UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Diseño y programación de algoritmos para el preprocesado y análisis de la complejidad de señales de electroencefalograma

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Prado Raposo, Sergio

Tutor/a: Monzó Ferrer, José María

Cotutor/a: Rey Solaz, Beatriz

CURSO ACADÉMICO: 2021/2022

Agradecimientos

Este proyecto va dedicado a mi familia, que me ha brindado la oportunidad de poder estudiar esta titulación, sin ellos no hubiera sido posible llegar hasta aquí. Pero en especial, agradecer a mi madre por estar siempre ahí y darme la ayuda y el apoyo necesario para poder superar cada uno de los obstáculos que han aparecido por el camino.

Agradecer también a mis tutores Jose y Bea por darme la oportunidad de trabajar en este proyecto. Gracias por compartir vuestros conocimientos conmigo, por ayudarme en todo momento a conseguir los objetivos y, sobre todo, por hacerlo siempre de una forma tan amable. Gracias a vuestro apoyo ha sido más fácil sacar el proyecto adelante.

Por último, agradecer a mis amigos por estar siempre dispuestos a ayudarme y por animarme en cada uno de los peores momentos. Y, por supuesto, a mi familia Erasmus, gracias a ellos he podido vivir la mejor experiencia de mi vida.

Resumen

El electroencefalograma (EEG) proporciona amplia información sobre la actividad eléctrica y dinámicas del cerebro. Para caracterizar las señales registradas con dicha técnica, se pueden aplicar distintas técnicas de procesado, entre ellas las medidas de complejidad de la señal.

El presente trabajo consistirá en el diseño y programación en Matlab de algoritmos para caracterizar de forma automática las señales de EEG desde el punto de vista de su complejidad. Para ello, se programarán distintos tipos de algoritmos, tanto para el preprocesado de la señal (filtrado, corrección de artefactos) como para el análisis de la complejidad de la señal (incluyendo el algoritmo de la entropía muestral).

Una vez programados los algoritmos, se realizará una validación de éstos con datos de EEG.

Resum

L'electroencefalograma (EEG) proporciona àmplia informació sobre l'activitat elèctrica i dinàmiques del cervell. Per a caracteritzar els senyals registrades amb dita tècnica, es poden aplicar distintes tècniques de processat, entre elles les mesures de complexitat del senyal.

El present treball consistirà en el disseny i programació en Matlab d'algoritmes per a caracteritzar de forma automàtica els senyals d'EEG des del punt de vista de la seua complexitat. Per a això, es programaran distints tipus d'algoritmes, tant per al preprocessat del senyal (filtrat, correcció d'artefactes) com per a l'anàlisi de la complexitat del senyal (incloent l'algoritme de l'entropia mostral).

Una vegada programats els algoritmes, es realitzarà una validació d'estos amb dades d'EEG.

Abstract

The electroencephalogram (EEG) provides extensive information on the electrical activity and dynamics of the brain. To characterize the signals recorded with this technique, different processing techniques can be applied, including signal complexity measurements.

The present work will consist of the design and programming in Matlab of algorithms to automatically characterize EEG signals from the point of view of their complexity. To do this, different types of algorithms will be programmed, both for signal preprocessing (filtering, artifact correction) and for signal complexity analysis (including the sample entropy algorithm).

Once the algorithms have been programmed, they will be validated with EEG data.

LISTA DE FIGURAS

LISTA DE TABLAS

Lista de Acrónimos

SNC – Sistema Nervioso Central

SNP – Sistema Nervioso Periférico

EEG – Electroencefalograma

ECoG – Electrocorticograma

E-EEG – Estéreo Electroencefalograma

Hz – Hercios

Fc – Frecuencia de corte

ms - milisegundos

Vmin – Tensión Mínima

Vmáx – Tensión Máxima

ÍNDICE

Capítulo 1. Introducción

1.1 Motivación

Desde que era pequeño siempre me ha gustado ayudar a las personas, y que mejor forma de hacerlo que con algo relacionado con la medicina. Gracias a la tecnología, se pueden aportar tanto mejoras como nuevos desarrollos que ayudan a salvar vidas.

El grado de telecomunicaciones sirve de ayuda en todos los procesos de análisis de datos. Desde su adquisición, preprocesado, procesado hasta todo tipo de análisis. Este proceso ayuda a interpretar todo tipo de resultados y sacar conclusiones sobre cualquier tipo de enfermedad fisiológica: enfermedades cardiovasculares, cerebrales, motrices…

En el caso de mi estudio, nos centraremos en el cerebro, lo que podemos considerar la "máquina" de las personas. Es fascinante ver como una pequeña parte de nuestro cuerpo es la culpable de todos los movimientos de nuestro cuerpo. Sin actividad cerebral, no hay vida. El estudio de las señales cerebrales ayuda a ver cómo reacciona una persona ante ciertos estímulos, y a su vez, podemos detectar anomalías que nos ayuden a prevenir enfermedades.

Por tanto, mi motivación principal es adentrarme en el mundo de la biomedicina con este proyecto y obtener una visión más global de la dificultad e importancia de este tipo de estudios, y poder adquirir conocimientos para poder aplicarlos en futuros estudios o proyectos que tenga la oportunidad de participar. Siempre con mi meta principal en mente, la de poder ayudar a las personas a tener una vida mejor.

1.2 Explicación del objetivo del trabajo

El objetivo principal del proyecto es el desarrollo de una interfaz gráfica que facilite el uso de los algoritmos de complejidad para el análisis de señales de EEG, ya que EEGLAB no incluye estos algoritmos.

Los subobjetivos seguidos en el proyecto para alcanzar el objetivo principal han sido los siguientes:

1) Definir flujo de trabajo para procesar señales de EEG desde el punto de vista de la complejidad.
2) Preprocesar de la señal mediante EEGLAB en la que se realiza un filtrado y corrección de artefactos de las señales proporcionadas.
3) Diseñar y programar algoritmos en Matlab para descartar las señales que no cumplan las condiciones exigidas por el estudio.
4) Analizar la complejidad de la señal mediante un procesado de la señal utilizando el algoritmo de la entropía muestral.
5) Implementar una interfaz gráfica para facilitar el uso de los algoritmos de complejidad de la señal, pudiendo ser utilizada para estudios diferentes y diferentes usuarios.

1.3 Planificación del trabajo

La metodología llevada a cabo en este proyecto consiste en lo siguiente:

En primer lugar, se van a utilizar señales reales de estudios para validar el procedimiento, proporcionadas por la Universidad de Baleares. Cabe decir que nuestro objetivo no es analizar la señales, sino usar dichas señales para validar el desarrollo.

A continuación, se definirá un flujo de trabajo en el que se indicará el método a seguir para el procesamiento de las señales de EEG desde el punto de vista de la complejidad.

Tras esto, se realizará un primer preprocesado mediante el uso de EEGLAB, donde se llevará a cabo un filtrado y corrección de artefactos, en el cual se detectarán las señales defectuosas, pudiendo descartar las primeras señales que no nos proporcionen información relevante en nuestro estudio.

Luego, s realizará un segundo preprocesado mediante el diseño y desarrollo de algoritmos en Matlab, siguiendo una serie de condiciones que debe cumplir nuestro estudio. En esta parte se volverá a descartar las señales que no cumplan dichas condiciones y nos quedaremos finalmente con las señales válidas que cumplan las restricciones y a las que podremos aplicar la última parte del estudio.

Por último, se realizará un procesado, en el que se analizará la entropía muestral por épocas y global de cada señal válida.

Todo esto será implementado en una interfaz gráfica, que podrá ser utilizada para realizar cualquier tipo de filtrado y posterior estudio de la entropía, en cualquier base de datos proporcionada.

1.4 Estructura del documento

En este apartado se hará un pequeño resumen de los capítulos que se van a abordar en el trabajo:

1) **Introducción:** En este capítulo se explicará la motivación que ha llevado a realizar este proyecto, así como los objetivos del trabajo y la metodología que se ha llevado a cabo para poder alcanzar cada uno de ellos.

2) **Marco Teórico:** En este capítulo se realizará una pequeña introducción al sistema nervioso humano, haciendo hincapié a la anatomía y fisiología del cerebro humano, así como todo lo relacionado al electroencefalograma. También se explicará cómo se han estructurado los datos.

3) **Preprocesado de datos a través de la aplicación EEGLAB de Matlab:** En este capítulo se hará un pequeño resumen del funcionamiento de la aplicación EEGLAB, además del uso que ha tenido en nuestro trabajo en el preprocesado de los datos.

4) **Diseño y programación de algoritmos en Matlab para caracterizar de forma automática las señales de EEG desde el punto de vista de su complejidad:** En este capítulo se explicará de manera detallada todo el diseño y desarrollo de algoritmos que se ha llevado cabo para alcanzar los objetivos marcados siguiendo las condiciones marcadas por el estudio, así como el análisis del procesado final mediante la entropía muestral.

