicación de OpenBci debe estar enviando información a la red “LSL”. Una vez hecho esto se lanza el prototipo, que está dividido en el bucle principal del juego (debe pasar por un proceso de inicializado) y el hilo de captura de datos. En el estado inicial tendremos el bucle principal del juego en funcionamiento pero no se estará recogiendo la información del EEG. Una vez se da la orden correcta en el bucle del juego (mediante tareas repetitivas) el hilo de captura empezará a guardar datos hasta que se le vuelva a indicar lo contrario. Cuando ocurre tal cosa, el bucle principal ordenará que los datos que se encuentran almacenados dentro del buffer pasen por el proceso de detección. Los resultados de esta detección pasarán al disco. Tras esto el bucle principal se encargará, mediante otra tarea, del movimiento del avatar del usuario. El estado del juego afectará constantemente al bucle principal, esto ocurre por ejemplo en el modo automático de funcionamiento, donde una vez acabado el movimiento se debe dar la orden de volver a iniciar la atención.

3.3. Etapas del flujo del programa

Durante el transcurso de la ejecución del programa principal del juego podemos distinguir diferentes partes:

- En el punto de entrada del programa se llama al constructor de la clase MyApp. Desde la inicialización de esta clase se llama al constructor del hilo que se encarga de recoger los datos.
- En el constructor de MyApp se llama al constructor de la clase base, se inicializan los parámetros generales y de la escena, se llama a la función de inicialición y finalmente se lanzan las tareas y threads necesarios.
- En la función de inicialización mencionada se construye la escena a partir de los parámetros básicos del juego y se asignan las funcionalidades a las distintas teclas.
- Las clase tarea (Task) en Panda3D, permiten ejecutar código de forma sincronizada con el bucle principal de juego. En este caso las tareas usadas realizan distintas funciones y se llaman en cada uno de los fotogramas dentro del bucle principal. Se inicializan mediante una función que es llamada desde el constructor de MyApp como se ha mencionado.

En las siguientes secciones se pasarán a describir las partes más importantes del videojuego y en qué parte de la ejecución tienen lugar.

3.4. Generación del estímulo

El trabajo de desarrollo de la aplicación comenzó buscando la forma de generar los estímulos visuales que vamos a utilizar.

```
1
2         self.stimulus_texture_files=["damero1.jpg","damero2.jpg","boton2.png","
              boton2.png"]
3         self.boxes_files = ["black.png","red.png"]
4         self.stimulus_type=[0,1,0] #1 is for two-pattern and 0 is for single
              pattern
5         self.stimulus_selected_texture_set=[0,0,1]
6
7         # Pos. stimuli
8         self.stimulus_frame=[(-0.775,-0.425,-0.775,-0.425)
              ,(-0.175,0.175,-0.775,-0.425),(0.425,0.775,-0.775,-0.425)]
9         self.boxes_frame=[(-0.8,-0.4,-0.8,-0.4),(-0.2,0.2,-0.8,-0.4)
              ,(0.4,0.8,-0.8,-0.4)]
```

Código 3.1: Selección de texturas y posicionamiento de estímulos

Como se puede ver en el código 3.1 es necesario especificar los archivos de texturas que se van a utilizar a utilizar en el constructor de MyApp. Posteriormente se especifican unas medidas que se utilizan para ubicar los estímulos (tres en la versión actual) sobre el espacio 2D del juego.

Aunque se trata de un juego 3D desarrollado en un motor gráfico 3D, Panda3D establece una espacio de “renderización” 2D para ubicar los elementos del interfaz del usuario, como leyendas de texto, botones o en este caso imágenes construidas con archivos gráficos que se cargan en memoria y que durante la ejecución se pueden variar.

```
1
2           cards=[CardMaker(str(x)) for x in range(self.nstimuli)]
3           boxes = [CardMaker(str(x)) for x in range(self.nstimuli)]
4
5           for ind in range(self.nstimuli):
6               cards[ind].setFrame(self.stimulus_frame[ind])
7               boxes[ind].setFrame(self.boxes_frame[ind])
8
9           self.icon=[ render2d.attachNewNode(cards[x].generate()) for x in range(
                len(cards))]
10          for icon in self.icon:
11              icon.setBin("fixed",0)
12
13          self.box=[ render2d.attachNewNode(x.generate()) for x in boxes]
14          self.myTexture = [self.loader.loadTexture(file) for file in self.
                stimulus_texture_files]
15          self.colors = [self.loader.loadTexture(file) for file in self.boxes_files
                ]
16          for ind in range(len(self.icon)):
17              ind_texture=self.stimulus_selected_texture_set[ind]*2
18              stimulus_type = self.stimulus_type[ind]
19              self.box[ind].setTexture(self.colors[0], 1)
20              self.icon[ind].setTexture(self.myTexture[ind_texture], 1)
```

Código 3.2: Generación del polígono para el estímulo

Como se muestra en el código 3.2, se pasan dichas coordenadas a la función constructor de la clase “Cardmaker” para crear las formas rectangulares que pasarán a ser los estímulos. Tras esto se añaden las “cards” al árbol de renderizado 2D. Antes de continuar, es necesario cargar en el juego las texturas que se seleccionaron antes. El bucle en la parte final del código 3.2 se encarga de aplicar las primeras texturas a los estímulos. Como se observa en 3.2 hay un segundo grupo de objetos llamados “boxes” que simplemente son unos rectángulos negros que contienen al estímulo para evitar que estos se confundan con el mapa mientras se recorre el laberinto. Todo esta parte de la creación del marco donde se incluyen los estímulos visuales se realiza en la función de inicialización del juego.

En este punto ya están los estímulos colocados en la escena 2D, así que lo que resta es hacer que parpadeen a la frecuencia a la que queremos. Para ello, la decisión sobre el cambio de una textura se realiza a partir de las ecuaciones 3.1 y 3.2:

$$d = \begin{cases} 1 & \text{si } c \geq S \\ 0 & \text{en otro caso} \end{cases} \tag{3.1}$$

$$c' = \begin{cases} c+1 & \text{si } d = 0 \\ c = 0 & \text{en otro caso} \end{cases} \tag{3.2}$$

donde d es el resultado de la decisión, c es un contador que se inicializa a 0 y que se actualiza según la ecuación 3.2 y S es una constante propia del estímulo que determinará junto a la tasa de frames F_r la frecuencia de modificación del estímulo visual. Si d vale 1 se considera que debe cambiarse o ocultarse la textura del estímulo y si es 0 no se modifica.

Por ejemplo, supongamos que $F_r = 60$ y $S = 5$. Cuando c toma el valor 5, d pasa a valer 1 (decisión de modificar el estímulo) y el nuevo valor del contador se establece otra vez a 0. Esto supone que cada 5 frames se cambia el estímulo. Como 60 dividido entre 5 es 12, esto quiere decir que la frecuencia del cambio del estímulo es 12 Hz. En general la frecuencia de modificación del estímulo es:

$$f_m = \frac{F_r}{S} \tag{3.3}$$

Pero esta no es realmente la frecuencia del estímulo, sino la frecuencia con la que el software opera un cambio en el objeto visualizado.

Si este cambio establece un ciclo, por ejemplo, establece una secuencia imagen 1 - imagen 2 - imagen -1 - imagen 2 etc,.. con N imágenes por ciclo, la frecuencia del estímulo es:

$$f_s = \frac{fm}{N} = \frac{F_r}{NS} \tag{3.4}$$

En este programa los ciclos se realizan con $N = 2$, así que para este sistema:

$$f_s = \frac{F_r}{2S} \tag{3.5}$$

En cuanto a la frecuencia base de la señales oscilatorias inducidas en el potencial de estado estacionario, hay que distinguir dos casos. En el primer caso, lo que se hace al operar el cambio en el estímulo es “esconder” la imagen que elicita la respuesta cerebral, es decir, si el ciclo es simplemente imagen - fondo, la frecuencia base de las señal inducida es igual a la frecuencia del estímulo:

$$f_i = \frac{F_r}{2S} \tag{3.6}$$

Pero si establecemos un ciclo con dos imágenes que elicitan cambios en la respuesta cerebral, es decir imagen 1 - imagen 2, la frecuencia inducida será el doble de la frecuencia del estímulo:

$$f_i = \frac{F_r}{S} \tag{3.7}$$

La programación de estas ecuaciones en la aplicación es facilitada por las tareas de Panda3d. Como se menciona en la descripción del motor, las tareas se ejecutan en cada iteración del bucle de juego principal (cada fotograma). Así, se utiliza la tarea descrita en el código 3.3 para cambiar las texturas de los estímulos. La correspondencia con el parámetro S de las ecuaciones anteriores es la variable “self.divs” que controla el divisor que se le aplicará a los 60 FPS para obtener la frecuencia de estímulos. Estos valores (uno para cada estímulo) los proporciona el constructor la clase App. Entonces podemos ver que la tarea definida con la función “stimulus”, se encarga de realizar las comprobaciones y actualizaciones del contador en “self.conts”, y de cambiar u ocultar las texturas correspondientes a cada caja de visualización (las cajas de visualización están en la lista self.icon)

```
    def stimulus(self, index,task):
        if keyMap["end_game"]:
            return task.done
        texture_set=self.stimulus_selected_texture_set[index]
        if self.conts[index] < (self.divs[index]):
            self.icon[index].setTexture(self.myTexture[2*texture_set], 1)
            self.conts[index] += 1
        else:
            if self.stimulus_type[index]==1:
                self.icon[index].setTexture(self.myTexture[2*texture_set+1], 1)
            else:
                self.icon[index].hide()
            self.conts[index] += 1
            if self.conts[index] >= (2 * self.divs[index]):
                self.icon[index].show()
                self.conts[index] = 0
        return task.cont
```

Código 3.3: Tarea que se encarga de cambiar las texturas de los estímulos

Con todo esto vemos que uno de los parámetros básicos que gobiernan las frecuencias inducidas en el cerebro, es la tasa de frames de la aplicación.

```
        globalClock.setMode(ClockObject.MLimited)
        globalClock.setFrameRate(self.frame_rate)
```

Código 3.4: Código para conseguir una tasa de frames estables

Puesto que lograr una tasa de frames estable es fundamental para la correcta operación del BCI hay que configurar el motor Panda3d para que lo logre en lo posible. En el código 3.4 se activa la opcioń “MLimited”, la cual hace que la aplicación vaya a la mayor tasa de fotogramas posible, salvo que intente ir por encima de lo establecido en “setFrameRate” en cuyo caso reduce la velocidad del bucle del juego para quedarse en lo establecido. Este ajuste debe hacerse durante la inicialización de la aplicación.Es necesario mencionar que la tasa de refresco que tenga la pantalla que se utilice a la hora de lanzar el videojuego debe ser suficiente para reflejar correctamente las frecuencias de los estímulos.

```

confvariables = '''
window-title Estimulo
show-frame-rate-meter True
fullscreen 0
framebuffer-srgb 0
'''
loadPrcFileData("", confvariables)
```

Código 3.5: Configuración de la ventana del videojuego

Para controlar en todo momento la tasa de frames en las pruebas se utiliza una característica del motor Panda3d, que permite mostrar esta tasa en la ventana de la aplicación. Para hacer esto, es necesario establecerlo como “parámetro de configuración”. En 3.5 se están indicando algunos de estos parámetros (esto debe hacerse antes de llamar al constructor de la aplicación). Entre ellos está la opción de pantalla completa, el título del juego y la opción de mostrar la tasa de frames.

3.5. Flujo de datos

3.5.1. Conexión al stream de datos

El otro núcleo de nuestro prototipo además de la generación del estímulo es la adquisición de datos. Durante la inicialización del juego se llama a la función “run_tasks_and_threads”. Dentro de ella encontramos lo que se muestra en el código 3.6 En estas líneas destacamos:

- Primero se lanza un hilo de ejecución encargado de la obtención de los datos. Este hilo se basa en la librería de Python “threading”.
- Este hilo, en perpetua ejecución cíclica desde que se lanza, tiene un estado interno que puede modificarse para variar su comportamiento. Cuando se lanza, en principio no asume una conexión con el stream de datos en la capa LSL, pero la tarea “conectar” que se lanza después del hilo, se encarga de ordenar esa conexión en el hilo.

```
1         self.ctr_data_thread.start_storing()
2         self.thread_data=data_module.Raise_Exception_Thread(target=self.
              ctr_data_thread.thread_data)
3         self.taskMgr.add(self.conectar,"conectar")
```

Código 3.6: Función “run_tasks_and_threads” que lanza el hilo de adquisición y almacenamiento de datos y la tarea que se encarga de la conexión a una fuente de datos.

```
1     def conectar(self, task):
2         try:
3             self.ctr_data_thread.connect()
4         finally:
5             if self.ctr_data_thread.connected:
6                 self.thread_data.start()
7             else:
8                 print("EEG stream not connected")
9
10        return task.done
```

Código 3.7: Tarea para conectar streaming

Por tanto, vemos en la tarea mostrada en 3.7 como esta se encarga de indicar al hilo de recogida de datos que debe conectarse a las placa.

La función connect, que vemos en el código 3.8 es uno de los métodos de la clase Control_Data_Thread y se encarga de buscar un “streaming de datos LSL” con el tipo EEG, disponible y conectarse a él.

```
    def connect(self):
        try:
            self.streams = resolve_byprop('type', 'EEG',timeout=5)
            print(self.streams)
            if len(self.streams)>=1:
                self.connected=True
                print('rate:', self.streams[0].nominal_srate())
                self.inlet = StreamInlet(self.streams[0])
            else:
                self.connected=False
                print("EEG stream not resolved")
```

Código 3.8: Función para conectar streaming

3.5.2. Hilo de adquisición de datos

Si la búsqueda de un streaming de datos da resultados se entra en el bucle que se muestra en el código 3.9. En caso contrario cuando se cumpla el tiempo de espera el videojuego se lanzará con normalidad y el usuario podrá moverse por el escenario con el uso del teclado. En todo momento puede volver a intentarse una conexión con el streaming de datos pulsando la tecla “R”.

```
1  def thread_data(self):
2          # first resolve an EEG stream on the lab network
3
4          atencion1 = False
5          noatencion2 = False
6
7
8          self.finish_state=False
9          while not self.finish_state:
10             sample, timestamp = self.inlet.pull_sample(timeout=0.0)
11
12             if sample is not None and self.list[0] == False and self.list[1] ==
                   False:
13                 str_data=''.join([str(timestamp)]+\
14                     [',{}'.format(x) for x in [float('nan')] * 16])
15                 self.store(str_data)
16             elif sample is not None and self.list[0] == True:
17                 str_data=''.join([str(timestamp)]+\
18                     [',{}'.format(str(x)) for x in sample])
19                 str_data=str_data[0:-1]
20                 self.store(str_data)
21                 self.buffer[self.index, 0] = timestamp
22                 self.buffer[self.index, 1:17] = sample
23                 self.index += 1
24
25                 if self.index == (self.max_data):
26                     self.index = 0
27                     self.full = True
28
29             elif sample is not None:
30                 str_data=''.join([str(timestamp)]+\
31                     [',{}'.format(x) for x in [float('nan')] * 16])
32                 self.store(str_data)
33
34         self.end_storing()
35         self.inlet.close_stream()
```

Código 3.9: Función para guardar datos en memoria

El código 3.9 muestra el hilo de ejecución que realiza la captura y el guardado de los datos. Como podemos observar el hilo entra en un bucle infinito que solo podrá romperse cuando se indique que se ha finalizado el streaming de datos, al modificar una de las variables de su estado. Dentro de este bucle tenemos dos posibilidades. La primera de ellas (línea 13) es que se considere que el usuario no está atendiendo al estímulo, por lo que los datos recogidos durante ese periodo de tiempo serán inservibles. Por ello, en lugar de guardar en el archivo de datos (utilizado para pruebas de calibración y análisis) los valores reales se guardan “nan” lo que facilita su eliminación durante el procesamiento.

Si por el contrario estamos en el segundo caso (línea 17) los datos reales serán almacenados en el archivo externo pero además se guardarán también en un **buffer cíclico** de datos para su uso posterior en el movimiento por el mapa. Dicho buffer de datos tiene una capacidad aproximada de 5 segundos en la configuración actual. Como es

cíclico, en caso de que se llene se empezarán a sustituir los primeros datos por los nuevos que se vayan recibiendo. El formato esperado de los datos incluye en cada muestra: el tiempo de la captura, y cada uno de los 16 canales del EEG en formato flotante.

En resumen, durante la inicialización del juego se busca un streaming de datos al que conectarse. La adquisición de estos datos se realizará en un hilo diferente al del juego con el objetivo de evitar que la frecuencia de muestreo de los datos de LSL se reduzca a la tasa de frames del juego, en lugar de la frecuencia necesaria para evitar que se pierdan datos del EEG. Durante la captura de datos, la información se guarda en dos lugares distintos: un archivo ".csv" para poder analizarlo a posteriori y un buffer de datos en memoria que será utilizado para realizar las acciones dentro del juego en tiempo real. Cada vez que se pase del estado de "no atención" al de "atención" se comienza a acumular datos en el buffer.