5) **App Designer:** En este capítulo, se realizará una validación de toda la aplicación mediante la ejecución de la base de datos proporcionada. Además, se explicarán los resultados obtenidos.

6) **Conclusiones y líneas futuras:** Se llevará a cabo las conclusiones obtenidas tras el desarrollo del proyecto y las futuras aplicaciones que se pueden llevar a cabo.

Capítulo 2. Marco Teórico

2.1 Sistema nervioso humano

El sistema nervioso humano es uno de los más complejos del ser humano. Se encarga principalmente, de recibir y procesar la información tanto del cuerpo humano, como del entorno y tras esto controlar las funciones de nuestro organismo.

El sistema nervioso se compone de dos células: las neuronas y las células gliales:

Las **neuronas** son las encargadas de percibir la información de nuestro cuerpo, procesarla, y a su vez se encargan de mandar respuestas a nuestro organismo, mientras que las **células gliales** son las encargadas de proteger a las neuronas y darles soporte. Sin las células gliales, las neuronas no podrían llevar a cabo sus funciones.

El sistema nervioso se compone por dos partes: el **sistema nervioso central (SNC) y el sistema nervioso periférico (SNP).**

El SNC está compuesto por el encéfalo (cerebro, cerebelo, bulbo raquídeo) y medula espinal, cuya función es el procesado de la información y el control del funcionamiento del cuerpo humano. A continuación, se detallan cada una de las funciones que tiene cada parte.

- El **cerebro,** que será explicado con más detalle en el siguiente punto, es el "eje" de todas nuestras acciones, tanto conscientes como inconscientes. Es uno de los órganos más delicados del ser humano, y de los más complejos. Es el que nos permite tener pensamientos, sentir, y gracias a sus capacidades cognitivas nos permite percibir estímulos, procesar la información y actuar para resolver cualquier problema que se nos presenta. Todo lo que hacemos, lo podemos realizar gracias a él.

- El **cerebelo** tiene como principal función la de recibir la información proveniente de otras estructuras del cerebro, de la medula espinal y de los receptores sensoriales y lo conecta con el aparato locomotor, es decir, huesos, músculos, articulaciones… para dar una respuesta de manera coordinada y uniforme.

- El **bulbo raquídeo** está situado justo debajo del cerebelo y tiene como principal función el control de la frecuencia cardiaca, la presión sanguínea y la respiración. También es la vía de paso para muchos nervios que transporta la información transmitida entre el cerebro y la medula espinal.

- La **médula espinal** es lo que conecta el cerebro con el resto del organismo. Contiene fibras nerviosas, neuronas que son encargadas de enviar información y órdenes a zonas del cuerpo para posteriormente realizar una acción. Por tanto, sus funciones principales son el procesamiento de la información y la transmisión de dicha información sensorial y motora. Otra de las tareas que tiene son los reflejos, que son reacciones rápidas ante una situación de peligro.

Se puede ver el SNC en la **Figura 1:**

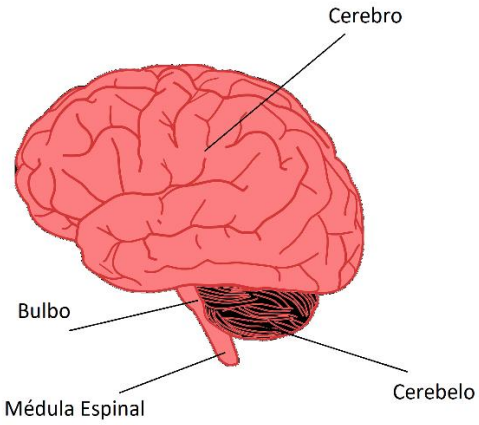


Figura 1: Sistema Nervioso Central.

El SNP abarca todos los nervios que salen del SNC hacia todo el cuerpo. Está compuesto por nervios y ganglios nerviosos. Se divide en dos partes, **el sistema nervioso somático**, que es el encargado de llevar toda la información sensorial y motora hasta el SNC y desde el SNC al resto del cuerpo. Por otro lardo, está **el sistema nervioso autónomo**, que es el encargado de todas las funciones involuntarias del organismo, como es el latido del corazón, digestión, respiración, temperatura corporal… Se puede apreciar el sistema nervioso humano en la **Figura 2:**

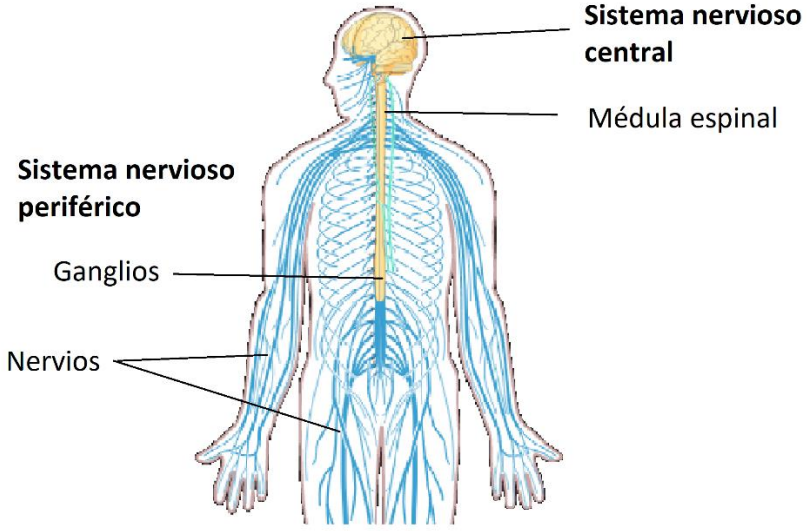


Figura 2: Sistema Nervioso Humano.

2.2 Anatomía y fisiología del cerebro humano

El cerebro está formado por una capa externa que se le denomina corteza cerebral y ésta está conectada por una red de axones a la parte interior del cerebro, que se compone del bulbo raquídeo y las unidades sensoriales y motoras.

Además, se compone por dos mitades que se les denomina hemisferios: el izquierdo y el derecho; ambos hemisferios están conectados, y cada uno tiene sus propias funciones, a pesar de que ciertas funciones son compartidas.

El hemisferio izquierdo está relacionado con la expresión verbal, las capacidades lógicas. Entre sus funciones se encuentran la capacidad de expresión, es decir, el habla; el razonamiento, la resolución de problemas, el aprendizaje… Además, controla los movimientos del lado derecho del cuerpo.

El hemisferio derecho trata la expresión no verbal. Entre sus funciones se encuentran las emociones, los sentimientos, la imaginación y, además también la orientación espacial. Y, controla los movimientos del lado izquierdo del cuerpo.

Se pueden apreciar las funciones de cada hemisferio en la **Figura 3:**

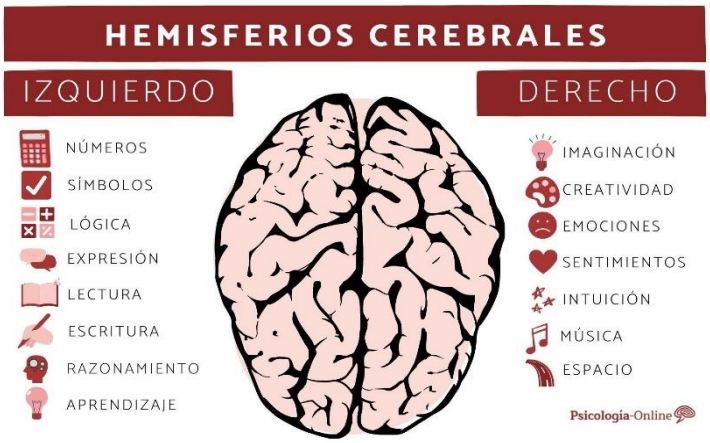


Figura 3: Hemisferios cerebrales [7].

Dentro de cada hemisferio nos encontramos con diferentes zonas, en la que cada una se desarrolla diferentes funciones:

El **lóbulo frontal** ocupa aproximadamente 1/3 de la superficie del cerebro, lo que le hace ser el lóbulo más grande del encéfalo humano. Las principales funciones que atribuimos a dicho lóbulo son las asociadas con las capacidades cognitivas, es decir, las que influyen en la conducta, pensamiento y personalidad de cada persona.

El **lóbulo parietal** está situado en la parte trasera del encéfalo y trabaja conjuntamente con todos los lóbulos del cerebro. Las principales funciones que tiene son la integración sensorial, es decir, es capaz de reunir la información proveniente de los sentidos e integrarla y tener una percepción sensorial completa. Otra de las funciones importantes que tiene es que nos posibilita el habla y el procesamiento de la información matemática.

El **lóbulo occipital** es el lóbulo más pequeño y se encuentra en la parte trasera del cerebro. La principal función de este lóbulo está relacionada con la vista, es capaz de procesar la información que vemos y responder mediante emociones. También relacionado con la vista, es que posibilita la identificación de colores, el reconocimiento de objetos o rostros conocidos, la captación de

movimientos fuera de nuestro campo visual y es capaz de transmitir la información recibida a otras zonas del cerebro y crear recuerdos.

El **lóbulo temporal** es una de las zonas más importantes de la corteza cerebral, es la encargada de integrar gran parte de la información sensorial que llega del entorno, por tanto, tiene muchas funciones. Entre ellas destaca que tiene una gran importancia en el proceso de información, tanto auditivo como visual. También interviene en las emociones y la memoria, tanto a corto plazo como a largo plazo. Además, también tiene relación con la escritura y el procesamiento y comprensión del lenguaje.

Cada una de las partes, se pueden apreciar en la **Figura 4:**

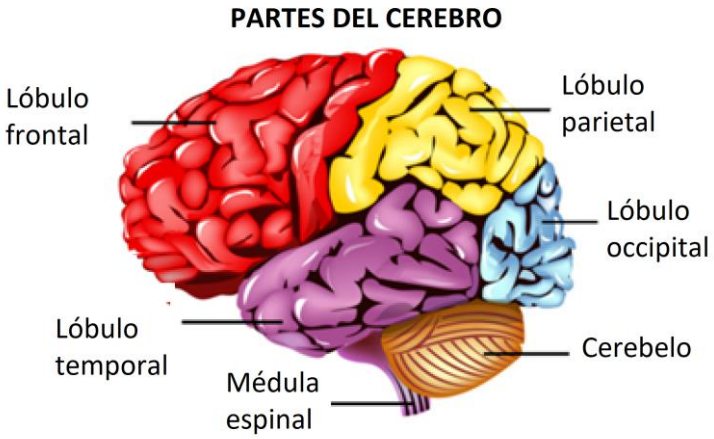


Figura 4: Partes del cerebro.

2.3 Electroencefalograma

El electroencefalograma (EEG) es un método que detecta y analiza la actividad eléctrica en el cerebro mediante pequeños sensores metálicos, llamados electrodos, que se colocan sobre el cuero cabelludo. Esta prueba permite detectar anomalías o alteraciones en la actividad cerebral que ayudan al diagnóstico de trastornos como la epilepsia, encefalopatía, daños cerebrales…

Remontándose a los inicios, el EEG fue desarrollado en 1924 por el neurólogo y psiquiatra alemán Hans Berger. Sus primeros estudios fueron sobre un joven que padecía un defecto en la tabla ósea del cráneo y, más adelante, ya comenzó a realizar estudios sobre personas que no padecían ningún trastorno cerebral. Los primeros estudios los realizaba utilizando electrodos de aguja y un galvanómetro de cuerda. Tras esto, fue publicando distintos informes en el que destacó la observación de continuas anomalías en la actividad electroencefalográfica en pacientes que padecían epilepsia.

Las señales típicas del EEG presentan una amplitud que oscila entre los 10 y 100 µV y el rango de frecuencias que suele presentar es entre 1 y 40 Hz. Más adelante, se detallará cada una de estas frecuencias.

2.3.1 Captación de la señal del EEG

En términos de medicina, los electrodos son dispositivos que mediante placas metálicas o agujas pequeñas transmiten una corriente eléctrica de un conductor a una parte del cuerpo o a otro medio de un paciente. Éstos llevan las señales eléctricas hasta los aparatos donde se registran para ayudar a diagnosticar las anomalías que puedan detectar.

Actualmente, la captación de la actividad cerebral en un EEG consta de cuatro procedimientos:

1. Mediante electrodos superficiales en el cuero cabelludo, que consta de tres tipos distintos: adheridos, de contacto y de casco de malla.
2. Mediante electrodos basales en la base del cráneo.
3. Mediante electrodos quirúrgicos corticales en el cerebro expuesto.
4. Mediante electrodos quirúrgicos intracraneales en una localización cerebral profunda.

Se puede comprobar en la **Figura 5.**

Figura 5: Electrodos superficiales [48], adheridos [48], de contacto y casco de malla [47].

Además, se puede registrar la actividad cerebral de tres formas diferentes.

- **Electroencefalograma** (EEG) en el que se utilizan los electrodos superficiales o basales.
- **Electrocorticograma** (ECoG) en el que se utilizan los electrodos quirúrgicos en la superficie de la corteza cerebral.
- **Estéreo electroencefalograma** (E-EEG) que son los utilizados para las localizaciones cerebrales profundas.

2.3.2 Colocación de los electrodos

Como se ha comentado en puntos anteriores, la capa exterior del cerebro se denomina corteza cerebral y está dividido en cuatro regiones: el lóbulo occipital, frontal, parietal y temporal. Cada una de ellas realiza unas funciones específicas.

Para grabar nuestros datos en el EEG es importante realizar una buena ubicación de los electrodos, es por ello por lo que se puso en práctica el sistema internacional de posicionamiento 10-20 que indica en qué lugar del cráneo deben estar colocados los electrodos para realizar las pertinentes mediciones en el EEG. El número de electrodos debe ser un mínimo de 21 electrodos.

El sistema de posicionamiento 10-20 se refiere a que la distribución de distancias entre los electrodos contiguos puede ser del 10% o 20% de la distancia total entre el nasión y el inion, como podemos ver en la **Figura 6**.

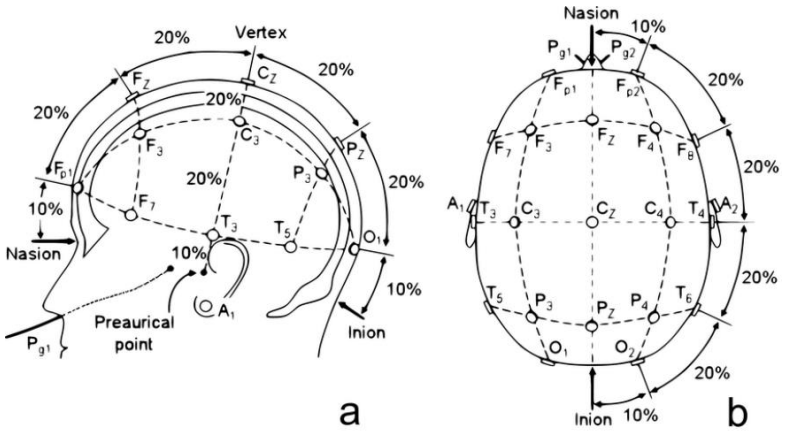


Figura 6: Sistema de posicionamiento 10-20. Punto de vista (a) lado izquierdo, punto de vista (b) sobre la cabeza [31].

Se utilizan cuatro puntos de referencia craneales universales: el primero de ellos es el **nasión**, que es el punto de intersección entre el hueso frontal y dos huesos nasales del cráneo, es el punto más profundo del puente nasal, el segundo de ellos es el **inion**, el cual es el punto de inserción del ligamento nucal y el trapecio y es la proyección más prominente del hueso occipital, y por último tenemos los **puntos pre-auriculares** que están situados detrás de los oídos.

La colocación de los electrodos está etiquetada en función al lóbulo que se refiera. Aunque no exista lóbulo central, se utiliza también para la identificación. Al Igual que el indicador frontopolar. Lo podemos observar en la **Tabla 1**.

Identificador Electrodo	Área o lóbulo
O	Occipital
P	Parietal
C	Central
T	Temporal
F	Frontal
FP	Frontopolar

Tabla 1: Identificadores Electrodos.

Como podemos ver en la **Figura 7**, los números impares (1, 3, 5 y 7) corresponden a los electrodos que están situados en el hemisferio izquierdo del cerebro; mientras que los números pares (2, 4, 6 y 8) se refieren a los electrodos colocados en el hemisferio derecho del cerebro. En cuanto a los electrodos que están colocados en la línea central están etiquetados por la letra "z".

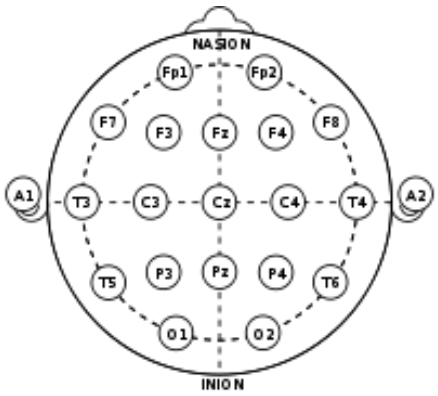


Figura 7: Colocación de electrodos en sistema internacional 10-20 [46].