3.6. Procesamiento de los datos

Una vez obtenido el buffer de datos se requiere un procesamiento previo antes de aplicar el mťodo de correlación canónica. Hay que señalar que el software permite procesar tanto el buffer de datos en memoria, como el archivo de datos almacenado una vez concluido un experimento.

```
1
2    if archivo:
3
4        lista = []
5        with open(data_file, newline='') as f:
6            reader = csv.reader(f)
7            for row in reader:
8                lista.append(row)
9
10       y = np.asarray(lista)
11    else:
12       y = data_file
13
14    nseries = y.shape[1]
15    default_output=(None,None)
```

Código 3.10: Llegada de datos

Como se aprecia en el código 3.10 el primer paso es trasladar los datos a una variable única, bien provengan del archivo registrado o bien del buffer de datos en memoria. Si es un archivo de datos debe leerse el archivo y trasladarlo a un array en memoria. Si en cambio se recibe el buffer de datos no hay que realizar ningún cambio.

Antes de continuar con la explicación es necesario entender que el formato en el que se guardan los datos previamente a su procesamiento se deriva del utilizado en (Wang et al. (2016)), donde se incluyen archivos para realizar pruebas con sistemas de detección de potenciales evocados de estado estacionario.

Los datos se guardan en un array con las siguientes dimensiones:

- Canal del EEG.
- Datos registrados.
- Target

- Sesión

El tipo de experimento para el que esta estructura está pensada es la de un individuo que atiende uno a uno una serie de estímulos o targets. Para cada target se guardan los datos del EEG de ese individuo. Cuando el individuo completa la secuencia de targets, se termina una sesión y el individuo puede comenzar otra atendiendo nuevamente a la serie de targets.

```
1
2   dat = y.astype(np.float64)
3
4     indarr=np.where(np.logical_not(np.isnan(dat[:,6])))[0]
5     # Indices de los puntos de comienzo de los bloques
6     if indarr.size>1:
7         indstart=np.where(np.diff(indarr)>1)[0]
8         start_points=np.zeros((indstart.size+1,),dtype=np.int64)
9         start_points[0]=indarr[0]
10        start_points[1:(indstart.size+1)]=indarr[indstart+1]
11    else:
12        return default_output
13
14    stop_points=np.zeros((indstart.size+1,),dtype=np.int64)
15    stop_points[0:indstart.size]=indarr[indstart]
16    stop_points[-1]=indarr[-1]
17
18    sesion_size= int(np.min(stop_points-start_points))
19    print("sesion_size {}".format(sesion_size))
20
21    num_atenciones= indstart.size + 1
22    num_sesiones = num_atenciones // targets_por_sesion
23    if ((num_atenciones %targets_por_sesion)>0):
24        num_sesiones += 1
```

Código 3.11: Construcción de sesiones

Asumiendo el escenario descrito para el registro de los datos en el archivo, el código en 3.11 pretende averiguar las puntos de las datos donde se comienza a atender a los estímulos, es decir, se buscan los índices donde los datos pasan de “nan” a valores reales. También se busca calcular el tamaño y cantidad de las sesiones de pruebas a considerar valiéndose del número de grupos con datos reales y de los targets que se consideraron para el experimento.

```
    nw = 125 # Ventana

    if(nw > sesion_size):
        nw = sesion_size

    if nw==0:
        return ([None]*3,None)

    k = sesion_size // nw
    sesion_size = k * nw
```

Código 3.12: Ajuste al tamaño de las sesiones

Después en el código 3.12 nos indica que cada sesión deberá tener un tamaño de datos registrados múltiplo de cierta ventana “nw”. En caso de ser menor tomaremos ´´nw” como tamaño. Si no, deberemos tomar como longitud el menor valor por debajo de la menor serie que sea múltiplo de “nw”. Esta parte del código está orientada a métodos de preprocesamiento y detección que trabajen con ventanas sobre los datos.

```
    data = np.ndarray(shape=(nseries,sesion_size,targets_por_sesion, num_sesiones
        ), dtype=np.float64)

    target = 0
    sesion = 0
    for m in range(nseries):
        for n in range(num_atenciones):
            if m == 0 or not acondiciona_flag:
                data[m,:,target,sesion] = dat[start_points[n]:(start_points[n]+
                    sesion_size), m]
            else:
                data[m,:,target,sesion] = acondiciona(dat[start_points[n]:(
                    start_points[n]+ sesion_size), m], nw)
            target +=1
            if target == targets_por_sesion:
                target = 0
                sesion += 1
        target = 0
        sesion = 0
```

Código 3.13: Ordenamiento de muestras

Finalmente, el código mostrado en 3.13 separa finalmente las muestras por estímulo al que se ha atendido, electrodo del que se ha tomado y sesión. En el caso de que los datos no provengan de un archivo registrado según las condiciones del tipo de experimento descrito, sino del buffer de datos en memoria utilizado para la detección en tiempo real, la sesión y el estímulo siempre serán únicos.

3.7. Señales de referencia

Antes de aplicar el método de anáisis de correlación canónica (CCA) es necesario disponer de las señales de referencia cuya frecuencia coincida con la de los estímulos.

```
def referencia(tend, f, nh, Fs):
    n_data = Fs * tend
    signal = np.empty((2 * nh, n_data), float)
    sin = np.zeros(shape=(len(f),nh,n_data))
    cos = np.zeros(shape=(len(f),nh,n_data))
    ref = np.zeros(shape=(len(f),2*nh,n_data))
    cont = 0
    for k in range(len(f)):
        for i in range(nh):

            sin[k,i,:]=np.sin(np.pi*2*(i+1)*f[k]*np.arange(n_data)/Fs)
            cos[k,i,:]=np.cos(np.pi*2*(i+1)*f[k]*np.arange(n_data)/Fs)

            ref[k,cont,:] = sin[k,i,:]
            cont += 1
            ref[k,cont,:] = cos[k,i,:]
            cont += 1

        cont = 0
    return sin, cos, ref
```

Código 3.14: Señales de referencia

En el código 3.14 se muestra la generación de las distintas señales de referencia mediante una función dedicada que toma como argumentos, el tamaño de la sesión obtenido anteriormente), la frecuencia de cada uno de los estímulos (generamos varias referencias, tantas como estímulos) y el número de armónicos que queremos tomar en cuenta (las pruebas siempre se hacían con tres armónicos para cada referencia).

3.8. Análisis de correlación canónica

Una vez procesados los datos y creadas las señales de referencia ya podemos aplicar el método de correlación canónica. De esta tarea se encarga una función que se encuentra definida junto al código encargado de procesar la información.

```
def correla(re, data,channels=None):
    ntargets=re.shape[0]
    nsessions=data.shape[-1]
    ntarpersession=data.shape[-2]
    if channels is None:
        channels=np.arange(data.shape[0])

    corrs=np.zeros((ntargets,ntarpersession,nsessions))
    detection=np.zeros((ntarpersession,nsessions))
    my_cca = CCA(n_components=1)
    for session in np.arange(nsessions):
        for observation in np.arange(ntarpersession):
            y = data[channels,:,observation,session]
            for target in np.arange(ntargets):
                x = re[target]
                try:
                    my_cca.fit(np.transpose(y), np.transpose(x))
                except:
                    print("Fitting Error")
                    corrs[target,observation,session]=np.nan
                    detection[observation,session]=np.nan
                else:
                    xp, yp = my_cca.transform(np.transpose(y), np.transpose(x))
                    corrs[target,observation,session]= np.corrcoef(xp[:,0], yp
                        [:,0])[0,1]
                    detection[observation,session]=np.argmax(corrs[:,observation,
                        session])

    return corrs, detection
```

Código 3.15: Método CCA

En el código 3.15 se pasan las referencias y los datos ya procesados. Se puede observar que la correlación se realiza dentro de un bucle “for”, puesto que el método requiere que los datos obtenidos sean comparados con cada una de las referencias. En este caso se deberá calcular tres veces la correlación, una por cada target. Una vez hecho esto, la función devolverá el valor de correlación correspondiente a cada frecuencia de referencia y el indicador del target con el que la correlación haya sido mayor. En caso de ser un buffer de datos el que entra en la función solo habrá un target con correlación máxima, si no habraá un resultado por cada estímulo y sesión de toma de datos.

Las funciones de procesamiento de los datos, la creación de la señal de referencia y la correlación se llaman en tareas repetitivas durante el bucle principal del juego. En la siguiente sección se explicarán más en detalle el momento específico en el que esto tiene lugar.

3.9. Tareas repetitivas dentro del bucle principal

Durante el bucle principal del juego las tareas son las encargadas de hacer que se produzcan los distintos eventos dentro del juego. A continuación se describen las más importantes que no han sido mencionadas todavía.

3.9.1. Movimiento en el laberinto

La tarea descrita en los códigos 3.16 y 3.17 se encarga de mover al usuario por el laberinto dependiendo del último estímulo detectado. Estos movimientos se realizarń siempre y cuando se haya detectado la confirmación (atención fijada en el estímulo de la derecha). La cantidad que se mueve o gira el usuario es pre-establecida por el constructor de la aplicación

```
1     def head_movement(self, task):
2
3         if self.action_confirmation != None and self.last_detected_action != None
              :
4             x = self.camera.getX()
5             y = self.camera.getY()
6             z = self.camera.getZ()
7             h = self.camera.getH()
8             p = self.camera.getP()
9             r = self.camera.getR()
10
11            if self.last_detected_action == 0:
12                y = y + self.movdelta*math.cos(h*np.pi/180)
13                x = x - self.movdelta*math.sin(h*np.pi/180)
14                self.movacc += self.movdelta
15                if self.movacc>=self.movtarget:
16                    #self.last_detected_action = None
17                    self.action_confirmation = None
18                    self.movacc=0.0
19                self.camera.setPos(x, y, z)
20
21            if self.last_detected_action == 3:
22                y = y - self.movdelta*math.cos(h*np.pi/180)
23                x = x + self.movdelta*math.sin(h*np.pi/180)
24                self.movacc += self.movdelta
25                if self.movacc>=self.movtarget:
26                    #self.last_detected_action = None
27                    self.action_confirmation = None
28                    self.movacc=0.0
29                self.camera.setPos(x, y, z)
```

Código 3.16: Tarea para moverse por el laberinto sin teclado (parte 1)

```
1
2               if self.last_detected_action == 1:
3                   h -= self.rotdelta
4                   self.rotacc += self.rotdelta
5                   if self.rotacc>=self.rottarget:
6                       self.action_confirmation = None
7                       self.rotacc=0.0
8                   self.camera.setHpr(h, p, r)
9
10              if self.last_detected_action == 4:
11                  h += self.rotdelta
12                  self.rotacc += self.rotdelta
13                  if self.rotacc>=self.rottarget:
14                      self.action_confirmation = None
15                      self.rotacc=0.0
16                  self.camera.setHpr(h, p, r)
```

Código 3.17: Tarea para moverse por el laberinto sin teclado (parte 2)

3.9.2. Tarea principal para la interacción con el usuario

La tarea principal para la interacción con el usuario se encarga de que la aplicación responda a las acciones en el teclado o al sistema BCI. Es una tarea repetitiva que se ejecuta una vez por fotograma. En el código esta tarea se representa por la función denominada “teclado”

Antes de poder implementar acciones con el teclado se debe definir un ‘’KeyMap” o mapa de claves. Este mapa de claves es un diccionario con partes del estado del juego que son necesarias consultar regularmente y que en su mayor parte son modificadas por las acciones del teclado. Esta definición debe hacerse antes de llamar al constructor de la aplicación. En 3.18 se muestra nuestro KeyMap.

```
1
2  keyMap = {
3
4  "up": False, # Movimientos del juego con el teclado
5  "down": False,
6  "left": False,
7  "right": False,
8  "go":False,
9  "back":False,
10 "rotate":False,
11 "negrotate":False,
12 "modo_automatico":False,
13 "d1": 0, # Modificaci'on de la frecuencia del estimulo
14 "d2": 0,
15 "d3": 0,
16 "attention": False, #Modo manual de observacion y deteccion de target: comienzo
17 "not_attention": False, # Modo manual de observacioncion y deteccion de target :
       final (deteccion)
18 "textura_cambio_izquierda":0, # Cambios de textura en los targets
19 "textura_cambio_derecha":0,
20 "textura_cambio_centro":0,
21 "estimulo_izquierda":False, # Marcar en el archivo de registro el target deseado
22 "estimulo_central":False,
23 "estimulo_derecha":False,
24 "reconectar": False, # Ordenar la reconexi'on con LSL (EEG)
25 "start_recording" : False, # Guardado de datos: el archivo previo es destruido y
       se comienza uno nuevo
26 "stop_recording": False, # Final de la grabacion de datos. El archivo resultante
       es analizado como un registro multisesion
27 "end_game":False
28 }
```

Código 3.18: KeyMap del prototipo

Una vez definido el “KeyMap”, durante la etapa de inicialización se asignarán las teclas a las distintas acciones. En los códigos 3.19 y 3.20 se muestran las asignaciones utilizadas.

```
1
2           self.accept("a", updateKeyMap, ["left",  True])
3           self.accept("a-up", updateKeyMap, ["left",  False])
4           self.accept("d", updateKeyMap, ["right",  True])
5           self.accept("d-up", updateKeyMap, ["right",  False])
6           self.accept("q", updateKeyMap, ["up",  True])
7           self.accept("q-up", updateKeyMap, ["up",  False])
8           self.accept("e", updateKeyMap, ["down",  True])
9           self.accept("e-up", updateKeyMap, ["down",  False])
10          self.accept("w", onceUpdateKeyMap, ["go"])
11          self.accept("s", onceUpdateKeyMap, ["back"])
12          self.accept("n", onceUpdateKeyMap, ["rotate"])
13          self.accept("b", onceUpdateKeyMap, ["negrotate"])
14
15
16          # Cambiar frecuencia estimulo
17          #
18          self.accept("1", conmuteKeyMap, ["d1",  5])
19          self.accept("2", conmuteKeyMap, ["d2",  5])
20          self.accept("3", conmuteKeyMap, ["d3",  5])
21
22          ndivs=len(self.divs)
23          if ndivs>=1:
24              updateKeyMap("d1",self.divs[0]-1)
25          if ndivs>=2:
26              updateKeyMap("d2",self.divs[1]-1)
27          if ndivs>=3:
28              updateKeyMap("d3",self.divs[2]-1)
```

Código 3.19: Asignación de teclas (Parte 1)

```
1
2
3           # Cambiar texturas
4           self.accept("g", conmuteKeyMap, ["textura_cambio_izquierda",2])
5           self.accept("h", conmuteKeyMap, ["textura_cambio_centro",2])
6           self.accept("j", conmuteKeyMap, ["textura_cambio_derecha",  2])
7
8           if ndivs>=1:
9               updateKeyMap("textura_cambio_izquierda",self.
                    stimulus_selected_texture_set[0])
10          if ndivs>=2:
11              updateKeyMap("textura_cambio_centro",self.
                    stimulus_selected_texture_set[1])
12          if ndivs>=3:
13              updateKeyMap("textura_cambio_derecha",self.
                    stimulus_selected_texture_set[2])
14
15
16          # Guardar stream
17          self.accept("l", updateKeyMap, ["end_stream",  True])
18          self.accept("p", updateKeyMap, ["start_stream",  True])
19
20          # Marcar atencion
21          self.accept("t", updateKeyMap, ["attention",  True])
22          self.accept("y", updateKeyMap, ["not_attention",  True])
23          self.accept("k", updateKeyMap, ["modo_automatico", True])
24
25          # Marcar targets
26          self.accept("z", updateKeyMap, ["estimulo_izquierda",  True])
27          self.accept("x", updateKeyMap, ["estimulo_central",  True])
28          self.accept("c", updateKeyMap, ["estimulo_derecha",  True])
29
30
31          # General
32          self.accept("m-up",updateKeyMap,["end_game",True])
33          self.accept("r", updateKeyMap, ["reconectar",  True])
34          self.accept("o", updateKeyMap, ["start_recording",True])
35          self.accept("p", updateKeyMap, ["stop_recording",  True])
```

Código 3.20: Asignación de teclas (Parte 2)

Hecho todo esto se puede integrar la tarea “teclado” en el bucle del juego. La tarea “teclado” contiene realmente las partes más importantes de la lógica del juego. Entre todas las funcionalidades destacan: la medida del tiempo transcurrido desde la última vez que se lanzó la tarea, la actualización del valor conseguido de la tasa de frames por parte de la aplicación, la actualizaci’on de la posición y orientación de la cámara que puede haber variado por la acción del sistema de colisiones, la consulta del estado del juego que varía por la acción del usuario en el teclado y el BCI y la realizaci’on de las operaciones que deben derivarse de esto.

```
1
2     def teclado(self, task):
3         dt = globalClock.getDt()
4         self.update_freqs()
5         x = self.camera.getX()
6         y = self.camera.getY()
7         z = self.camera.getZ()
8         h = self.camera.getH()
9         p = self.camera.getP()
10        r = self.camera.getR()
11        self.texto_info_status.setText(self.status_text())
12
13        ndivs=len(self.divs)
14        if ndivs >= 1:
15            self.divs[0]=1+keyMap["d1"]
16        if ndivs >= 2:
17            self.divs[1]= 1+keyMap["d2"]
18        if ndivs >= 3:
19            self.divs[2] = 1+ keyMap["d3"]
20
21        self.update_freqs()
22
23        ntexts=len(self.stimulus_selected_texture_set)
24
25        if ntexts>=1:
26            self.stimulus_selected_texture_set[0] = keyMap["
                  textura_cambio_izquierda"]
27
28        if ntexts>=2:
29            self.stimulus_selected_texture_set[1] = keyMap["textura_cambio_centro
                  "]
30
31        if ntexts>=3:
32            self.stimulus_selected_texture_set[2] = keyMap["
                  textura_cambio_derecha"]
```