La **Figura 8**, muestra un ejemplo de un EEG:

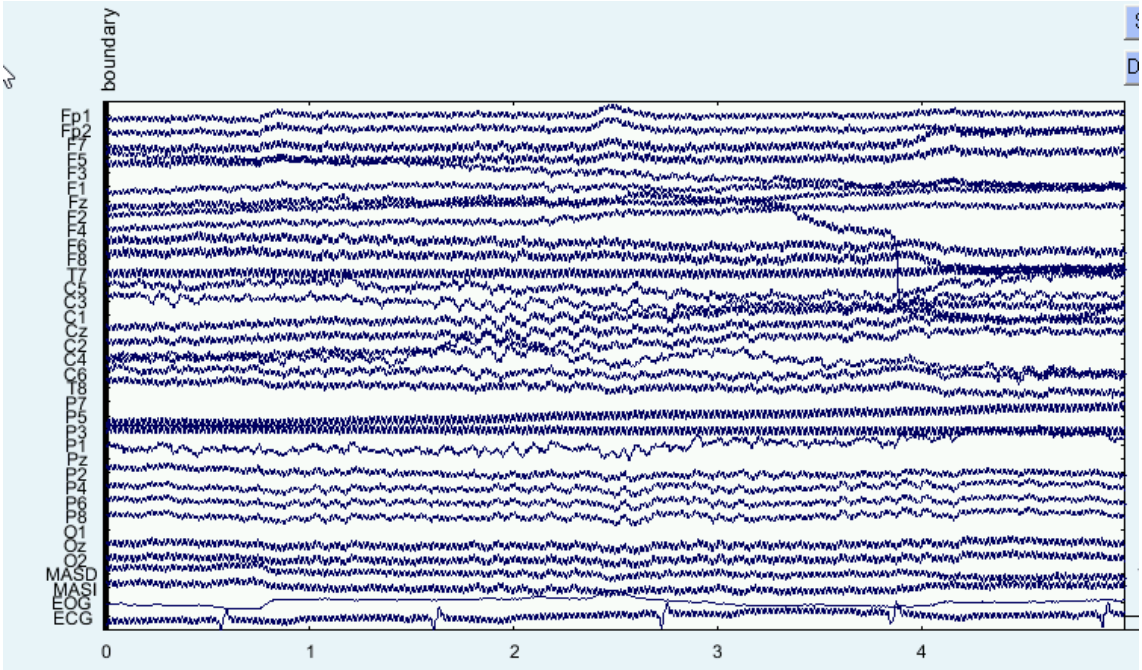


Figura 8: Ejemplo de EEG.

2.3.3 Tipos de montaje

Una vez colocados los electrodos y captada la señal, se procede a registrar las señales procedentes del EEG. Para ello, debemos diferenciar entre dos tipos de registros: monopolares y bipolares.

En los **registros monopolares** se captura cada una de las señales procedentes de cada uno de los electrodos de manera independiente. El electrodo en el que se registra la señal se le denomina electrodo activo, y a su vez el cable conectado al equipo se toma de un electrodo denominado, electrodo de referencia. Este electrodo debe estar a un potencial de 0V para no perder ningún tipo de información. Se puede ver en la **Figura 9.**

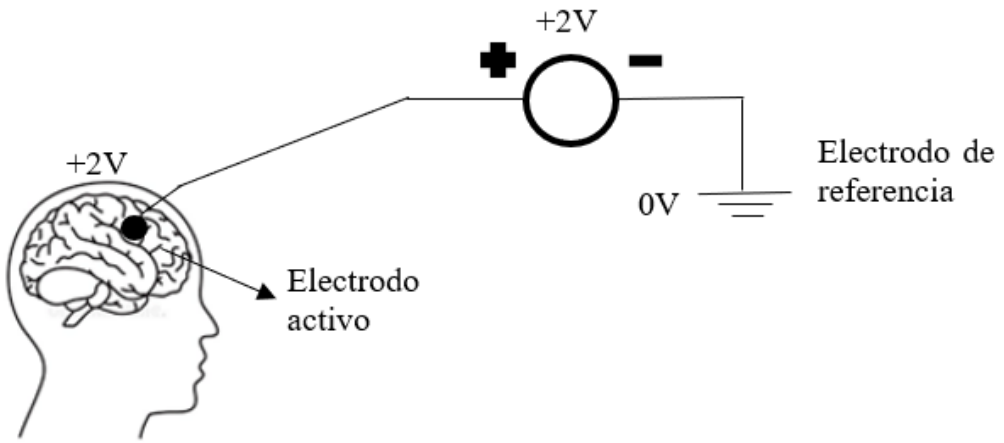


Figura 9: Registro Monopolar.

En los **registros bipolares** se captura tomando parejas de electrodos y se registra la diferencia de tensión entre cada par de puntos. Es muy importante escoger las parejas de electrodos adecuadas para que nos proporcione información útil. Los electrodos utilizados como pareja se denominan electrodos activos. Se puede ver en la **Figura 10.**

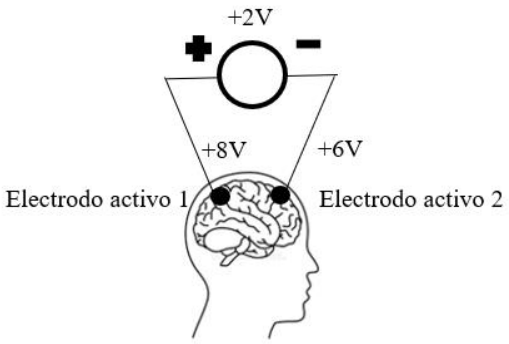


Figura 10: Registro Bipolar.

Dentro de los registros bipolares, tenemos dos tipos: los longitudinales y transversales. Como podemos ver en la **Figura 11**.

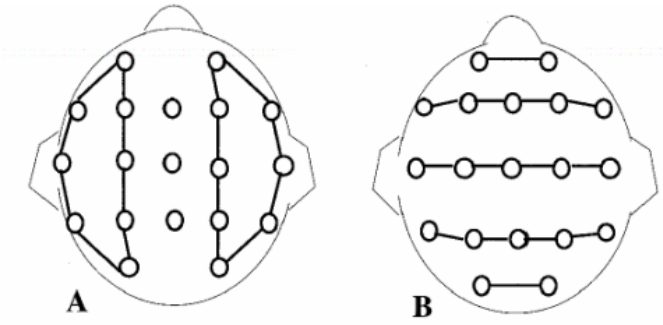


Figura 11: A - Registro Bipolar longitudinal. B - Registro Bipolar transversal [38].

2.3.4 Ritmos cerebrales recogidos en el EEG

La información que se registra se compone de un número variado de ondas que pueden aparecer tanto aisladas como en grupos (ritmos cerebrales). Estas ondas se diferencian entre ellas por una serie de parámetros:

Uno de estos parámetros es la frecuencia, que indica la cantidad de veces que aparece un tipo de onda formando parte de un ritmo cerebral. Hay distintos tipos de frecuencias:

- **Ondas Delta:** Son las ondas que oscilan entre los 1-4 Hz. Son ondas muy lentas, pero que presentan la mayor amplitud. Estas ondas se asocian con el sueño profundo y la anestesia, es decir, son las ondas que aparecen en situaciones cerebrales donde el individuo no tiene conciencia de la actividad. Por lo tanto, estas ondas deltas se encuentran estrechamente relacionadas con el estado de tranquilidad y la forma de natural de conseguir que nuestro cerebro produzca dichas ondas es mediante el sueño profundo y la meditación.

- **Ondas Theta:** Son las ondas que oscilan entre los 4-8 Hz. Estas ondas se asocian a las capacidades imaginativas, la reflexión y el sueño. Estas ondas producen muchos beneficios, como puede ser la reducción de la ansiedad y dolores crónicos.

- **Ondas Alfa:** Son las ondas que oscilan entre los 8-12 Hz. Estas ondas se asocian al estado de calma, pero cuando no hay sueño, es decir el momento en el que hay relajación. Por lo que, un nivel alto de estas ondas puede propiciar falta de concentración, y una disminución de éstas pueden conllevar estados de estrés e insomnios.

- **Ondas Beta:** Son las ondas que oscilan entre los 12-30 Hz. Estas ondas se asocian al estado de una actividad neuronal intensa, es decir, aparecen cuando estamos realizando una actividad normal en la vida cotidiana, donde estamos prestando mayor atención, o también en estados de vigilia, resolviendo problemas o pensando y razonando situaciones complejas.

- **Ondas Gamma:** Son las ondas que aparecen a partir de los 30 Hz. Son ondas con una frecuencia muy rápida y son difíciles de captar para los electroencefalogramas. Es un tipo de onda que tiene su origen en el tálamo y puede servir para detectar algún tipo de enfermedad. Además, se asocian a estados de felicidad y con tareas de una necesidad alta de procesamiento cognitivo.

En la **Tabla 2**, se recogen las ondas y el tipo de forma de cada una de ellas.