Código 3.21: Acciones independientes de pulsación

El código 3.21 muestra parte de estas funciones. Recordemos que se aplica en cada iteración del bucle del juego. En esta parte inicial de la tarea se obtiene la posición del jugador, se actualiza la tasa de frames, se calculan los divisores para determinar la frecuencia de los estímulos y se actualiza el texto informativo de la parte superior de la pantalla.

```
        if keyMap["rotate"]:
            #pos.z -= self.speed * dt
            h -= self.rotdelta
            self.rotacc += self.rotdelta
            if self.rotacc>=self.rottarget:
                keyMap["rotate"]=False
                self.rotacc=0.0

        if keyMap["negrotate"]:
            h += self.rotdelta
            self.rotacc += self.rotdelta
            if self.rotacc>=self.rottarget:
                keyMap["negrotate"]=False
                self.rotacc=0.0

        if keyMap["go"]:
            y = y + self.movdelta*math.cos(h*np.pi/180)
            x = x - self.movdelta*math.sin(h*np.pi/180)
            self.movacc += self.movdelta
            if self.movacc>=self.movtarget:
                keyMap["go"]=False
                self.movacc=0.0
            print(x)
            print(y)

        if keyMap["back"]:
            y = y - self.movdelta*math.cos(h*np.pi/180)
            x = x + self.movdelta*math.sin(h*np.pi/180)
            self.movacc += self.movdelta
            if self.movacc>=self.movtarget:
                keyMap["back"]=False
                self.movacc=0.0
```

Código 3.22: Movimiento usando teclado

El código 3.22 muestra el control sobre los movimientos del usuario por el mapa con el uso del teclado: avanzar o retroceder una cantidad preestablecida y girar a la derecha o izquierda una cantidad.

```
        if keyMap["end_game"]:
            self.desconectar()
            # Cerramos segundo archivo de datos
            self.comprobar.close()
            # Terminamos la aplicaci'on
            return sys.exit()
```

Código 3.23: Fin del juego

En el código 3.23 se permite salir del juego pulsando una tecla, realizando previamente las operaciones de desconexión y cierre de archivos.

```
1
2         if keyMap["reconectar"]:
3             if self.ctr_data_thread.connected:
4                 self.desconectar()
5             self.taskMgr.add(self.conectar,"conectar")
6             updateKeyMap("reconectar",False)
```

Código 3.24: Reconectar al streming

En el código 3.24 se muestra como es posbile reconectar con el streaming de datos. Utilizar esta función hará que se borren los datos guardados en el archivo.

```
1
2         if keyMap["start_recording"]:
3             self.ctr_data_thread.start_storing()
4             self.texto_guardando_datos.show()
5             updateKeyMap("start_recording",False)
6
7         if keyMap["stop_recording"]:
8             try:
9                 time.sleep(0.1)
10                self.texto_guardando_datos.hide()
11                self.ctr_data_thread.end_storing()
12                corrs, max, qval = procesamiento_datos.procesamiento("datos.csv",
13                self.freqs,
14                targets_por_sesion=len(self.divs),
15                acondiciona_flag=False,
16                channels=self.channels)
17                self.comprobar.write(str(max))
18                self.comprobar.write('\n')
19                self.comprobar.write(str(qval))
20            except Exception as e:
21                print(e)
22                print("Error en stop_recording action: tratamos de cerrar")
23                self.desconectar()
24                self.comprobar.close()
25                return sys.exit()
26
27            self.comprobar.write('\n')
28            updateKeyMap("stop_recording",False)
```

Código 3.25: Abrir y cerrar archivo

El código3.25 sirve para ordenar empezar a escribir en el archivo de datos (solo válido después de una reconexión) y para dejar de grabar datos y cerrar el archivo de datos y desconectarse del streaming de datos.

```
1
2
3           if keyMap["attention"]:
4               updateKeyMap("modo_automatico",False)
5               self.box[0].setTexture(self.colors[0], 0)
6               self.box[1].setTexture(self.colors[0], 0)
7               self.box[2].setTexture(self.colors[0], 0)
8               self.red_color = False
9               self.ctr_data_thread.att()
10              self.texto_atendiendo_estimulo.show()
11              updateKeyMap("attention",False)
```

Código 3.26: Indicar atención

En el código 3.26 se le indica al hilo de recogida de datos que se está prestando atención a un estímulo, por lo que se empezarán a guardar datos en el buffer y en el archivo de datos se guardarán los datos reales (en lugar de “nan”).

```
1           if keyMap["not_attention"]:
2               self.ctr_data_thread.noatt()
3               self.texto_atendiendo_estimulo.hide()
4               time.sleep(0.1)
5               info = self.ctr_data_thread.data_buffer()
6               try:
7                   corrs, max, qval = procesamiento_datos.procesamiento(info,
8                       self.freqs,
9                       targets_por_sesion=1,
10                      acondiciona_flag=False,
11                      channels=self.channels)
12              except Exception as e:
13                  print(e)
14                  print("Error en la deteccion manual")
15                  self.comprobar.write("Error en deteccion\n")
16              self.texto_info_detected_target.setText('Ultimo target detectado {}
                    Qval {}'.format(max,qval))
17              info="detecta {}".format(max)
18              if corrs is None:
19                  corrs=["NA","NA","NA"]
20              else:
21                  if max == 2:
22                      self.action_confirmation = max
23                      self.box[2].setTexture(self.colors[1], 0)
24                  else:
25                      self.red_color = True
26                      if self.last_detected_action == max:
27                          self.last_detected_action += 3
28                      else:
29                          self.last_detected_action = max
30                      self.action_confirmation = None
```

Código 3.27: Acciones a realizar cuando se ha dejado de prestar atención al estímulo (Parte 1)

```
                match self.last_detected_action:

                    case 0:
                        self.accion = "Avanzar"
                        if self.red_color:
                            self.box[0].setTexture(self.colors[1], 0)
                    case 1:
                        self.accion = "Girar_derecha"
                        if self.red_color:
                            self.box[1].setTexture(self.colors[1], 0)
                    case None:
                        self.accion = "Confirmar"
                        if self.red_color:
                            self.box[2].setTexture(self.colors[1], 0)
                    case 3:
                        self.accion = "Retroceder"
                        if self.red_color:
                            self.box[0].setTexture(self.colors[1], 0)
                    case 4:
                        self.accion = "Girar_izquierda"
                        if self.red_color:
                            self.box[1].setTexture(self.colors[1], 0)

                self.texto_accion_guardada.setText(self.accion)

            for corr in corrs:
                info += " {}".format(corr)
            info += " {}".format(qval)
            info += "\n"
            self.comprobar.write(info)
            updateKeyMap("not_attention",False)
```

Código 3.28: Acciones a realizar cuando se ha dejado de prestar atención al estímulo (Parte 2)

El código 3.27 informa al hilo de recogida de datos que se ha dejado de prestar atención. Además da la orden de realizar la correlación con los datos dentro del buffer y posteriormente recibe la información del estímulo resultado que se ha obtenido. Esa información será utilizada por la tarea “head_movement” para realizar la ación dentro del laberinto.

En resumen, el uso no automático del prototipo es el siguiente: se utiliza el código 3.26 para indicar atención y el usuario debe estar centrado ya en el estímulo. Cuando haya pasado un tiempo se debe pulsar la tecla correspondiente al código 3.27 para hacer la correlación y moverse dentro del laberinto.

```
1
2
3           if keyMap["modo_automatico"]:
4               if self.automatic_time == 0:
5                   self.ctr_data_thread.att()
6                   self.box[0].setTexture(self.colors[0], 0)
7                   self.box[1].setTexture(self.colors[0], 0)
8                   self.box[2].setTexture(self.colors[0], 0)
9                   self.red_color = False
10                  self.texto_atendiendo_estimulo.show()
11              self.automatic_time += dt
12
13              if (self.automatic_time > self.automatic_time_attention) and not self
                    .automatic_part_detection:
14                  keyMap["not_attention"] = True
15                  self.automatic_part_detection = True
16
17              elif self.automatic_time > self.automatic_time_total:
18                  self.automatic_time = 0
19                  self.automatic_part_detection = False
```

Código 3.29: Iniciar modo automático

El código 3.29 inicia el modo automático de funcionamiento. Una vez se entre en este modo, el usuario atenderá a un estímulo, tras un tiempo predefinido (5 segundos en la versión actual), sin necesidad de pulsar ninguna otra tecla, se le indicará al usuario que puede dejar de atender mediante un cambio en la caja que contiene al estímulo. Después de eso se realizará el análisis de correlación mostrando el resultado con un cambio en la zona de la pantalla con el target detectado, además se esperan 3 segundos para que comience el nuevo proceso de atención a otro estímulo y la recogida de datos. Los tiempos de atención y cambio de estímulo se deben definir en constructor de la clase MyApp.

3.10. El mapa del juego

El primer paso para construir el mapa del juego es diseñar un modelo 3D haciendo uso de Blender y posteriormente exportarlo a un formato propio de Panda3d mediante el uso de la herramienta de consola “blend2bam”. En la figura 3.5 se muestra el diseño de mapa empleado. Una vez exportado el modelo, el constructor de nuestra aplicación se encargará de cargarlo y realizar todas las operaciones pertinentes con él.

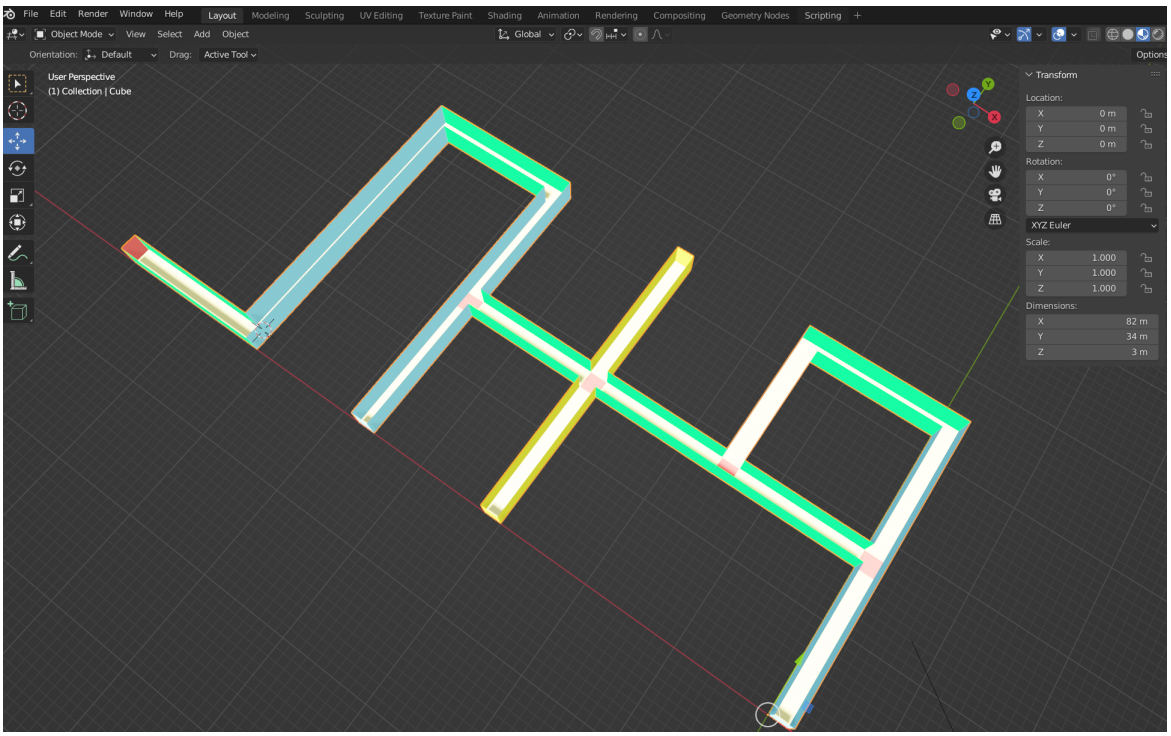

Figura 3.5: Mapa utilizado en el proyecto

```
1         self.disableMouse()  # Si no se desactiva el raton la camara volvera a su
              lugar inicial tras el movimiento
2         # Let's load the building model and save it as self.building.
3         self.building = self.loader.loadModel("prueba.bam")
4         self.building.reparentTo(self.render)
5         self.building.setScale(1, 1, 1)
6         self.camera.setPos(1, 1, 1)
7         self.camLens.setNear(0.1) # Clipping Near Plane de la camara reducido
              para evitar ver a trav'es de las paredes
8         self.camera.setHpr(0, 0, 0) # Orientaci'on de la c'amara
```

Código 3.30: Carga del modelos 3D

En el código 3.30 se carga el modelo 3D, se renderiza y se le aplica un escalado. También se indica la posición y orientación de la cámara (usuario) dentro del espacio 3D.

```
1
2         ambientLight = AmbientLight('ambientLight') # Iluminaci'on ambiente para
              zonas no ilumiandas por el resto
3         ambientLight.setColor((0.2, 0.2, 0.2, 1))
4         ambientLightNP = self.render.attachNewNode(ambientLight)
5         self.render.setLight(ambientLightNP)
6         dlight = DirectionalLight('my dlight') # Ilumianci'on direccional
7         dlight.setColor((0.8,0.8,0.8,1.0))
8         dlnp = self.render.attachNewNode(dlight)
9         dlnp.setHpr(45,-45,0)
10        self.render.setLight(dlnp)
```

Código 3.31: Iluminación

Con el código 3.31 se maneja la iluminación del mapa para poder visualizar los mate-

riales del laberinto y mejorar la sensación de movimiento en el espacio.

```
1
2           self.cTrav = CollisionTraverser()
3           self.pusher = CollisionHandlerPusher() # Manejador que evita que dos
                objetos solidos se solapen
4
5           # Colision en la camara
6           colliderNode = CollisionNode("player")
7           # Add a collision-sphere centred on (0, 0, 0), and with a radius of 0.3
8           colliderNode.addSolid(CollisionBox(Point3(-0.75,-0.75,-0.75), Point3
                (0.75,0.75,0.75)))
9           collider = self.camera.attachNewNode(colliderNode)
10          self.pusher.addCollider(collider, self.camera)
11          self.cTrav.addCollider(collider, self.pusher)
```

Código 3.32: Colisiones del jugador

El código 3.32 se encarga de aplicar los sólidos de colisión a la cámara para evitar que atraviese las paredes del mapa (cuando a estas se le apliquen sólidos de colisión).

```
1         # Extraido de https://discourse.panda3d.org/t/collision-mesh-from-loaded-
              model-for-built-in-collision-system/27102 (ver referencias al final de
              la memoria)
2         # Creamos una copia de nuestro laberinto
3         building_copy=self.building.copy_to(self.render)
4         # No se va a renderizar, solo se usa para generar solidos poligonales de
              colisiones
5         building_copy.detach_node()
6         # Aplicamos previamente las transformaciones a los vertices para
              optimizar
7         building_copy.flatten_light()
8         # Vamos a crear un nodo que sea la ra'iz de los solidos de colisi'on
9         collision_root=NodePath("collision_root")
10        collision_root.reparent_to(self.building)
11
12        # Buscamdos todos los nodos de tipo GeomNode en nuestra geometria
13        for model in building_copy.find_all_matches("**/+GeomNode"):
14            model_node=model.node() # Obtenemos un puntero al nodo
15            collision_node=CollisionNode(model_node.name)
16            collision_mesh=collision_root.attach_new_node(collision_node)
17            # Obtenemos la lista de geoms del nodo
18            for geom in model_node.modify_geoms():
19                # Descomponemos la geometria en sus primitivas sin
                      transnformarlas
20                geom.decompose_in_place()
21                # Obtenemos los datos de los vertices
22                vertex_data = geom.modify_vertex_data()
23                vertex_data.format = GeomVertexFormat.get_v3() # Se establece un
                      formato para los vertices
24                #Creamos una vista de memoria para el array de vertices formato
                      flotante
25                view = memoryview(vertex_data.arrays[0]).cast("B").cast("f")
26                #Obtenemos las primitivas en forma de lista de indices de
                      vertices
27                index_list = geom.primitives[0].get_vertex_list()
28                index_count = len(index_list)
29                #Asume primitivas que son triangulos
30                for indices in (index_list[i:i+3] for i in range(0, index_count,
                      3)):
31                    #Creamos una lista de Point3, son los vertices de la
                          primitiva
32                    points = [Point3(*view[index*3:index*3+3]) for index in
                          indices]
33                    #Creamos un solido de colision
34                    coll_poly = CollisionPolygon(*points)
35                    # Lo agregamos al nodo de colision
36                    collision_node.add_solid(coll_poly)
```

Código 3.33: Colisiones de las paredes

En el código 3.33 vemos una técnica extraída de (Epihaius et al. (2021)) para aplicar los

sólidos de colisión a los polígonos que conforman las paredes del laberinto. Esto, sumado al sólido de colisión implementado en la cámara impiden que el usuario atraviese las paredes del laberinto. Esta técnica no es infalible y pueden encontrarse algunos “bugs” en las esquinas del mapa.