ONDA (FRECUENCIA)	FORMA DE ONDA
DELTA (1-4 Hz)	
THETA (4-8 Hz)	
ALFA (8-12 Hz)	
BETA (12-30 Hz)	
GAMMA (>30 Hz)	

Tabla 2: Tipos de ritmos cerebrales.

2.4 Estructura de los datos

La señal de EEG que se ha utilizado para validar los algoritmos fue tomada en la Universidad de Baleares y contiene registros de electroencefalograma de pacientes. Ésta se registró a partir de 32 electrodos en cuero cabelludo colocados siguiendo el sistema internacional 10/20 y con electrodos en mastoides como referencia.

Las señales se adquirieron utilizando un amplificador Brain Amp con una frecuencia de muestreo de 1000 Hz, un filtro paso alto a 0,10 Hz, un filtro de paso bajo a 70 Hz y un filtro de muestra de 50 Hz. En relación con el tiempo del estímulo va de -100ms a 900ms.

La adquisición de las señales EEG se hizo continuamente cuando los participantes estaban viendo las imágenes afectivas. La adquisición de datos siempre comenzó 5 minutos antes de la presentación del primer estímulo.

Capítulo 3. Preprocesado de datos a través de la aplicación EEGLAB de MATLAB

3.1 Flujo de trabajo

Este es el flujo de trabajo dónde se definen las partes que se harán en EEGLAB, en Matlab y en App Designer. Se puede ver en la **Figura 12:**

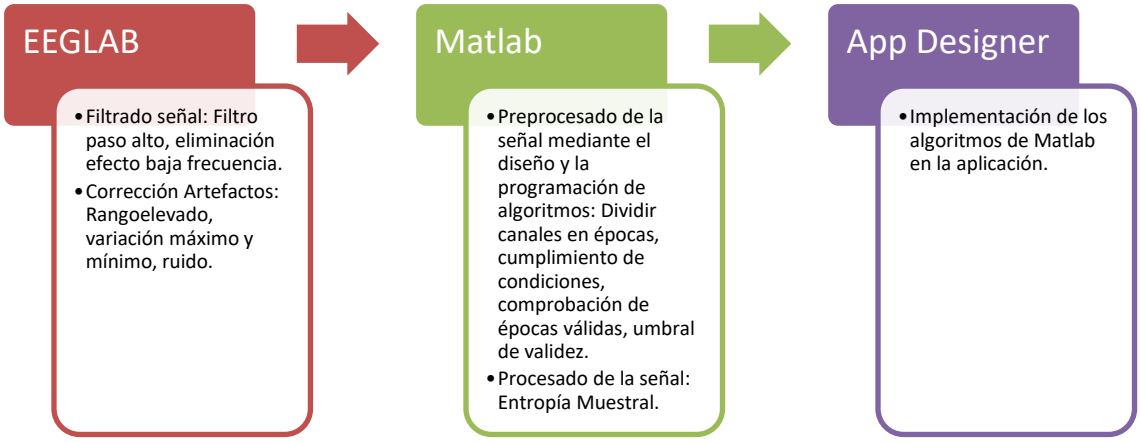


Figura 12: Flujo de trabajo EEGLAB, Matlab y App Designer.

3.2 Introducción a EEGLAB

EEGLAB es una caja de herramientas interactiva de Matlab en la cual se procesan los datos de la electroencefalografía (EEG), la magnetoencefalografía (MEG) y otras señales electrofisiológicas. Dicha herramienta implementa distintas funciones tales como el filtrado de los datos, rechazo de artefactos y varios modos de visualización de datos. Es una herramienta muy usada en el análisis de señales de EEG, pero no incluye ciertos algoritmos como los análisis de complejidad.

Cuando inicializas el programa, aparecen una serie de variables principales en el espacio de trabajo de Matlab (workspace), las cuales podemos ver a continuación en la **Figura 13**:

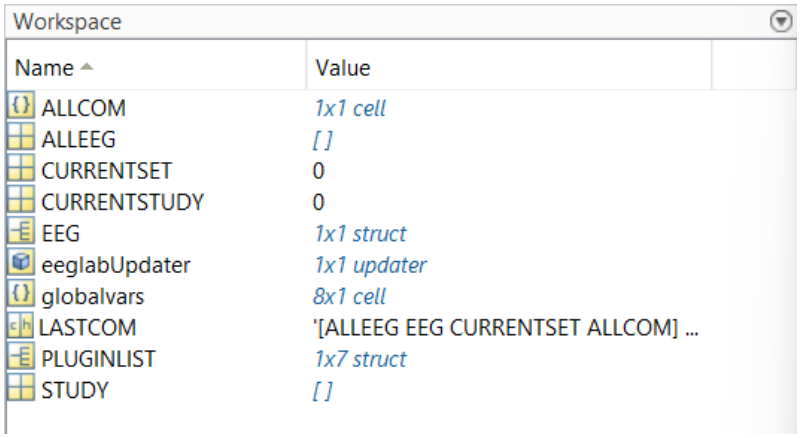


Figura 13: Variables EEGLAB en Workspace.

Las variables más importantes son las siguientes:

- **ALLCOM:** En esta variable se almacenan todos los comandos que son ejecutados desde el menú de EEGLAB.

- **ALLEEG:** Esta variable corresponde a la matriz de todos los conjuntos de datos cargados de EEG.

- **CURRENTSET:** Esta variable muestra el índice del conjunto de datos actual del EEGLAB. Es decir, si cargamos 3 bases de datos y estamos trabajando con la base de datos número dos, el índice que marcará esta variable será el "2".

- **EEG:** Aquí se indica el conjunto de datos de EEG actual. Es una gran estructura que contiene una gran cantidad de campos.

- **LASTCOM:** Se trata de una variable que nos muestra el último comando emitido desde el menú de EEGLAB.

- **STUDY:** Se trata de una variable que muestra la estructura del grupo de EEGLAB.

Una vez explicadas las distintas variables que aparecen en el workspace de Matlab tras la ejecución de EEGLAB, se explicará la interfaz gráfica de EEGLAB. A continuación, se muestra en la **Figura 14**:

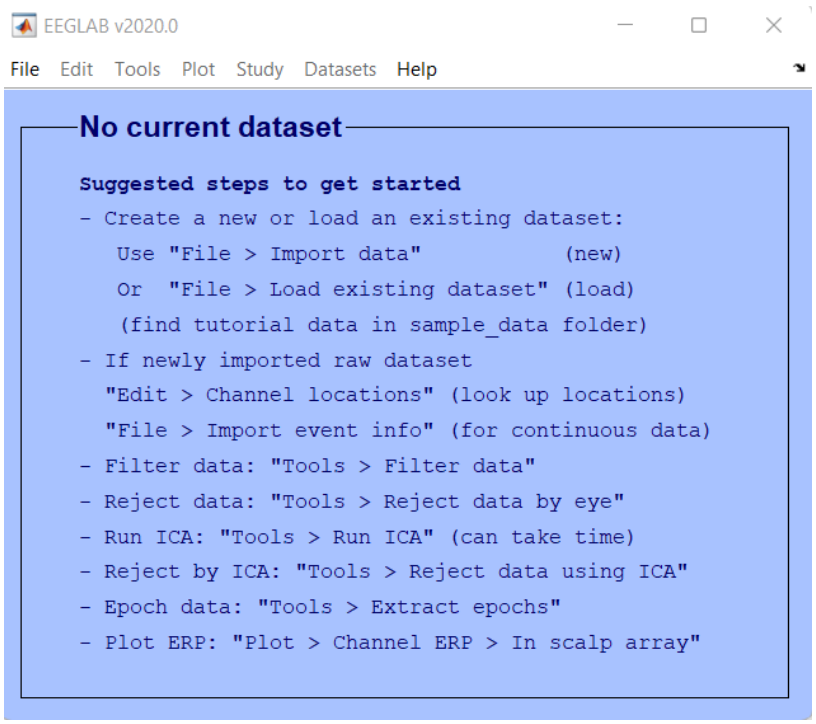


Figura 14: Interfaz Gráfica EGGLAB.

3.3 Preprocesado de datos

En primer lugar, se deben cargar los datos de la base de datos en EEGLAB. Se hará de la siguiente manera: ***File – Import data – Using EEG functions and plugins – From Brain Vis. Rec. .vhdr or .ahdr file.***

Se utiliza esta función porque los datos que estamos utilizando se registraron en este formato, en concreto Brain Amp. Hay otras opciones que podrían elegirse dependiendo del formato de los datos de entrada.

Una vez cargados los datos, como se puede apreciar en la **Figura 15**, podemos ver los principales atributos de la base de datos.

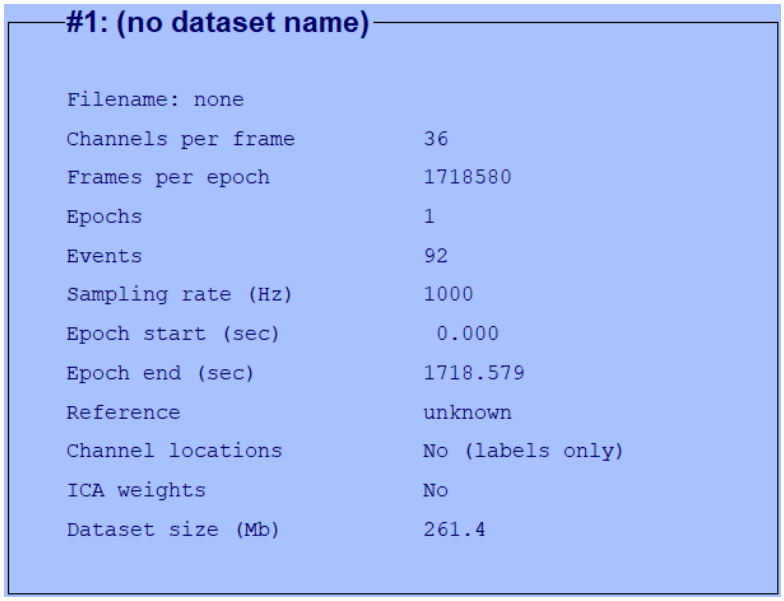


Figura 15: Atributos principales base de datos en EEGLAB

En nuestro se puede apreciar en primer lugar el número de canales que contiene nuestra base de datos, un total de 36 canales. Además, podemos ver como EEGLAB considera todos los datos como una única época que contiene un total de 1718580 frames.

Una época es un segmento de tiempo en que se divide la señal de EEG para realizar los análisis. Inicialmente, EEGLAB ha considerado toda la señal como una única época. Lo habitual será dividirla posteriormente en segmentos más cortos para realizar los análisis.

Es importante explicar, que los frames corresponde al número total de muestras que nuestro equipo ha registrado en esa época. El siguiente atributo contiene los registros de los eventos experimentales que han ocurrido mientras se registraban los datos, en nuestro caso 92 eventos. Por otro lado, tenemos la frecuencia de muestreo en 1000 Hz y, por último, dentro de lo más destacable tenemos el inicio y el final de la época. En nuestro caso tenemos una duración de 1718.579 segundos para la época.

Teniendo cargados los datos, ya podemos realizar las siguientes funciones que contiene la aplicación, para ver las características principales de la base de datos y realizar el preprocesado de los datos.

En primer lugar, nos encontramos con la pestaña ***Edit***, en la que se encuentran varias funciones interesantes:

- **Event Values:** En esta pestaña podemos encontrar cada uno de los eventos y sus principales características tales como: latencia, duración del evento, canal, tipo…

- **Channel Locations:** Este apartado es muy útil para ver la ubicación de los canales. Una vez dentro, se muestran las etiquetas del canal cargado y las coordenadas polares. Además, puedes visualizar las ubicaciones 2-D y 3-D de los canales, en el que se mostrarán los electrodos. En cualquier momento, se puede consultar un gráfico en el que solo muestre las ubicaciones de los canales deseados. En nuestro caso tenemos la siguiente figura, en los que se ven representados 32 de los 36 electrodos, ya que los 4 que no se muestran pertenecen a electrodos del corazón. Se puede apreciar en la **Figura 16:**

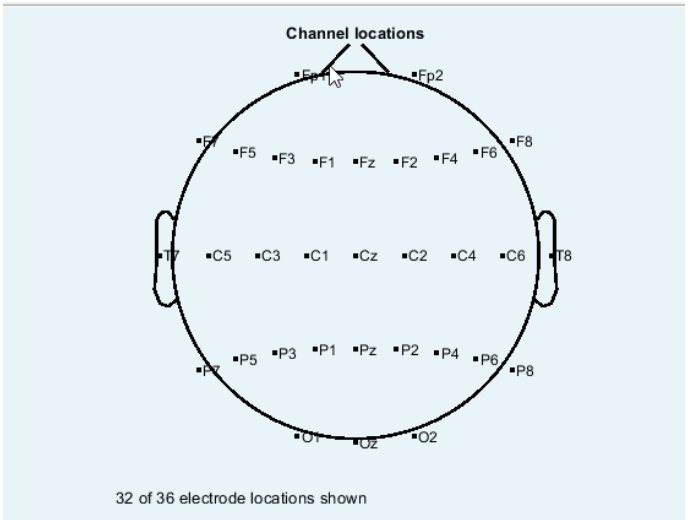


Figura 16: Channel Locations.

Tras ver las principales características de los datos, procedemos a realizar un primer preprocesado de la señal. Para ello, en primer lugar, eliminamos las tendencias lineales. Para ellos es necesario ir a ***Tools – Filter the data – Basic FIR Filter.*** Tras esto se realiza un primer filtrado paso alto con frecuencia de corte (fc) a 1Hz, a partir de aquí se irá modificando la fc de paso alto. Como se veía en la **Tabla 2**, la señales de EEG empezaban en 1Hz, y por ello, el filtrado estará por debajo de 1 Hz. Se puede ver este filtrado en la **Figura 17:**

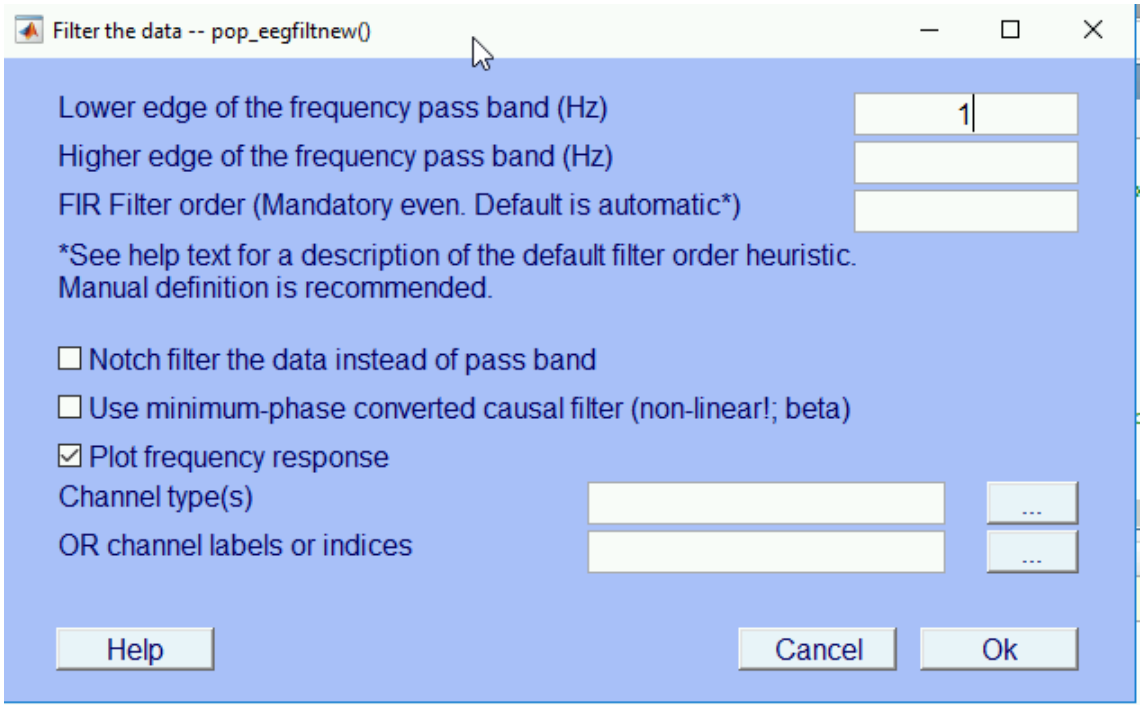


Figura 17: Filtrado de los datos.

Tras esto, queremos rechazar los artefactos y en primer lugar se procede a eliminar los canales defectuosos. Para ello, vamos a la pestaña ***Plot,*** para ver el resultado del filtrado y poder ver la señal de todos los canales.