Capítulo 4

Evaluación del interfaz cerebro-computador

Una vez completado el desarrollo del prototipo nos disponemos a realizar una serie de pruebas para probar la eficacia del método. Se han propuesto dos tipos de recorridos por el laberinto: utilizando el teclado para indicar la atención del usuario al estímulo y utilizando el modo automático del juego.

4.1. Análisis previos

Para asegurarnos que las señales capturadas eran correctas y reunían las características deseadas se crearon scripts para Matlab donde analizar el archivo de datos en crudo almacenado por la aplicación. Para esta prueba el usuario miraba durante aproximadamente 5 segundos un estímulo y después pasaba al de la derecha, hasta completar la serie.

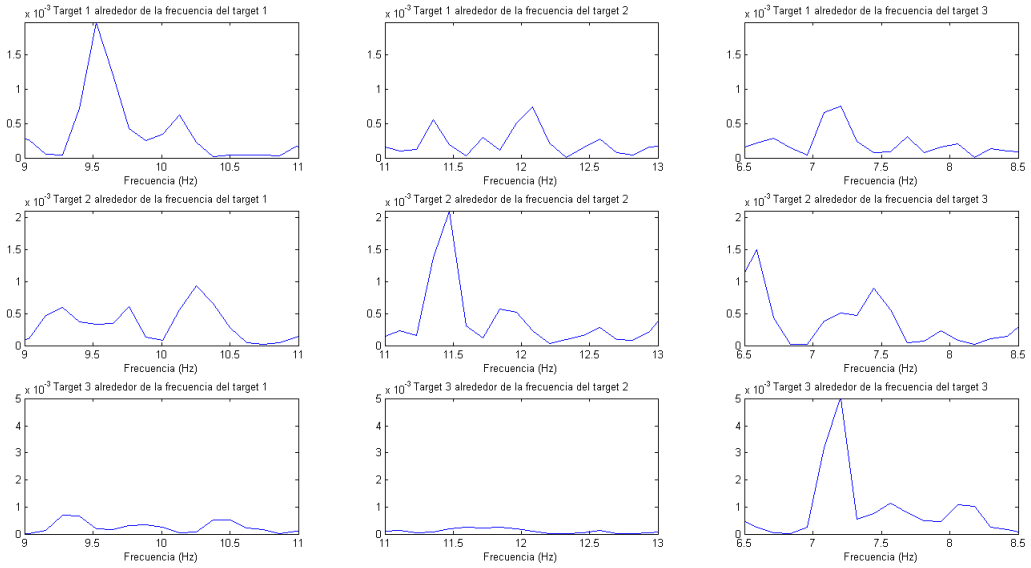

Figura 4.1: Análisis del espectro de frecuencias, como validación previa del funcionamiento correcto de la estimulación y la adqusición de datos (prueba 1)

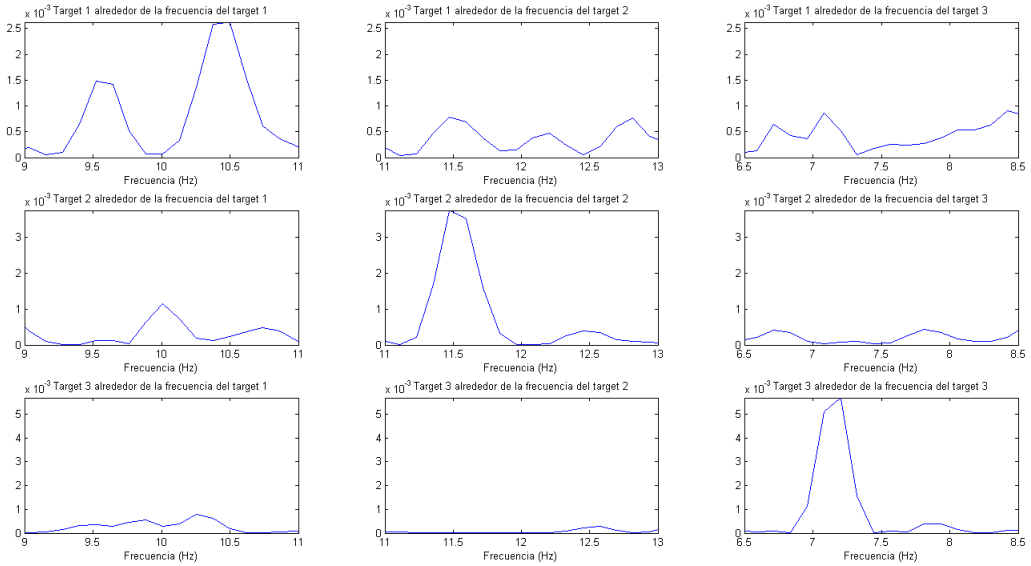

Figura 4.2: Análisis del espectro de frecuencias, como validación previa del funcionamiento correcto de la estimulación y la adquisición de datos (prueba 2)

Las figuras 4.1 y 4.2 muestran los espectros de las pruebas previas. En cada fila se encuentra el espectro cuando se observa un determinado target. En cada columna se aprecian los espectros en las zonas que se encuentran alrededor de las frecuencias nominales de nuestros estímulos (10, 12 y 7.5 Hz). Se pude apreciar que los picos en cada caso se encuentran en la zona correspondiente al estímulo al que se ha atendido (se encuentran desplazados debido a que la tasa de fotogramas de la aplicación no es exactamente 60 FPS)

Los resultados de estas pruebas son lo bastante positivos como para continuar haciendo experimentos con esta configuración.

4.2. Pruebas de funcionamiento de la aplicación

A continuación se detallan diferentes pruebas realizadas en la aplicación con la finalidad de obtener conclusiones sobre la robustez, simplicidad y operatividad del interfaz BCI propuesto. Las condiciones de estas pruebas son las siguientes.

En todas las pruebas el estímulo izquierdo es de tipo simple (imagen que aparece y desaparece), y la imagen es un patrón de tipo “damero”. El estímulo central es de tipo doble formado con dos dameros opuestos (en lugar de hacer desaparecer la imagen, se invierten los cuadrados negros y blancos). En el tercer estímulo, situado más a la derecha, la tipología varía según las pruebas. Las frecuencias ideales de los estímulos son de 10, 12 y 7.5 Hz de izquierda a derecha. En la figura 4.3 se muestra un momento del juego en el inicio del recorrido del laberinto con los tres estímulos.

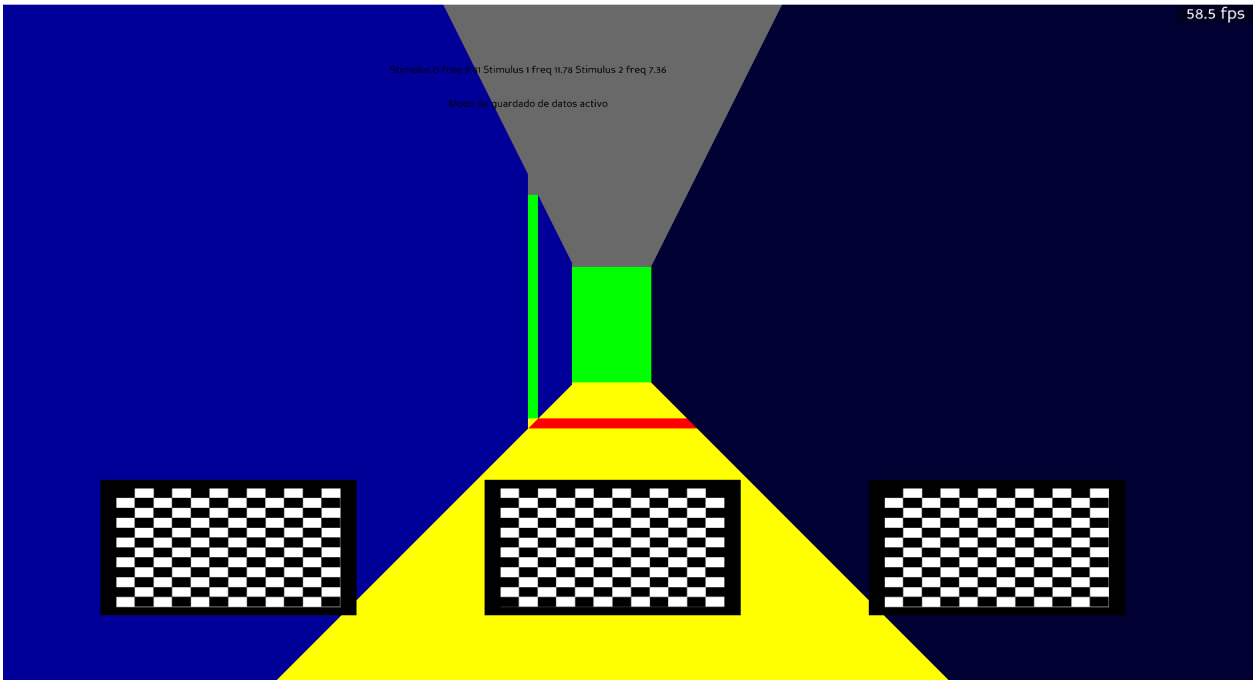

Figura 4.3: Vista prueba del juego

Además, en la detección se utilizan tres armónicos y los 16 canales del EEG facilitados por el dispositivo. La frecuencia de muestreo es 125 Hz y se aplicaron los siguientes filtros en la aplicación Open BCI GUI:

- Filtros “Notch” en 50 Hz y 60 Hz para eliminar interferencias.
- Filtro pasabanda Butterworth con banda de paso entre 5 Hz y 50 Hz.

El ordenador utilizado para las pruebas tiene un procesador Intel Core I5-3330, $3{,}0$ GHz con una memoria de 16 GB, tarjeta gráfica NVIDIA GeForce GTX 1060 3GB, sistema operativo Windows 10 Profesional (10.0.19044) y monitor Acer de 23 pulgadas (75 Hz). Hoy en día sería un ordenador en la gama de bajas prestaciones.

Todas las pruebas se realizaron con el mismo sujeto. Este sujeto es uno de los tutores del proyecto y por lo tanto ya estaba acostumbrado al sistema y entrenado con el interfaz de usuario.

Las pruebas se realizaron sobre un escenario consistente en un laberinto con pasillos de 2 unidades de ancho, con giros a derecha e izquierda de 90 grados. El sistema se configuró para realizar avances con un paso de 1 unidad y giros con un paso 30 grados.

4.2.1. Pruebas con modo manual

Para este tipo de de pruebas el usuario indicará al sistema que está prestando atención con la tecla ”T” e indicará que deja de prestar atención con la tecla ”Y”. Se han realizado dos pruebas de este tipo. En la primera prueba, el estímulo de confirmación es de tipo simple con una imagen que aparece y se oculta, siendo esta imagen un cuadrado blanco. A continuación se detallan los resultados de esta prueba.

Debido a que la frecuencia del juego no es completamente estable durante todo el transcurso del juego las frecuencias de los estímulos varían, y también lo hacen las frecuencias de las señales de referencia utilizadas para hacer el análisis de correlación canónica.

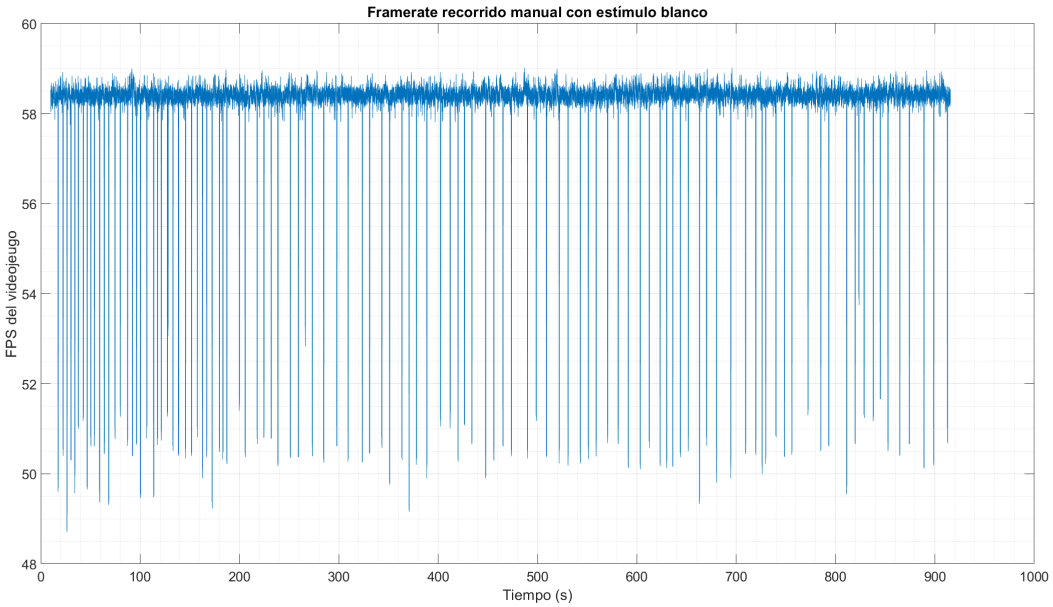

Figura 4.4: Evolución del framerate en la primera prueba manual

Como se observa en 4.4 los FPS del juego tienden a oscilar entre los 59 como máximo y llegan a bajar hasta los 57.8. Sin embargo, en la imagen se observan caídas puntales de hasta los 49 FPS esto se debe principalmente a los eventos tales como indicar la no atención del usuario y calcular la correlación.

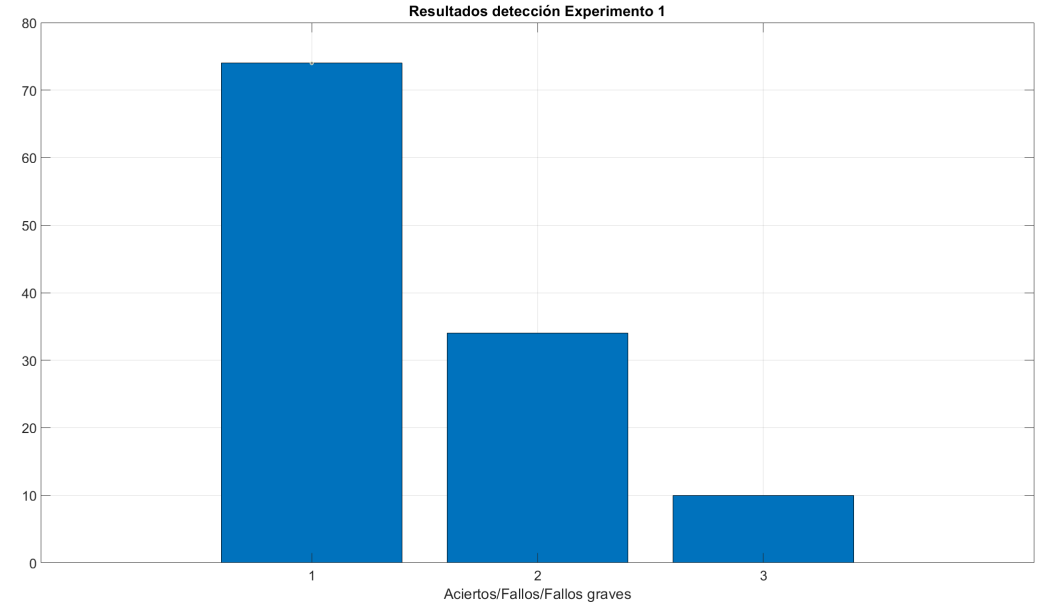

Figura 4.5: Resultados experimento manual 1

En 4.5 podemos ver la tasa de fallos de este recorrido. De las 108 detecciones totales hubo error en 34 de ellas ($31{,}48\,\%$). De esos 34 errores, 10 de ellos fueron errores peligrosos. Un error peligroso es aquel en el que el usuario quiere elegir una nueva acción pero en su lugar se confirma la realización del movimiento guardado, es decir, se produce un movimiento no deseado.

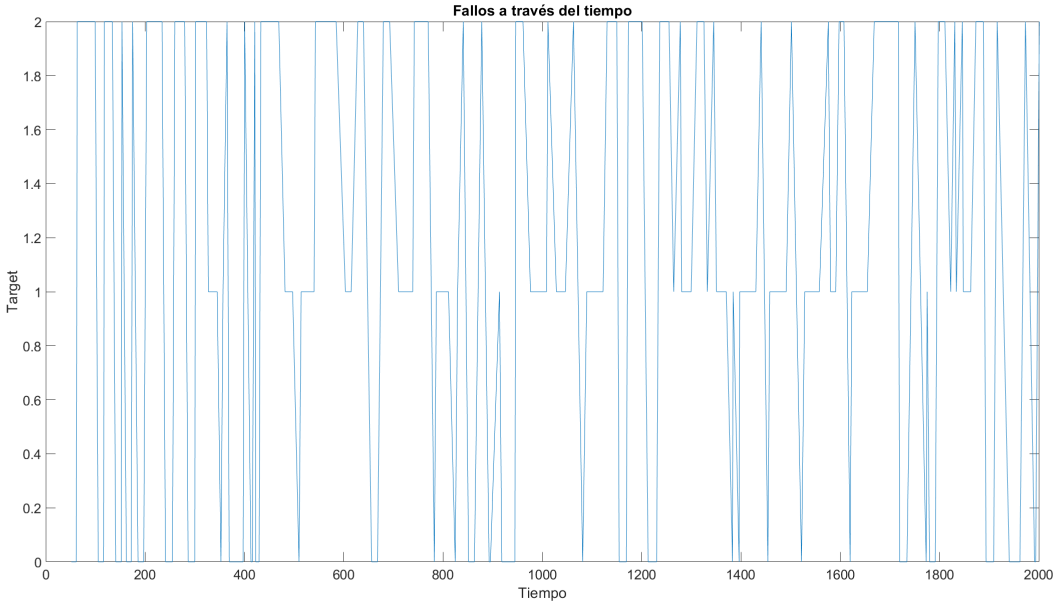

Figura 4.6: Fallos experimento 1 frente a tiempo

La figura 4.6 nos muestra los fallos cometidos en el experimento a través del tiempo. En caso de encontrar una línea horizontal es que la correlación ha dado el resultado deseado, los picos en la gráfica muestran los momentos donde se han cometido los fallos.Cuanto más larga sea la línea horizontal más veces se ha repetido el comando de forma continuada. Como podemos ver la mayoría de estos fallos se concentran al final de la gráfica. En este experimento no se consigue completar el laberinto.

Pasamos ahora con el segundo de los recorridos manuales, en este caso el tercer estímulo es un damero en lugar de un cuadrado blanco, donde se alternan los cuadrados blancos y negros.

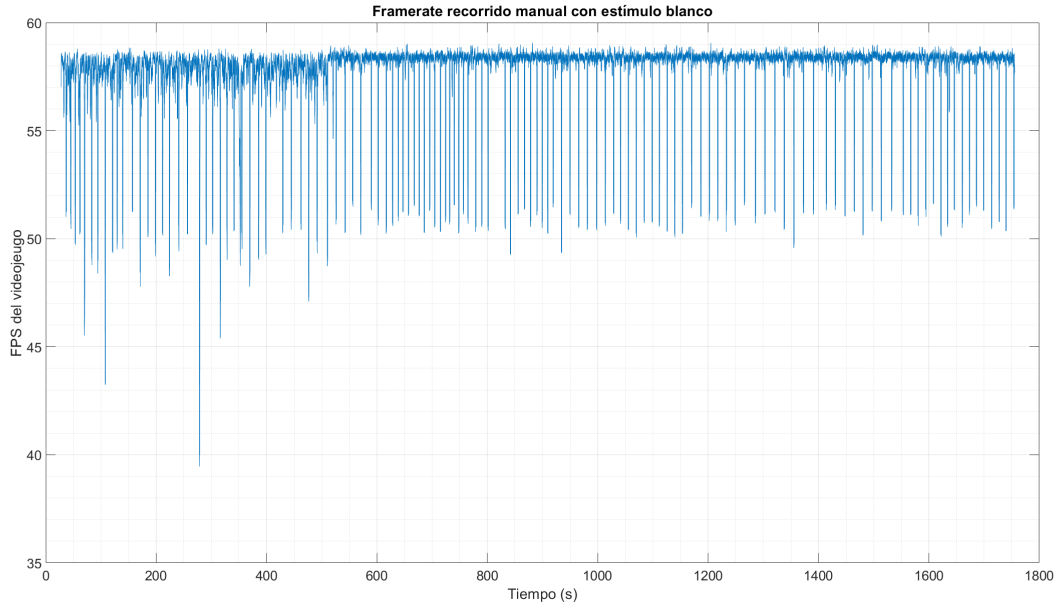

Figura 4.7: Evolución del framerate en la segunda prueba manual

En la figura 4.7 se muestra la evolución del framerate del juego en el segundo experimento. Salvando las caídas en los primeros minutos del experimento, los FPS se mantienen similares a lo observado en el primer caso.

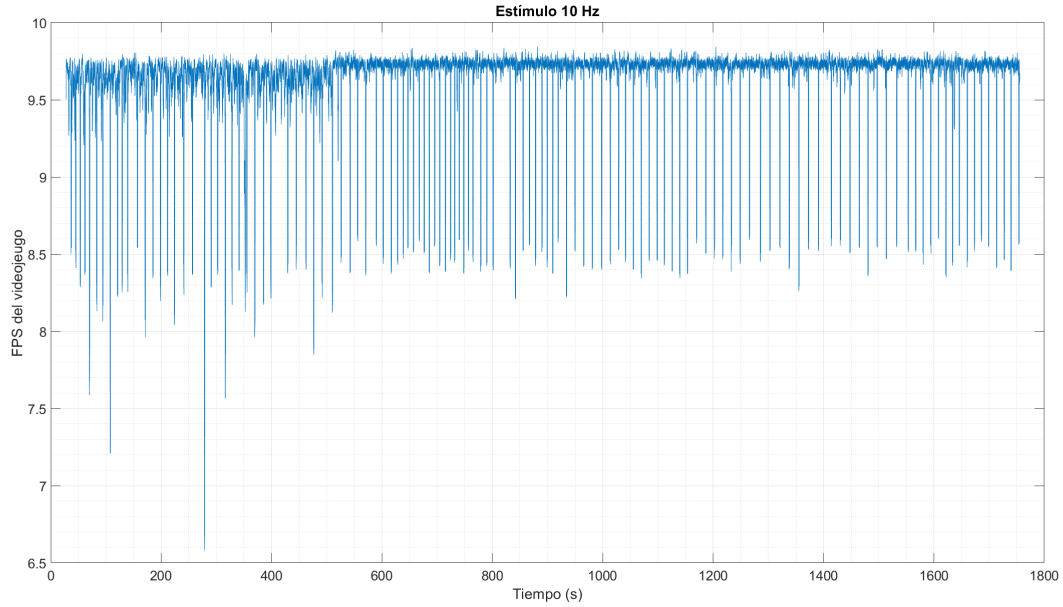

Figura 4.8: Evolución del estímulo de 10 Hz en la segunda prueba

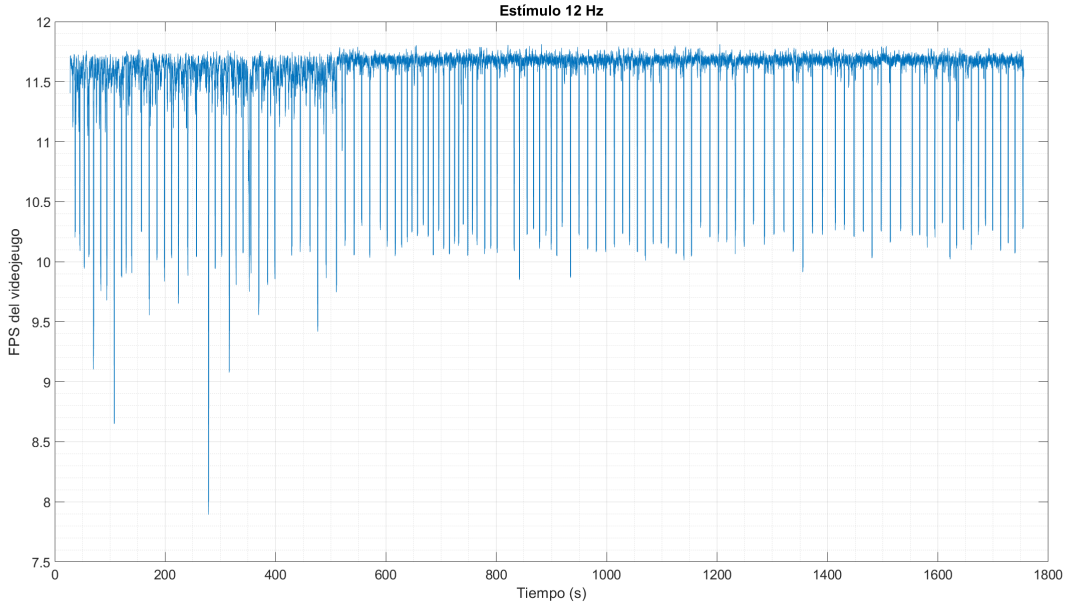

Figura 4.9: Evolución del estímulo de 12 Hz en la segunda prueba

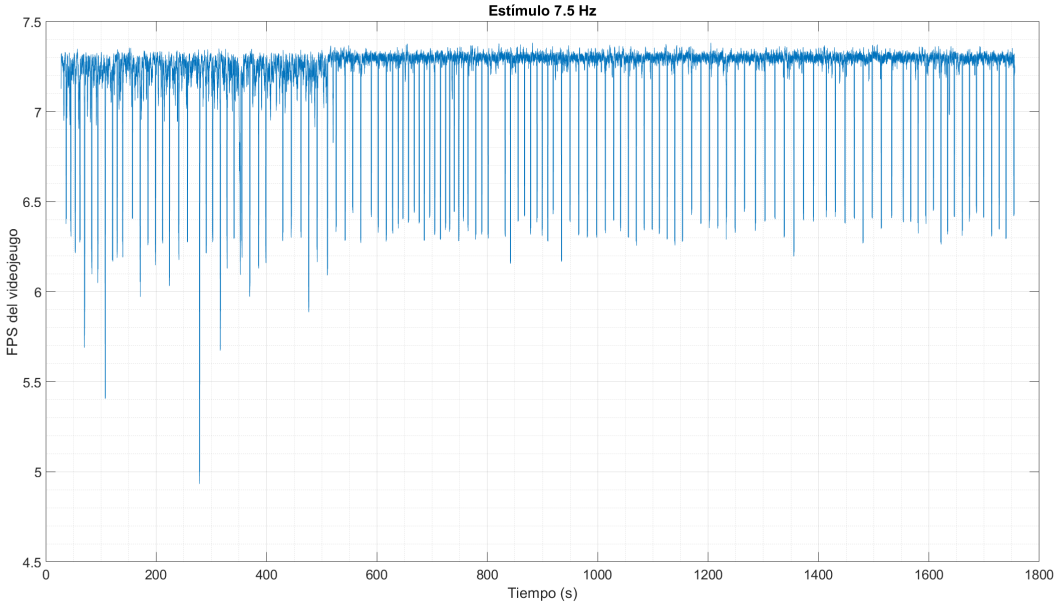

Figura 4.10: Evolución del estímulo de 7.5 Hz en la segunda prueba

Como puede apreciarse en las figuras 4.8, 4.9 y 4.10 el comportamiento de las frecuencias de los estímulos es un reflejo del comportamiento del framerate del juego. Esta característica será común a todos los experimentos que se realicen. Es importante señalar que una caída excesiva en la tasa de frames puede ocasionar que la frecuencia de los estímulos baje demasiado y no se obtenga un potencial de estado estacionario en el cerebro, o este no se detecte.

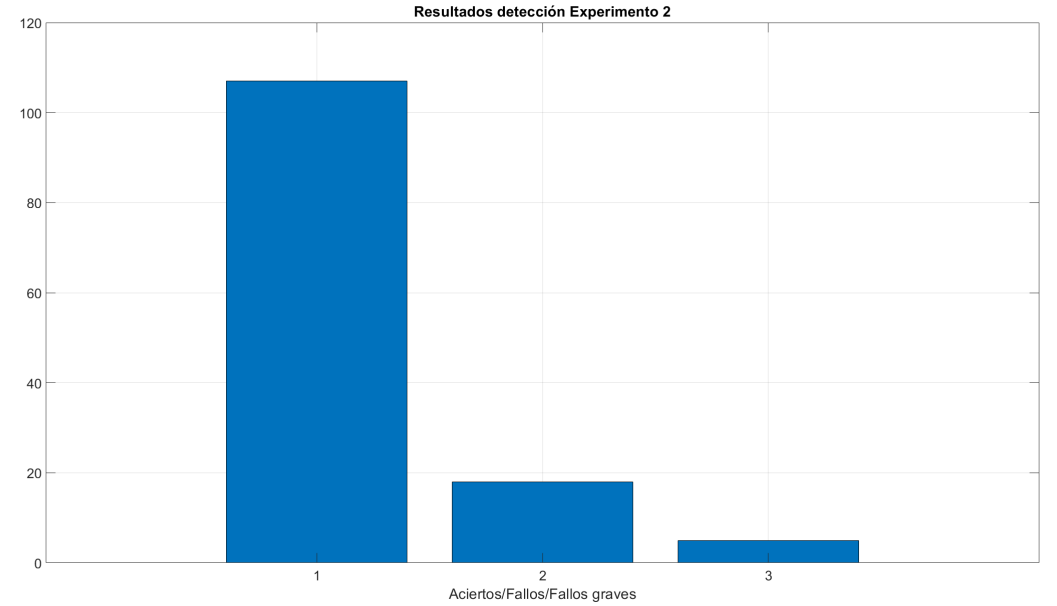

Figura 4.11: Resultados experimento manual 2

Como puede observarse en 4.11 de las 125 iteraciones se han acertado 107 de ellas ($85{,}6\,\%$), una mejora sustancial respecto al $68{,}52\,\%$ de aciertos del experimento realizado con un cuadrado blanco en lugar de un damero. Los errores peligrosos mantienen el

Trayecto	Distancia	Giros izquierda	Giros Derecha
1	82	2	0
2	147	3	2
3	66	2	0
4	66	2	0

Tabla 4.1: Características de los trayectos analizados en el modo automático. Las unidades de distancia pueden considerarse metros, asumiendo que los pasillos tienen 2 metros de ancho y el móvil se circunscribe a un cubo de $1{,}5$ metros por $1{,}5$ metros.

porcentaje con respecto al experimento anterior (en este caso se produjeron 5).

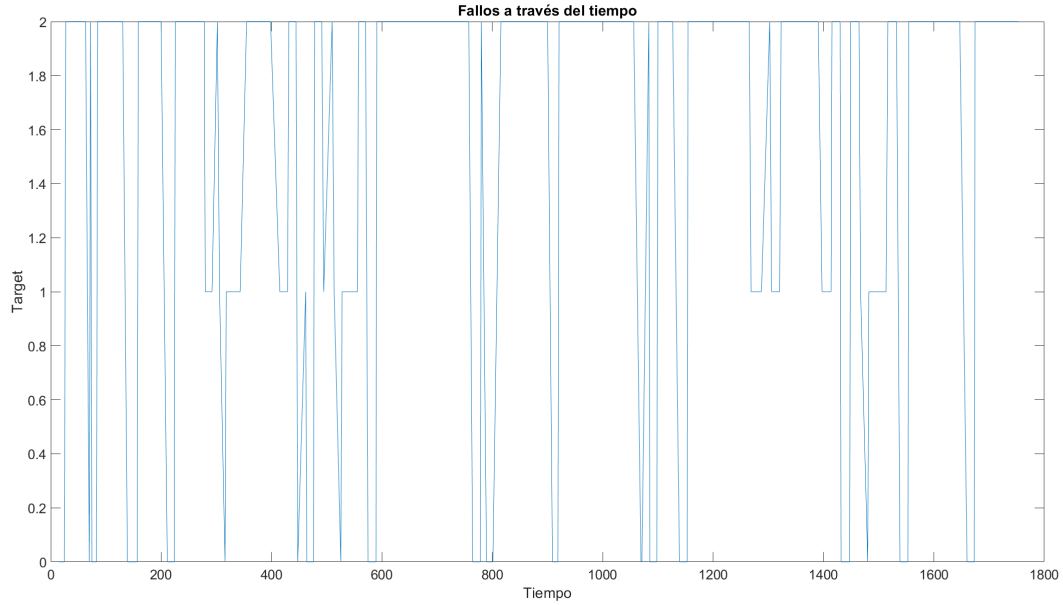

Figura 4.12: Fallos experimento 2 frente a tiempo

En el caso del segundo experimento vemos una 4.6 más ”limpia”, los fallos se distribuyen más por toda la duración del experimento.

4.2.2. Pruebas con modo automático

Las pruebas del modo automático tienen las mismas características que la segunda de las pruebas en modo manual (frecuencias y tipos de estímulos) con la salvedad que ahora no se utilizarán las teclas para indicar la atención y no atención del usuario. En su lugar, al inicio de la prueba se pulsará la tecla ”k” que activa el modo automático. En este modo se guardarán datos del EEG durante cinco segundo en el buffer cíclico en memoria, pasado ese tiempo un indicador le informará al usuario que los datos han sido tomados y este tendrá 3 segundos para centrar su atención en otro estímulo. Esto se repetirá hasta que se complete la trayectoria.