- **Channel Data (scroll):** Aquí se utiliza para desplazarse por los datos del canal del conjunto de datos actual. Podemos ver la amplitud de la barra de escala vertical, además del rango, el número de canales a mostrar. También son muy útiles para ver el canal con mucho más detalle, las funciones de zoom o la función grid para mostrar líneas de cuadricula tanto horizontales como verticales. Se obtiene la gráfica de la **Figura 18:**

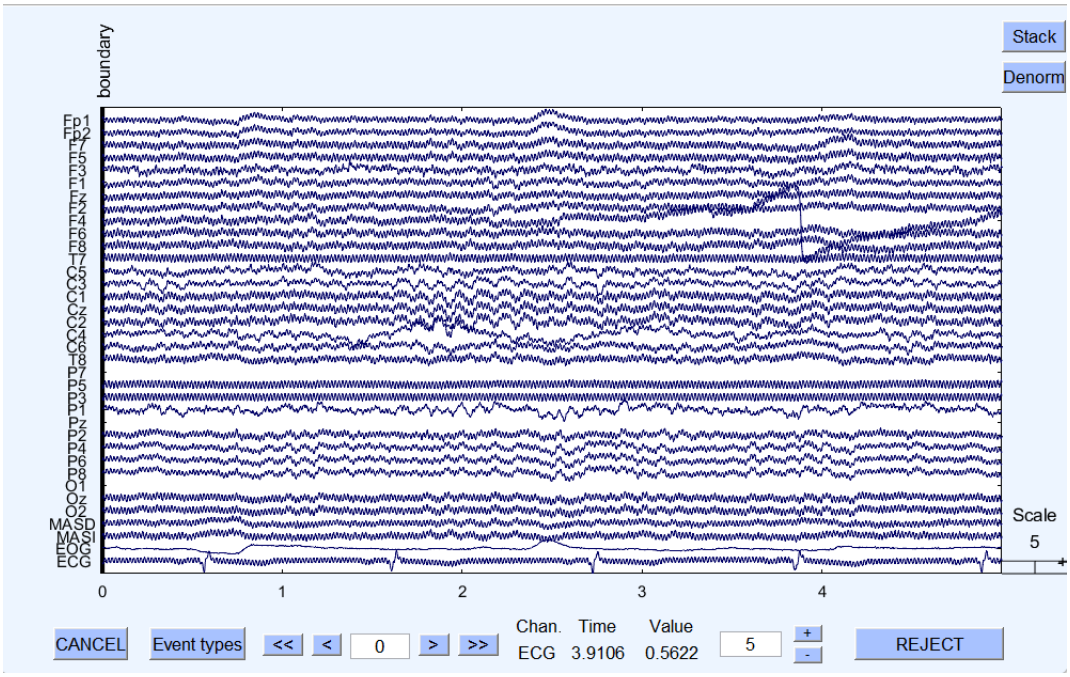


Figura 18: Gráfica canales después filtrados (1Hz)

Si presionamos el botón Stack, se pueden apilar los datos, y es más sencillo ver los datos anómalos que aparecen. Se puede comprobar en la **Figura 19:**

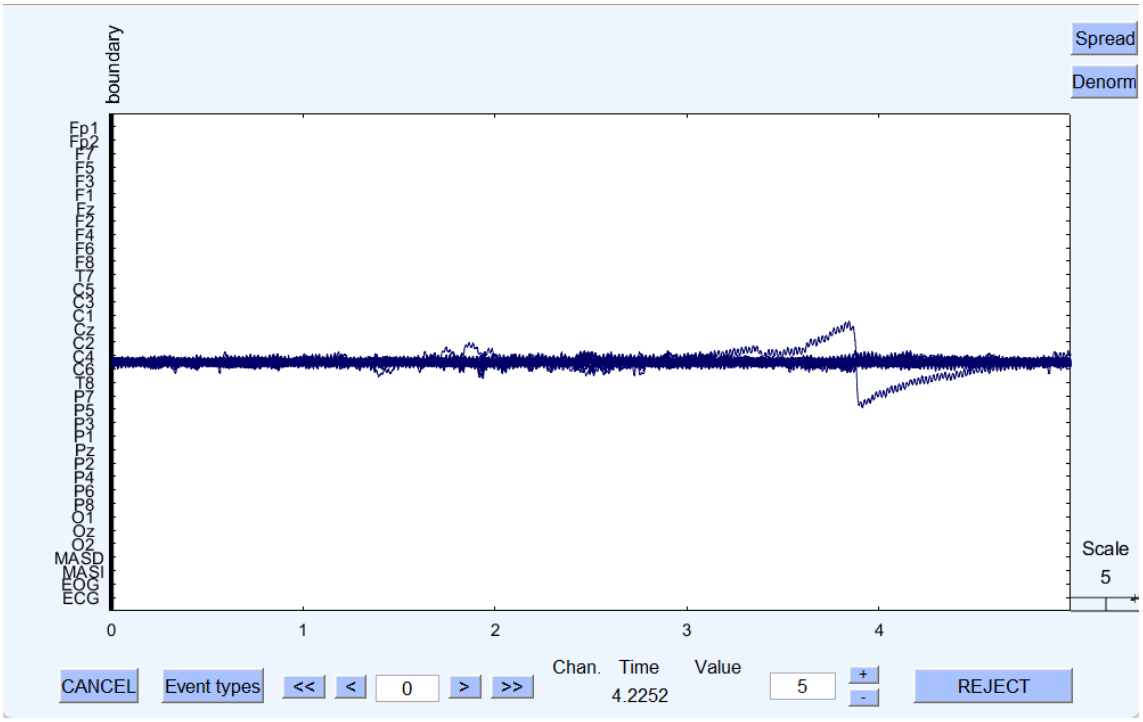


Figura 19: Gráfica con los datos apilados (Stack).

También hay otra función que puede ayudar a ver mejor los canales defectuosos:

- **Channel spectra and maps:** En esta gráfica podemos ver el espectro de la actividad de cada uno de los canales, éste es representado por un color distinto. Además, dentro de la gráfica se puede ver la distribución de energía que sigue el cuero cabelludo. En nuestro caso tenemos la **Figura 20**:

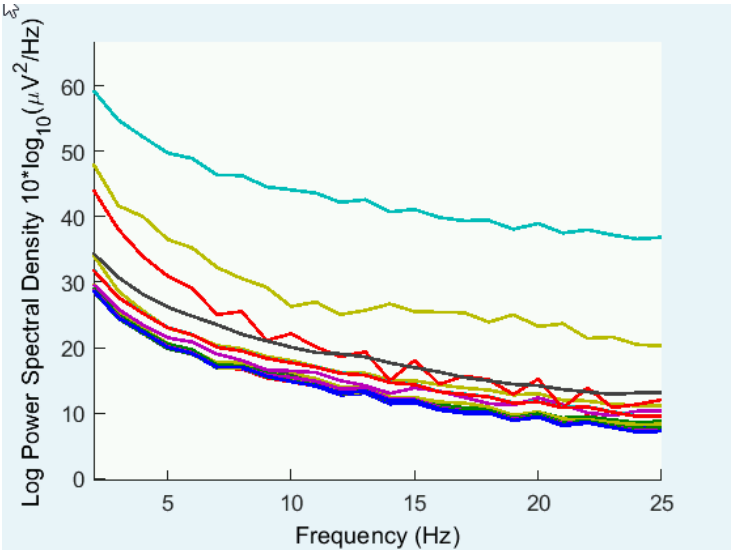


Figura 20: Gráfica del espectro de las señales.

Tras esto se procede a eliminar los canales que nos son válidos para nuestro estudio. En primer lugar, eliminamos los últimos 4 canales, ya que pertenecen al corazón y no a nuestro estudio del cerebro, y los canales que hemos detectado que han sido defectuosos tras el primer filtrado. Las señales pueden ser defectuosas por diversas razones, como por ejemplo señales con un rango muy elevado, mucha variación entre máximo y mínimo, además debido a una mala conexión de los electrodos puede llevar a una mala adquisición debido al ruido.

En nuestro caso, las gráficas anteriores solo hemos captado 4 segundos, pero analizando las señales más allá de los 4 segundos, se ha comprobado que hay señales que también tienen el mismo comportamiento y han sido eliminadas.

Se haría de la siguiente manera: ***Edit – Select Data*** y en el apartado de Channel range pondremos los canales que deseamos eliminar de nuestro estudio y seleccionaremos la casilla de remove. Se puede ver en la **Figura 21:**

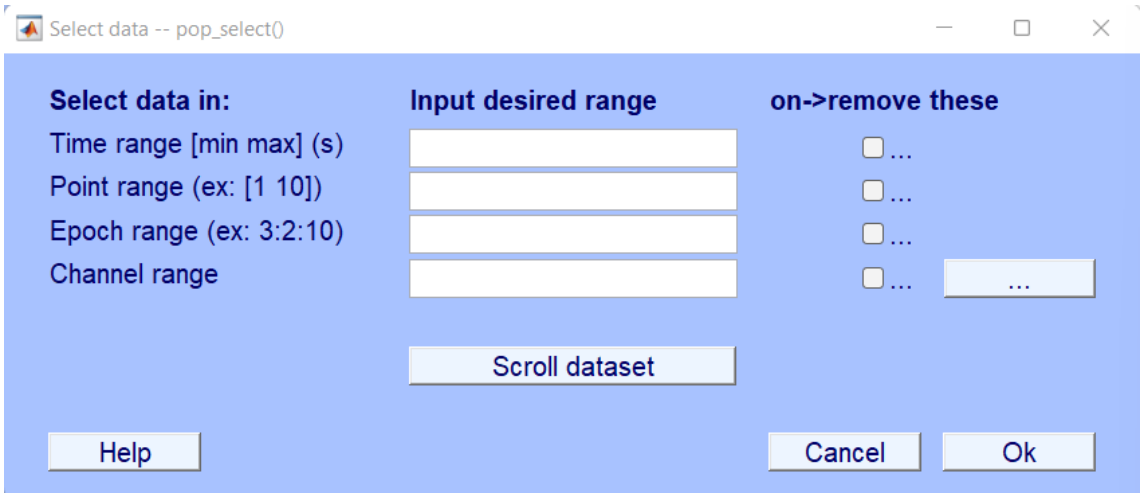


Figura 21: Borrar canales defectuosos.