Se han tratado de seguir 4 trayectorias distintas por el laberinto. Como en el caso de las pruebas manuales se analizará el framerate del juego durante la partida, así como la tasa de fallo obtenida. Las características de los trayectos se muestran en la tabla 4.1

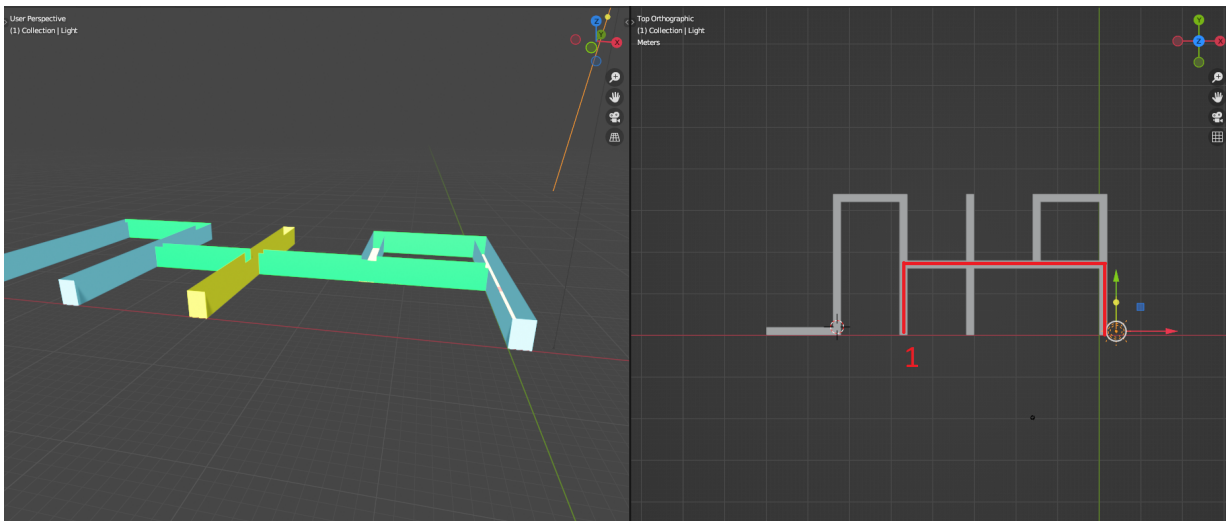

Figura 4.13: Primera trayectoria en modo automático

En la figura 4.13 se muestra el primer trayecto que se ha seguido con el modo automático. El interés de este trayecto radica en que hay dos giros a la izquierda, que es el movimiento que más problemas de maniobrabilidad genera porque:

- El ancho del pasillo respecto ala caja de colisión utilizada par la cámara es bastante justo y esto complica de por si el giro, como ocurriría en una situación real.
- En esta versión del videojuego no se puede girar el punto de vista independientemente de la dirección de avance, lo que complica situárse espacialmente y calcular el giro.
- Los avances y giros son de tipo fijo, lo que obliga a realizar varias combinaciones de maniobras.
- En los giros a la izquierda es preciso atender dos veces consecutivas al estímulo central si se viene de una situación de “avances”.

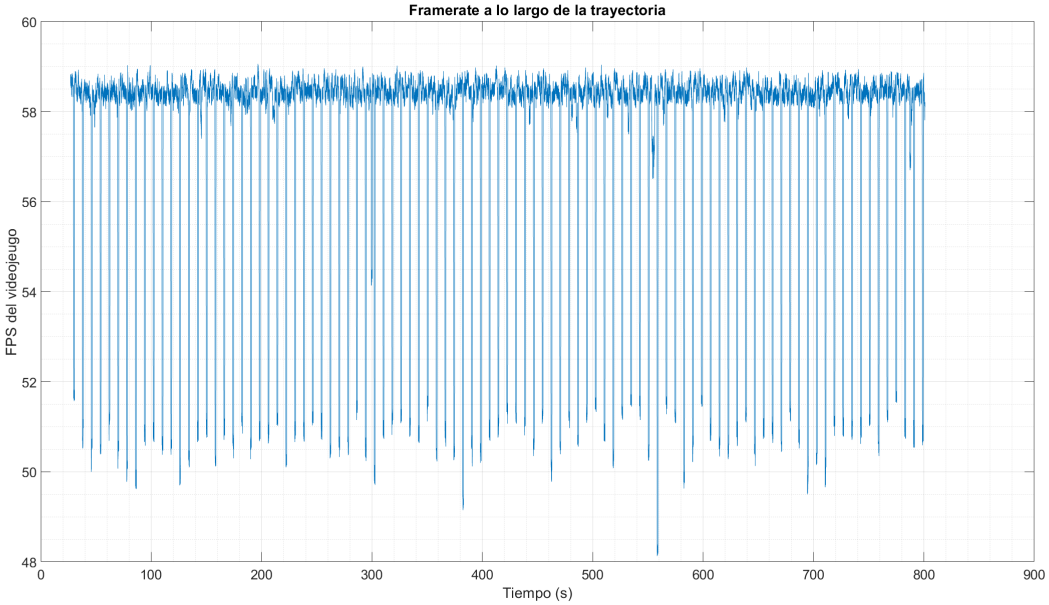

Figura 4.14: Framerate durante la primera prueba automática

La figura 4.14 muestra el framerate durante esta primera trayectoria. El único cambio

a destacar con respecto a los casos anteriores es que la caída de fotogramas debido a los eventos es equidistante en este caso. Esto se debe a las restricción de 5 segundos para la atención del usuario. Es de esperar que, como en los casos anteriores, las frecuencias de los estímulos se comporten de forma similar al framerate de la aplicación.

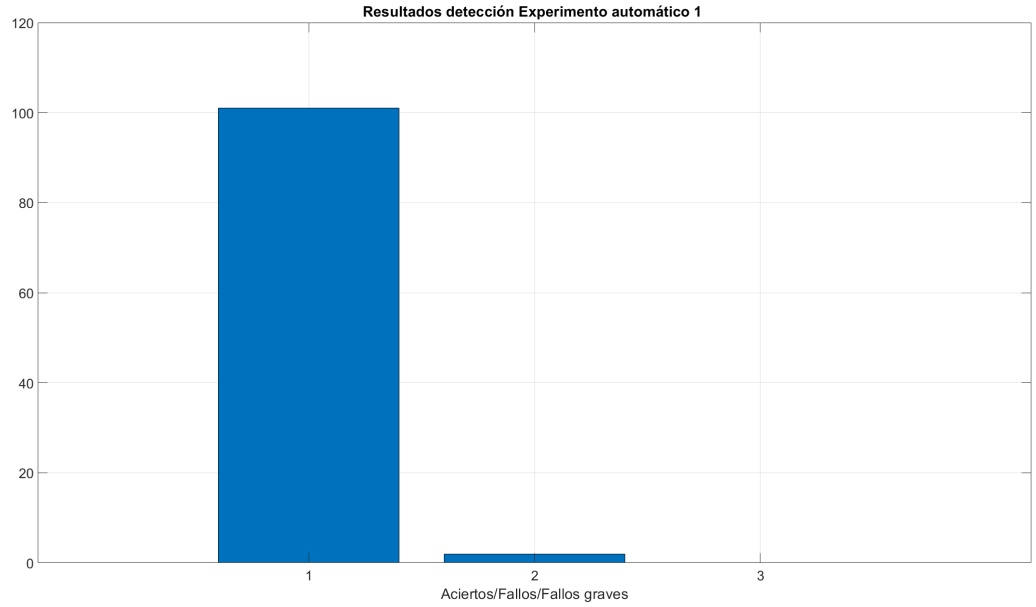

Figura 4.15: Fallos durante el primer trayecto automático

En la figura 4.15 podemos observar los resultados de este trayecto. De las 103 órdenes que introdujo el usuario 101 de ellas fueron discernidas correctamente por el método ($98\,\%$ de acierto). En este caso ninguno de los fallos ocurridos genera peligro.Este ha sido el experimento en el que menos fallos se han cometido.

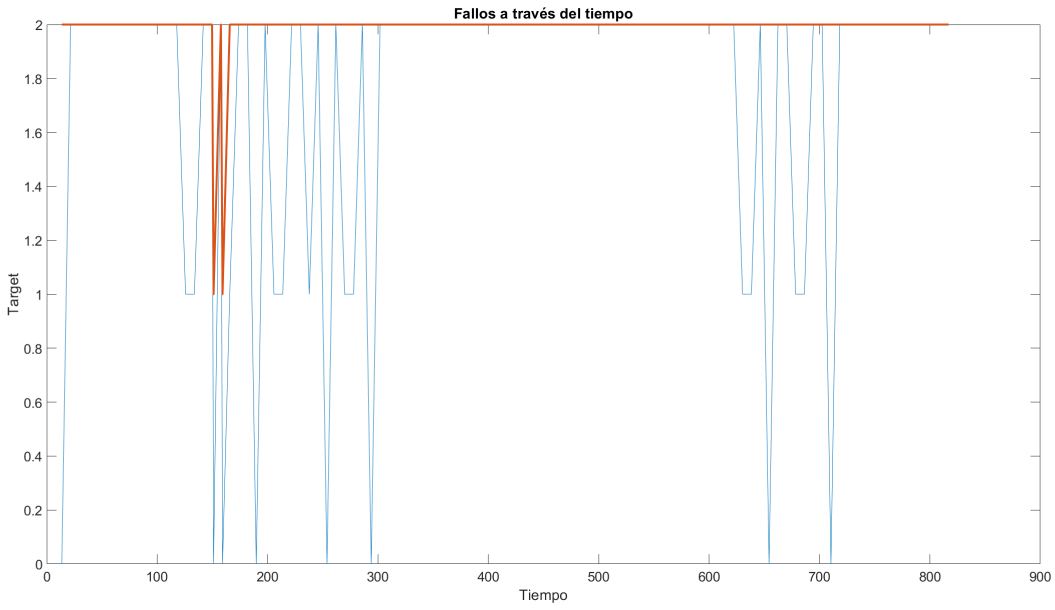

Figura 4.16: Fallos en el tiempo del primer trayecto automático

En la figura 4.16 se muestra la lista de órdenes que se le han dado al juego. En este

caso la línea azul indica los comandos que han sido aplicados, mientras que si la línea naranja baja a ”1” se indica que el comando detectado en ese momento no fue el deseado por el usuario. Para completar este circuito hicieron falta alrededor de 800 segundos (13 minutos y 20 segundos)

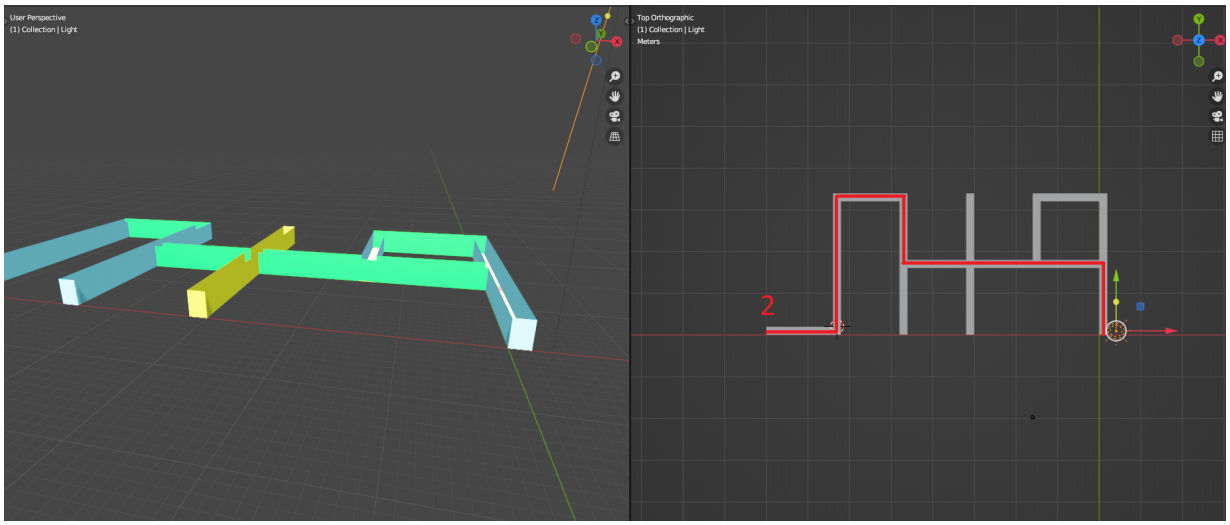

Figura 4.17: Segunda trayectoria en modo automático

La figura 4.17 se corresponde con el segundo trayecto seguido, En esta prueba se completa el circuito, es el recorrido más largo.

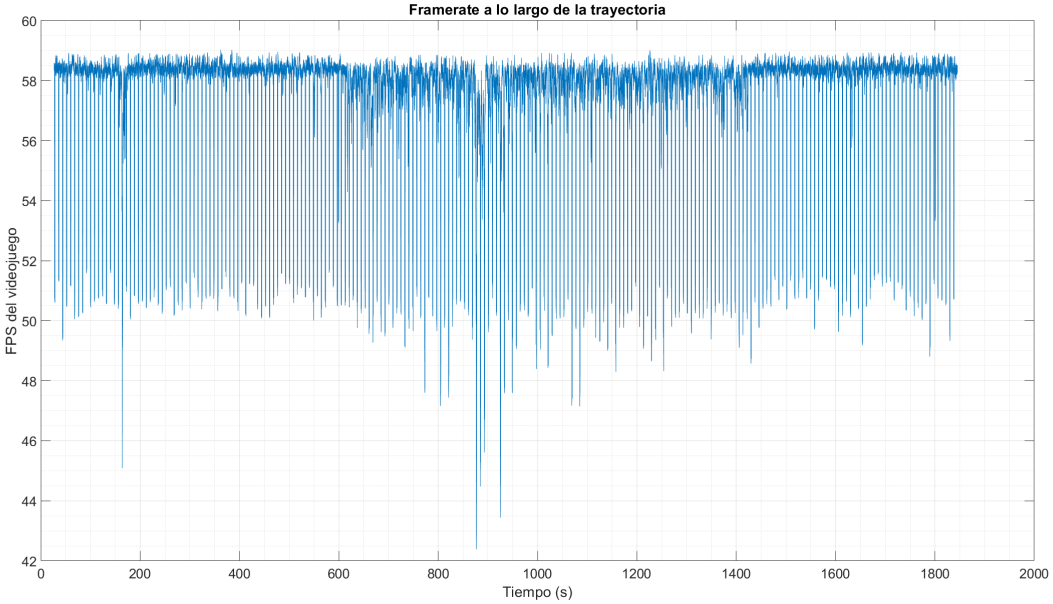

Figura 4.18: Tasa de frames durante la segunda prueba automática

Al igual que en el caso anterior, la figura 4.18 muestra los FPS de la aplicación durante todo el recorrido. Es interesante destacar que este ha sido el recorrido más largo del modo automático y que ha habido durante el recorrido una zona donde la media de los FPS se ha reducido, eso podría deberse a que el sistema operativo lanzó una tarea más prioritaria.

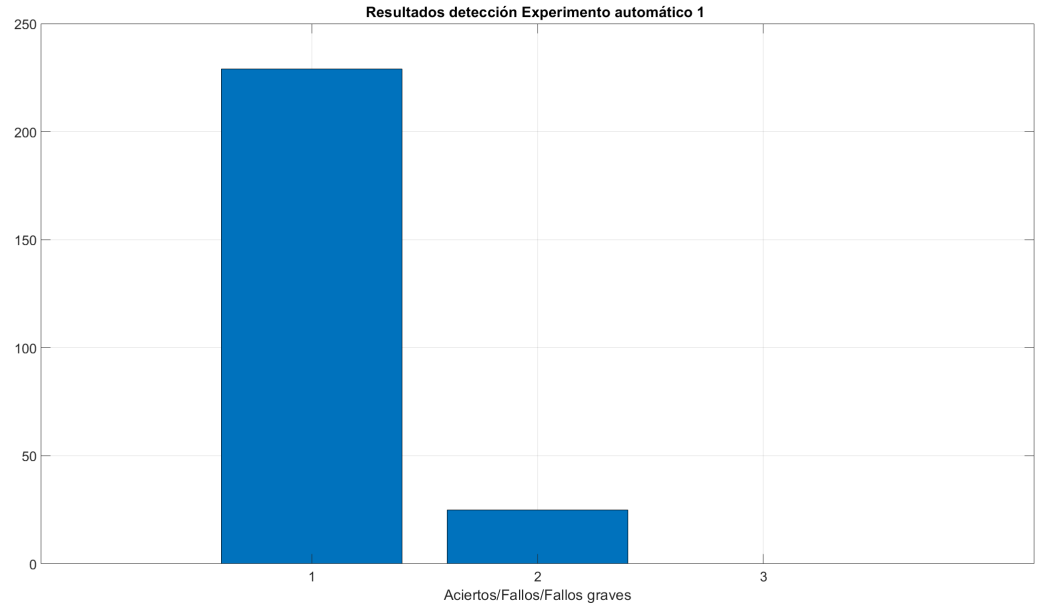

Figura 4.19: Fallos en el tiempo del segundo trayecto automático

En el caso del trayecto más largo se ha cometido el mayor número de fallos de todos los casos automáticos tal y como se aprecia en la Figura 4.19. De las 254 acciones ordenadas, 25 de ellas han resultado en fallo (un $90\,\%$ de acierto), aunque ninguno de ellos acaba siendo uno peligroso.

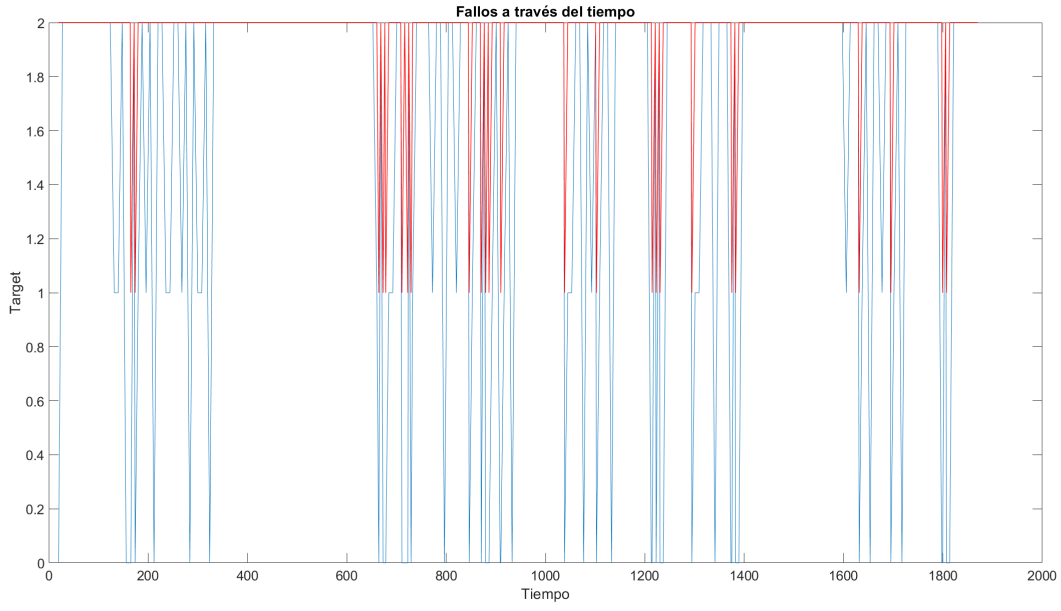

Figura 4.20: Fallos en el tiempo del segundo trayecto automático

En 4.20 observamos los fallos cometidos a lo largo del tiempo. Cabe resaltar que la mayoría de estos fallos se cometen durante la zona de caída de fotogramas. Esta caída resulta en una reducción de la frecuencia de todos los estímulos. La cantidad de fallos puede deberse a que el estímulo derecho (7.5 Hz nominales) reduce su frecuencia por debajo de los 7 Hz, lo que dificulta la generación del SSEVP. Por tanto durante ese

periodo la confusión de ese estímulo se vuelve más común. El tiempo empleado para este experimento fue de alrededor de 1800 segundos (30 minutos)

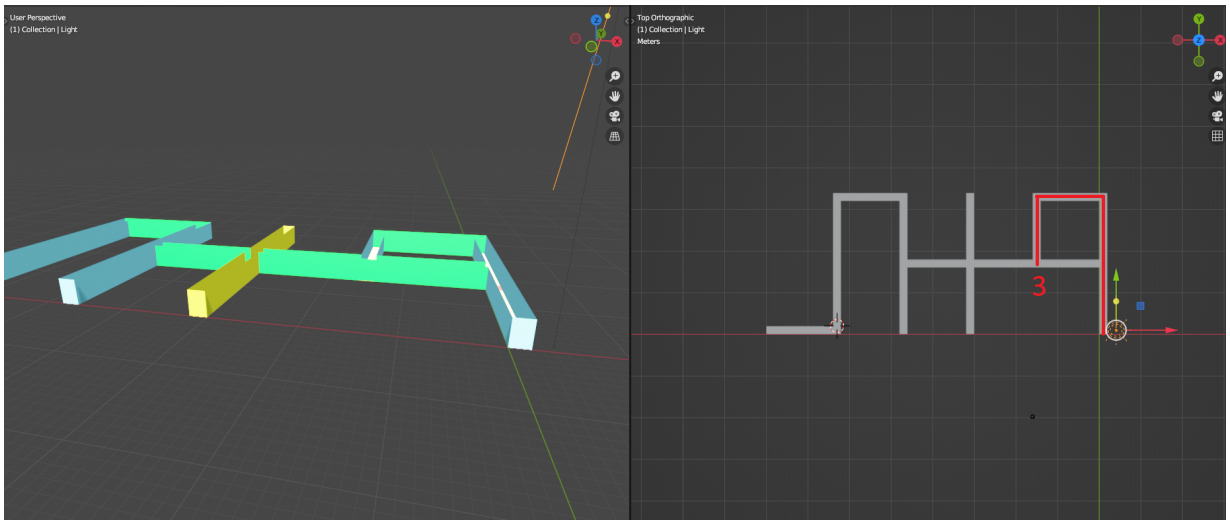

Figura 4.21: Tercera trayectoria en modo automático

El recorrido mostrado en la figura 4.21 es el que se ha seguido para la tercera prueba.

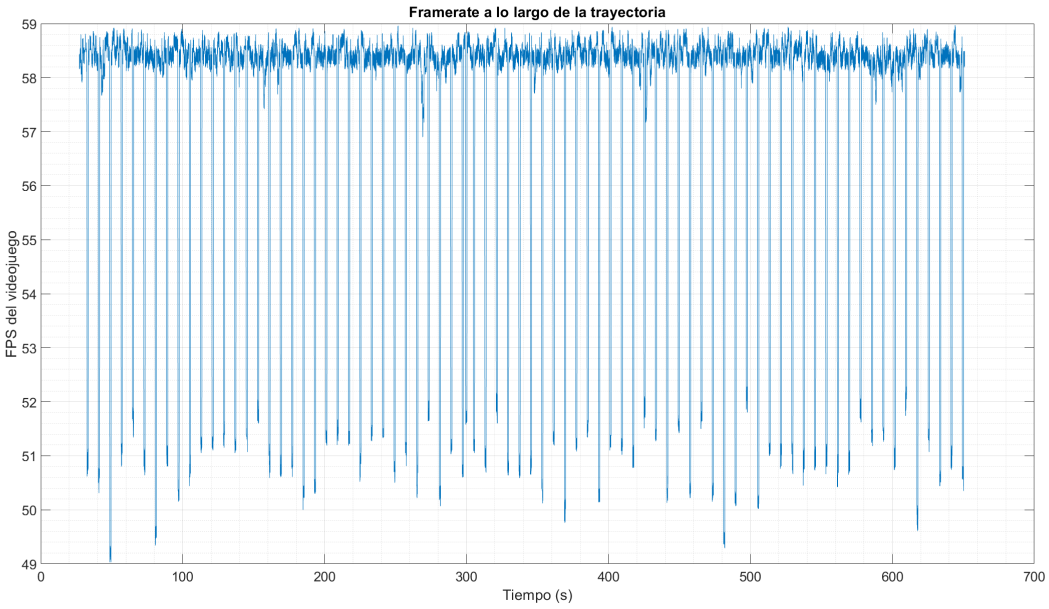

Figura 4.22: Framerate durante la tercera prueba automática

La figura 4.22, donde se muestra la tasa de frames, no muestra ninguna variación destacada con respecto a las pruebas anteriores.

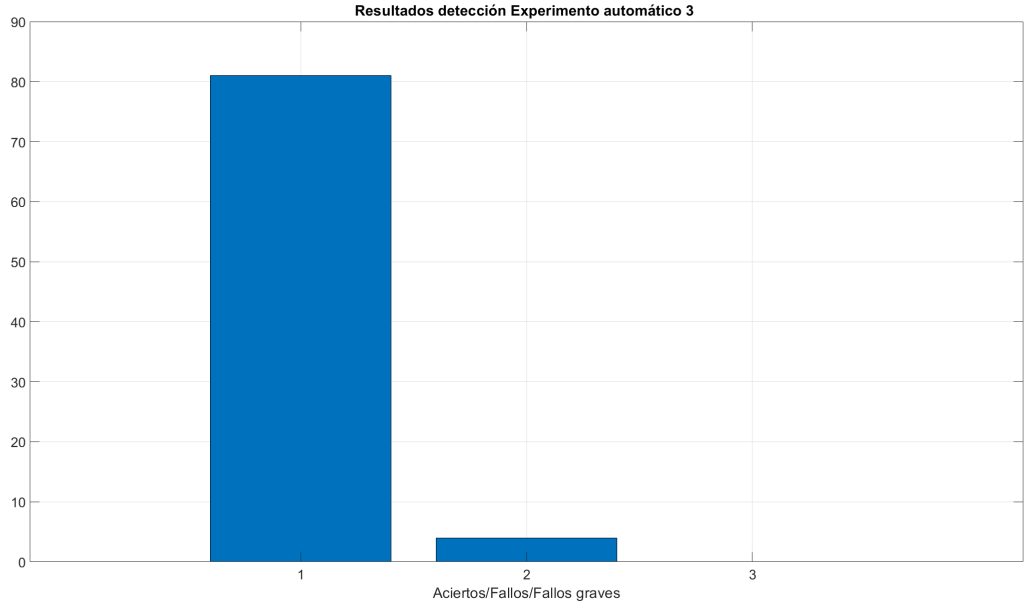

Figura 4.23: Fallos durante el tercer trayecto automático

En este tercer trayect, cuyos resultados se muestran en la figura 4.23 se han fallado solo 4 detecciones de las 85 acciones que se produjeron en total ($95{,}3\,\%$ de acierto). De nuevo ninguno de los fallos dio lugar a un acción peligrosa.

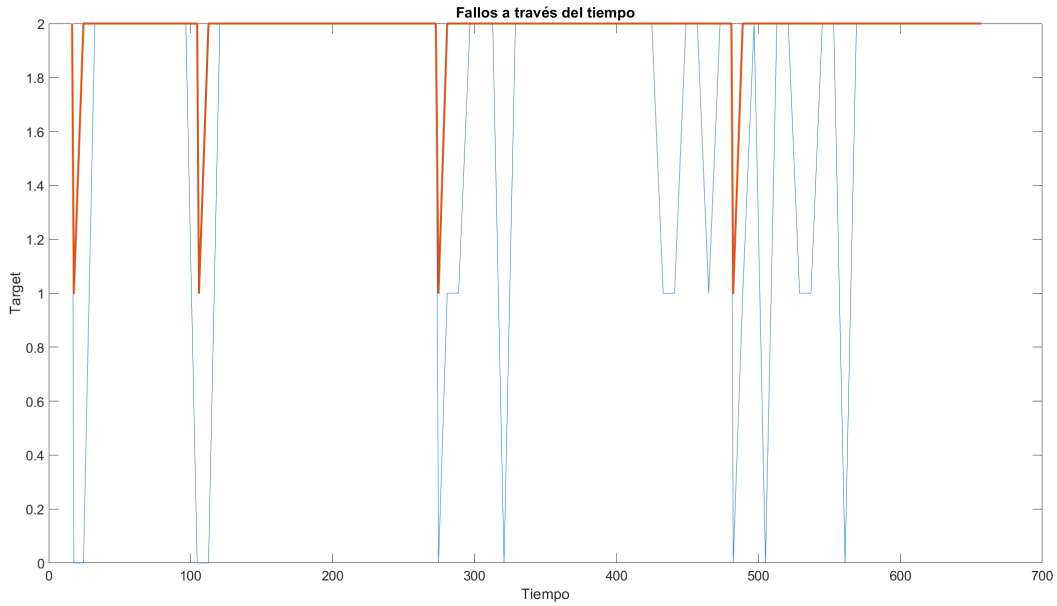

Figura 4.24: Fallos en el tiempo del tercer trayecto automático

En la figura 4.24 se muestran los fallos cometidos a lo largo de este recorrido al igual que en los dos casos anteriores. Esta vez tres de los cuatro fallos que se cometieron tuvieron lugar al confundirse el comando de confirmar con el comando de avanzar (7.5 y 10 Hz).El recorrido duró alrededor de 657 segundos (10 minutos y 57 segundos).

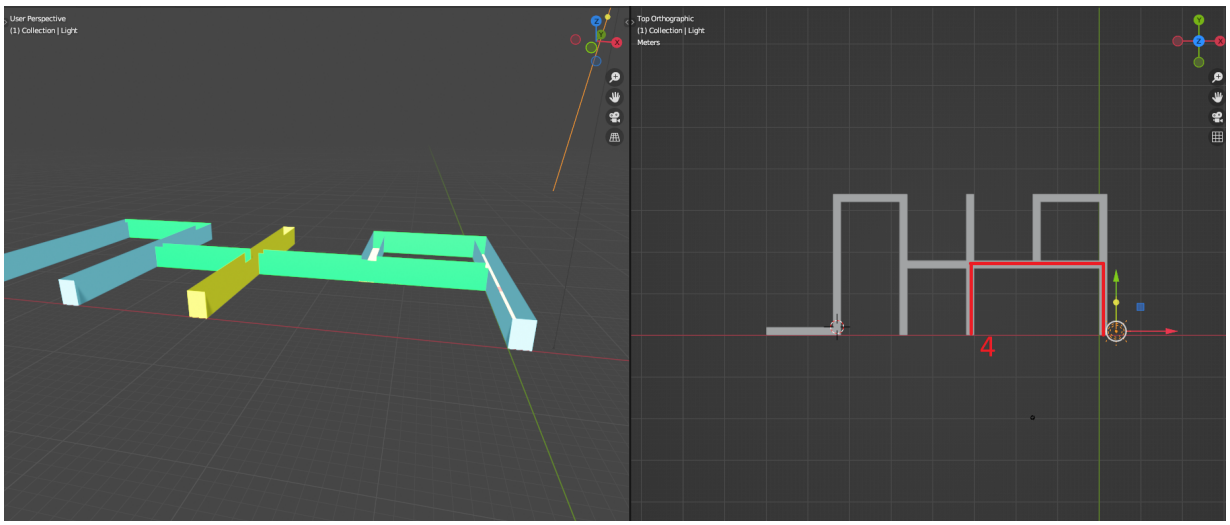

Figura 4.25: Cuarta trayectoria en modo automático

La figura 4.25 muestra el último recorrido para el que se ha realizado la prueba. Este recorrido es muy similar al trayecto 1, pero requiere menos tiempo de confirmación.

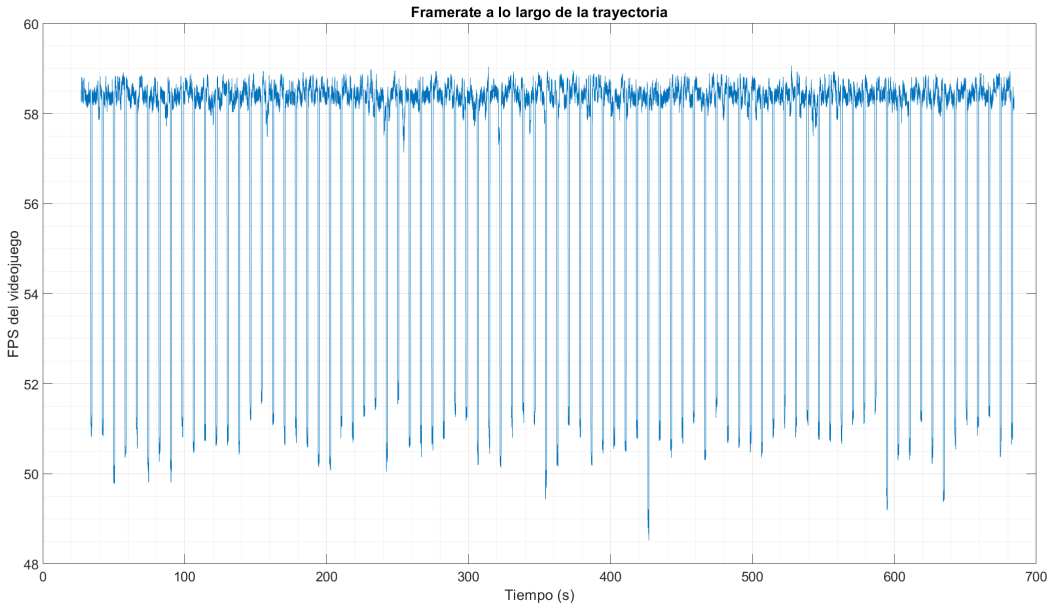

Figura 4.26: Tasa de frames durante la cuarta prueba automática

En este caso, como se puede ver en la figura 4.26 hay momentos puntuales donde caen los FPS del juego. Sin embargo esto no afectó a los resultados obtenidos.

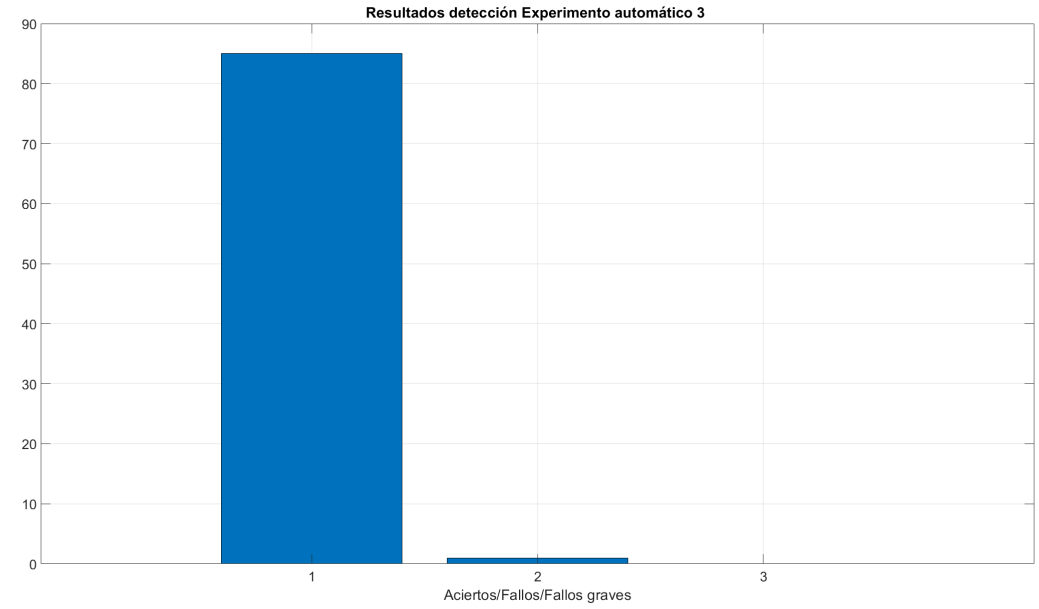

Figura 4.27: Fallos durante el cuarto trayecto

En la figura 4.27 podemos apreciar los resultados de la última prueba. De las 86 órdenes hubo un único fallo ($99\,\%$ de acierto), el cual no fue peligroso.

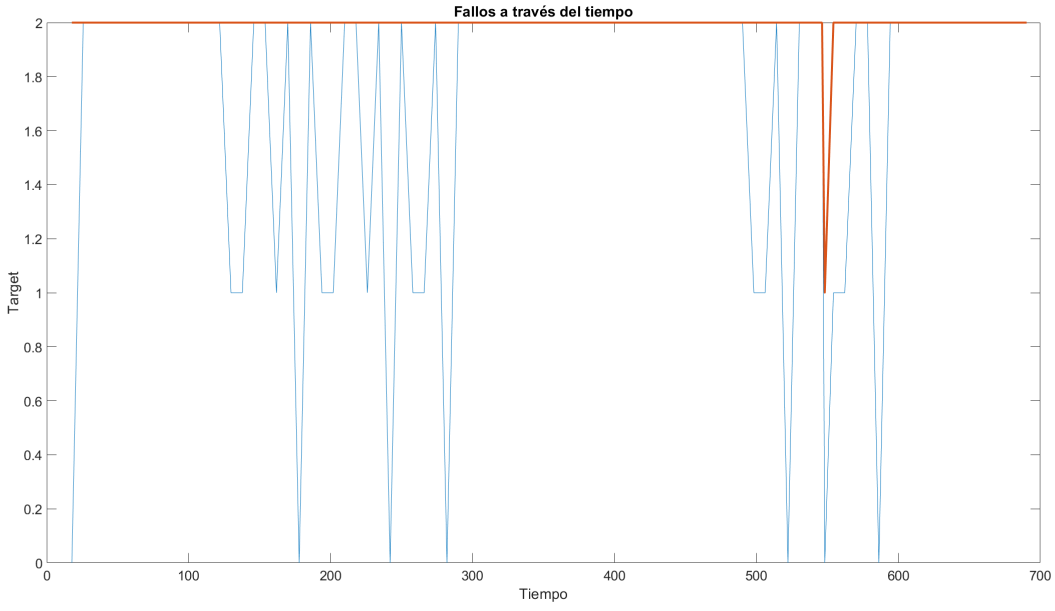

Figura 4.28: Fallos en el tiempo del cuarto trayecto automático

Como podemos ver en 4.28 se comete un único fallo en el que se confunde una orden de girar con una orden de avanzar. Este recorrido duró aproximadamente 690 segundos (11 minutos y 30 segundos). Esta trayectoria tiene la misma longitud que la anterior, pero duró 30 segundos menos. Este se debe tanto a los fallos como la experiencia en las maniobras de los giros.

En la tabla 4.2 se resumen los resultados obtenidos en el modo automático.

Trayecto	Velocidad media (m/s)	Comandos	Porcentaje de acierto
1	0.105	103	98 %
2	0.081	254	90 %
3	0.100	85	95.3 %
4	0.096	86	99 %

Tabla 4.2: Resumen de los resultados obtenidos en el modo automático para los cuatro trayectos. La velocidad media se obtuvo dividiendo la longitud del trayecto entre el tiempo total empleado en el recorrido.

4.3. Principales conclusiones obtenidas de los experimentos realizados

En suma, después de haber realizado estos experimentos se pueden sacar las siguientes conclusiones:

- El sistema permite evaluar un interfaz de usuario BCI porque da cuenta de aspectos como los errores cometidos, y algunos de los factores que inciden en ello.
- La tasa de frames del sistema juega un papel importante en el correcto funcionamiento del método. El desconocimiento de esta tasa o la caída excesiva por debajo de los 60 FPS puede llegar a inutilizar los estímulos utilizados para la detección.
- El interfaz propuesto tiene condiciones para ser robusto puesto tiene implementado un target de confirmación. Es decir, antes de poder realizar cualquier acción el usuario debe fijarse en un target destinado a confirmar la acción que se ha guardado
- Sin embargo, estas condiciones afectan a la maniobrabilidad. Se avanza poco a poco en los trayectos, puesto que para cada nueva acción individual se debe observar primero un estímulo y luego se debe atender al de confirmación. Para las acciones inversas se debe observar dos veces el mismo estímulo y después confirmarlo. En caso de fallo a la hora e detectar se pierde bastante tiempo corrigendo el error.
- Cuando hay que dar un comando muchas veces seguidas la posibilidades de error de ese comando son pequeñas.
- Por el contrario, cuando se observa un target por mucho tiempo y después se quiere dar una orden distinta la probabilidad de cometer un error es mayor.
- Por lo general, las pruebas automt́icas genera menos fallos que las manuales, posiblemente influya que el tiempo dedicado a la detección es mayor.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusión

Una vez concluido el desarrollo de la aplicación y obtenido los resultados lo único que resta es comprobar si hemos cumplido los objetivos marcados al principio del proyecto y si los resultados obtenidos cumplen con las expectativas.En primera instancia los objetivos propuestos fueron los siguientes:

- Desarrollar un sistema para estudiar el potencial del registro de las señales cerebrales o electroencefalograma (EEG) como elemento básico de un interfaz cerebro - computador.
- Construir un interfaz cerebro - computador capaz de producir comandos pertinentes para un móvil destinado a realizar trayectorias en un espacio 3D con obstáculos.
- Evaluar un interfaz cerebro computador orientado a una utilización simple, robusta y segura.

Una vez terminado el prototipo se puede concluir que los objetivos propuestos se han cumplido satisfactoriamente.

- Utilizando las herramientas proporcionadas por los desarrolladores de OpenBCI y el protocolo “LSL” se han podido registrar en tiempo real los datos del EGG y almacenarlos para su posterior uso dentro de la aplicación.
- Una vez recogidos esos datos la aplicación diseñada es capaz de analizarlos y, mediante el análisis e correlación canónica, discernir cual de los comando quiere aplicar el usuario. Con el uso de tres estímulos distintos el usuario es capaz de recorrer un laberinto. Realmente solo son necesarios dos de ellos, así que actualmente sí seria posible moverse por el espacio y no solo por el plano. Sin embargo se ha optado por la fiabilidad de los movimientos antes que por esta función. También el sistema construido es capaz de detectar colisiones con objetos que se interpongan en su camino por lo que es un buen reflejo de lo que sucederá en un escenario real.
- Con los resultados obtenidos en las pruebas en las que se utiliza el modo de funcionamiento automático se puede afirmar que la aplicación desarrollada cumple con los tres parámetros establecido en el tercero de los objetivos. Es de simple utilización, solo requiere que el usuario se ponga el casco y mire a la pantalla, es además fácil de comprender para el usuario; es robusta, pues se requiere una doble confirmación para poder realizar una acción, reduciendo la posibilidad de un error catastrófico, por tanto también es segura. Los datos obtenidos apoyan este último punto, en las pruebas de modo automático (que es el que más se asemeja a una situación real) no hubo ninguna confirmación errónea, por lo que el usuario no llegó a moverse por

error.

En relación a los resultados obtenidos en las pruebas podemos afirmar que estos se encuentran dentro de los paámetros esperados. La tasa de acierto supera el $90\,\%$ en todos los experimentos y como se menciona más arriba ninguno de esos errores desemboca en un movimiento no deseado.Es necesario destacar la importancia de conocer un todo momento y mantener lo más estable posible la tasa de fotogramas del videojuego. El desconocimiento de este valor impide el correcto cálculo de las referencias para aplicar el método de CCA. Si la tasa de fotogramas se reduce demasiado las frecuencias de los estímulos caerían por debajo de los 7 Hz, donde se dificulta enormemente la generación de los potenciales evocados visuales en estado estacionario (SSVEP) y por tanto la correcta detección de la orden del usuario.

Otro punto a discutir es el tiempo que lleva finalizar un recorrido. Los recorridos más cortos necesitaban de 15 minutos para completarse y el laberinto completo requería media hora. Si se eliminara el comando de corfirmación y se aumentara la distancia de avance este tiempo se reduciría de forma sustancial. Sin embargo, no es recomendable hacer este cambio hasta solucionar completamente la estabilidad de la tasa de fotogramas del juego.

5.2. Líneas futuras

En esta sección se van a comentar algunos de los siguientes pasos en el desarrollo de futuras aplicaciones de esta índole, así como también algunas de las funcionalidades que podrían añadirse a esta aplicación y aspectos a mejorar en el futuro.

- Uno de los primeros pasos para mejorar la aplicación es introducir una medida de calidad de la detección. En el estado actual siempre que se inicia una detección se da un resultado de correlación: sin embargo esto llega a ser un problema en el modo automático, puesto que no hay forma de saber si el usuario estaba realmente atendiendo a un estímulo o por el contrario estaba distraído, descansando o ya había llegado a su destino. Este valor de calidad podrá permitir estblecer umbrales en la detección, habilitando nuevas posibilidades y mejorando la robustez del método sin la necesidad de utilizar un estímulo de confirmación
- En relación al punto anterior, esta versión del modo automático produce falsos positivos cuando no se está atendiendo a ningún estḿulo. Una de las formas de solucionar esto será introduciendo un modo de reposo, en el que se pueda entrar para desactivar el modo automático cuando no se pretenda atender a ningún estímulo.
- Se podrían introducir nuevos métodos de funcionamiento, en el que el usuario por ejemplo atienda a un estímulo para seleccionar un destino y la ruta hacia él, dejando el movimiento en manos de otra aplicación y solo siendo necesario atender a un estímulo al principio y al final de cada recorrido
- El objetivo final de este estudio de “BCI” es poder utilizar un sistema de navegación basado en la captura el ´´EEG” en una silla robótica. Por tanto este proyecto es una simulación de un escenario real, por lo que uno de los objetivos futuro es introducir este sistema en una silla de ruedas.
- Es necesario mejorar el hardware del sistema para futuras aplicaciones. El casco que se está utilizando actualmente causa molestias en el usuario tras un tiempo de uso. Además se deberá buscar una configuración que ponga más electrodos en la zona occipital, lo que ayudaría a compensar el ruido y mejoraría la calidad de la

detección.

- Sería interesante probar nuevas configuraciones de estímulos para el prototipo, estudiando la posibilidad de añadir un mayor número de movimientos. También podrían probarse otros metódos de detección para BCI como es el caso de NextMind (NextMind (2022))

5.3. Experiencia del TFG

La experiencia de hacer este proyecto ha sido muy positiva. Escogí este proyecto porque el tema del que trataba me parecía interesante y quería hacer algo distinto. He tenido unos cuantos baches durante el desarrollo del prototipo pero todo pudo salir adelante al final. Me he visto en la necesidad de trabajar con herramientas que hasta el momento me eran desconocidas: apenas había trabajado con Python durante el grado y mucho menos con un motor de videojuegos. Tampoco era familiar con el modelado 3D, pero he tenido que aprender a utilizar Blender para el proyecto. En definitiva, creo que he aprendido a utilizar muchas herramientas nuevas gracias a este TFG y eso me parece lo más importante.

Capítulo 6

Summary and Conclusions

6.1. Conclusion

Once we have concluded the development and made the relevant experiments, the only thing left is checking if we have met the goals that were proposed in the Introduction section. Those objectives were the following:

- To develop a system por studying the potential of electroencephalography (EEG) as the key element for Brain-Computer-Interfaces (BCI).
- To build a BCI capable of producing commands for a mobile performer destined to make trajectories in a 3D space with obstacles.
- to evaluate A BCI interface oriented to a simple, safe and sturdy usage

Once the prototype was finished, we can declare that the goals were fulfilled satisfactorily.

- Using the OpenBCI tools and the “LSL” protocol we were able to register EEG data in real time and store them for its subsequent use.
- Once collected, the application analyzes the data and, using the CCA, it can discern the command that the user wants to apply. Three stimuli are enough to move through the maze. Actually, we only need two for doing that, but we use a third stimulus to confirm the previous selected action. This system is also able to detect collisions, so it is a good reflection of a real situation.
- The results obtained using the automatic mode confirms that the application fulfills the requirements established in the third objective. It is simple to use, it only requires the helmet and no previous training. It is robust, because we have reserved a stimulus por the movement confirmation. As seen in the Results section we get above 90 percent success when detecting an action.

The results we obtained are in the range we expected. We have a high success rate in every single automatic experiment. It is important to note that the framerate must be monitored through the whole experiment. An excessive fall in the FPS can result in targets hitting values under 7 Hz which makes it harder for the brain to generate SSVEP and thus the correct detection of the stimulus.

Another thing to discuss is the fact that it takes too much time to complete the maze (between 15 and 30 minutes). If we got rid of the confirmation stimulus the runs would be much faster. However, until we can fully control the framerate this option must be discarded

6.2. Future activities.

In this section we are going to discuss the next steps in the development of our BCI and also the new functionalities that could be added in the future

- One of the firsts things to do is to improve the prototype is adding a quality measurement. Right now, there is always a detection, it doesn't matter if the user wasn't focusing on a stimulus, there will always be an action. Using a quality measurement will help to decide if the result is valid each time a detection occurs. With this, we will no longer need a confirmation stimulus, so that stimulus could be used for something else.
- This version of the prototype produces false positives when the user is not focusing. One way to overcome this problem is introducing a rest mode in which there would be no further detections.
- New modes would be possible. For example a mode the user picks if he wants to select a destination and route. With this mode there would be no more detection until the users has finished the route.
- This prototype is a simulation. The final goal is to implement this system in a robotic wheelchair.
- The hardware needs to be improved. It is uncomfortable for the user after a while. It is also necessary to look for a disposition that places more electrodes in the back of the head.
- It would be interesting to test new stimuli configuration, thus adding more movement capabilities to the user. New acquisition methods could also be tested. Such as NextMind (NextMind (2022))

6.3. TFG Experience

Working on this project has been a very positive experience. I chose this project because I found it interesting and I wanted to do something different. I've had ups and downs during the development of the prototype but everything turned out well in the end. I have needed to use tools that I wasn't familiar with. I had barely programmed with Python before this project let alone using a videogame engine. I hadn't worked with a 3D computer graphics software. Definitely the best part of this project has been the acquired knowledge about all of these tools.

Capítulo 7

Presupuesto

7.1. Presupuesto estimativo

Concepto	Coste unitario	Horas	Coste total
Investigación	20	20	400€
Desarrollo	15	150	2250€
Documentación	15	30	450€
Licencias	840	1	840€
Hardware	1400	1	1400€
Instalaciones	0	200	0€
Gastos generales	50	1	50€
	Total	403	5390€

Tabla 7.1: Presupuesto del proyecto

Capítulo 8

Bibliografía

Blender (2022). Website de blender. blender.org. Accessed: 2022-09-12.

Epihaius, panda3dmastercoder, Thaumaturge, Sampenland, and Rdb (2021). Collision mesh from loaded model for built-in collision system. https://discourse.panda3d.org/t/collision-mesh-from-loaded-model-for-built-in-collision-system/27102.

Korstanje, J. (2021). Canonical correlation analysis. https://towardsdatascience.com/canonical-correlation-analysis-b1a38847219d.

Kothe, C., Medine, D., Boulay, C., Grivich, M., and Stenner, T. (2019). Introduction. https://labstreaminglayer.readthedocs.io/info/intro.html#streaming-layer-api.

Lalor, E. C., Kelly, S. P., Finucane, C., Burke, R., Smith, R., Reilly, R. B., and McDarby, G. (2005). Steady-state vep-based brain-computer interface control in an immersive 3d gaming environment. *EURASIP Journal on Advances in Signal Processing*, 2005(19).

Lcuyer, A., Lotte, F., Reilly, R., Leeb, R., Hirose, M., and Slater, M. (2008). *Brain-Computer Interfaces, Virtual Reality, and Videogames*, volume 41.

Leeb, R., Friedman, D., Müller-Putz, G. R., Scherer, R., Slater, M., and Pfurtscheller, G. (2007). Self-paced (asynchronous) bci control of a wheelchair in virtual environments: A case study with a tetraplegic. *Computational Intelligence and Neuroscience*, 2007:1–8.

Lin, Z., Zhang, C., Wu, W., and Gao, X. (2006). Frequency recognition based on canonical correlation analysis for ssvep-based bcis. *IEEE Transactions on Biomedical Engineering*, 53(12):2610–2614.

NextMind (2022). Website de nextmind. next-mind.com. Accessed: 2022-09-12.

OpenBCI (2021). Ultracortex mark iv: Openbci documentation. https://docs.openbci.com/AddOns/Headwear/MarkIV/.

OpenBCI (2022). Open source tools for neuroscience. https://openbci.com/about.

Penn State's Department of Statistics (2016). Lesson 13: Canonical correlation analysis: Stat 505. https://online.stat.psu.edu/stat505/lesson/13.

ScikitLearn (2022). Cross decomposition. https://scikit-learn.org/stable/modules/cross_decomposition.html#cross-decomposition. Accessed: 2022-09-12.

Touyama, H., Aotsuka, M., and Hirose, M. (2008). A pilot study on virtual camera control via steady-state vep in immersing virtual environments. In *Proceedings of the Third IASTED International Conference on Human Computer Interaction*, HCI '08, page 43–48, USA. ACTA Press.

Wang, Y., Chen, X., Gao, X., and Gao, S. (2016). *A Benchmark Dataset for SSVEP-Based Brain-Computer Interfaces*, volume PP.