Tras eliminar los canales defectuosos que hemos podido eliminar en el primer filtrado, procedemos a realizar un segundo filtrado esta vez a 0,5Hz, 0,4Hz y 0,3Hz. El objetivo de este filtrado es eliminar las caídas debido al efecto de la baja frecuencia. Tras esto, obtenemos las siguientes gráficas de las señales ya filtradas en la **Figura 22, Figura 23 y Figura 24**:

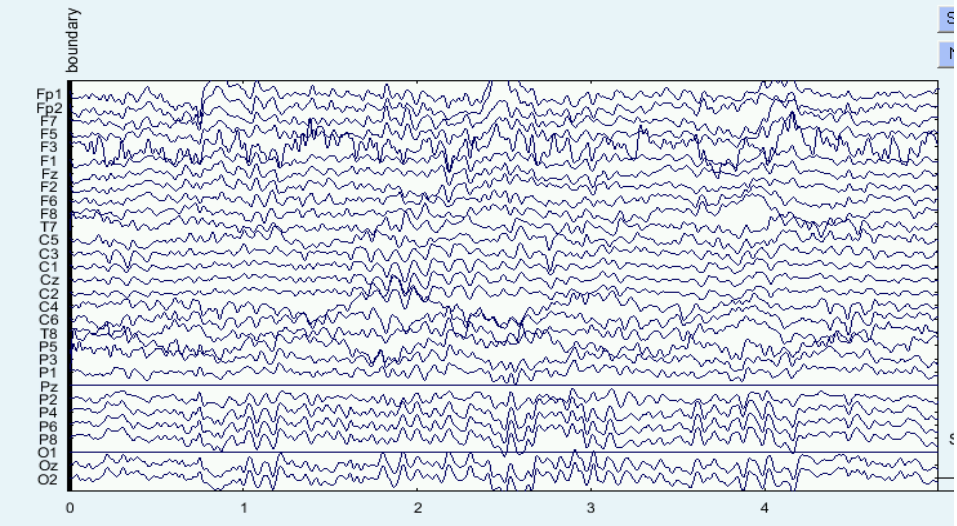


Figura 22: Filtrado de las señales a 0,5Hz.

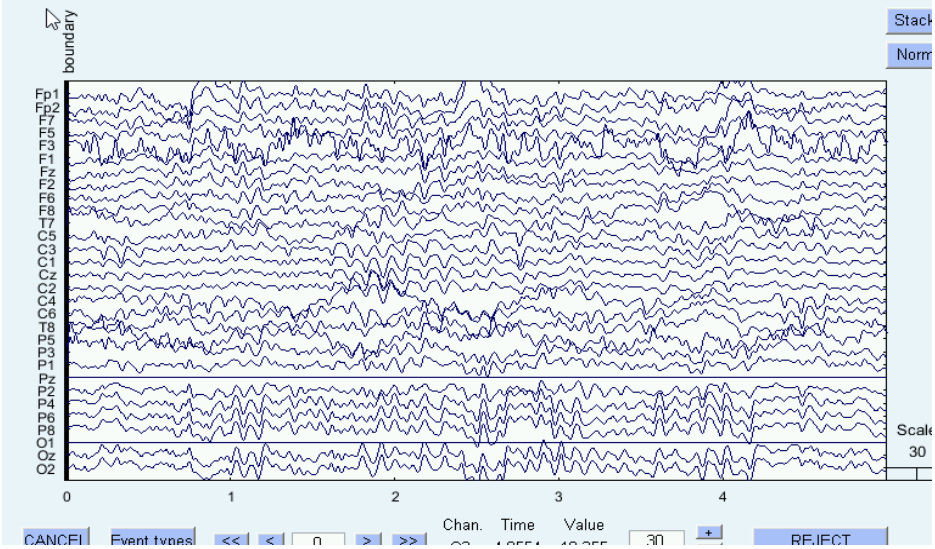


Figura 23: Filtrado de las señales a 0,4Hz.

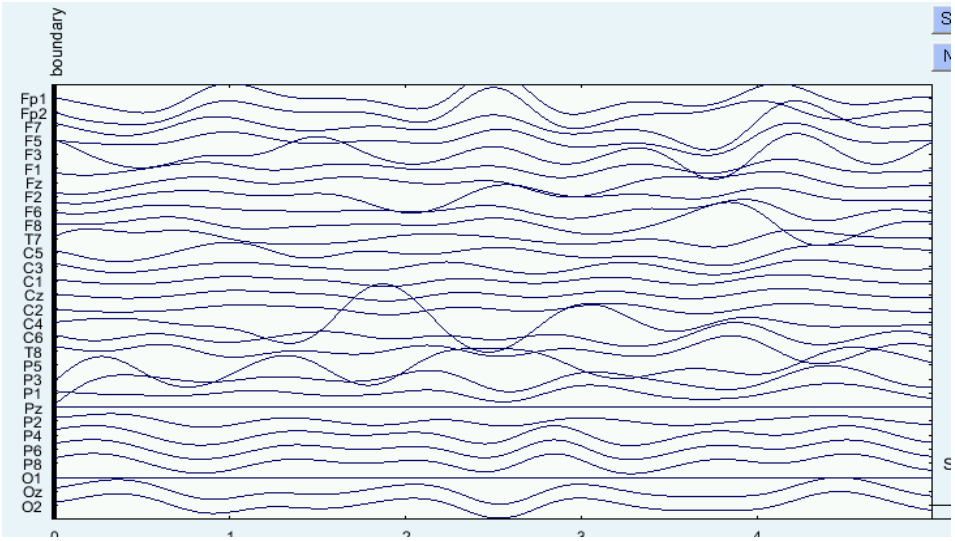


Figura 24: Filtrado de la señal a 0,3Hz.

Tras comprobar los siguientes filtrados, se puede ver en la **Figura 24,** que los componentes de baja frecuencia hacen que las señales que no pertenece a EEG, enmascaran la señal de EEG. Por lo que podemos ver que es suficiente con un filtrado paso alto de 0,5Hz para eliminar el efecto de la baja frecuencia

Solo se ha representado una parte de la señal, por lo que se han eliminado algunas señales, que se van mucho del rango de voltajes que pertenecen a una señal de EEG. Por tanto, tras eliminar los canales defectuosos después del último filtrado, quedarán los siguientes canales para proceder a la siguiente parte del estudio. Se puede apreciar en la **Figura 25:**

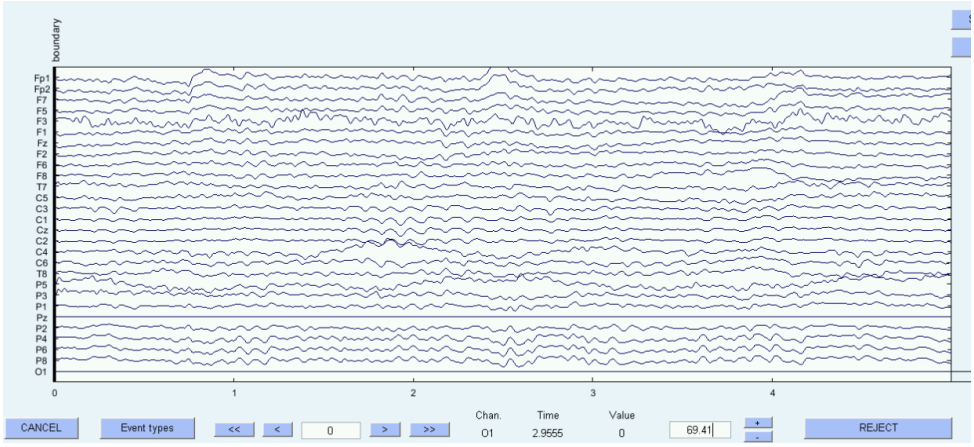


Figura 25: Filtrado de la señal completa.

3.4 Extracción de época de datos

Otras de las funciones de las que dispone EEGLAB es la extracción de época de datos, para ellos tendremos que ir a la pestaña de ***Tools – Extract Epochs***. Se puede ver en la **Figura 26:**

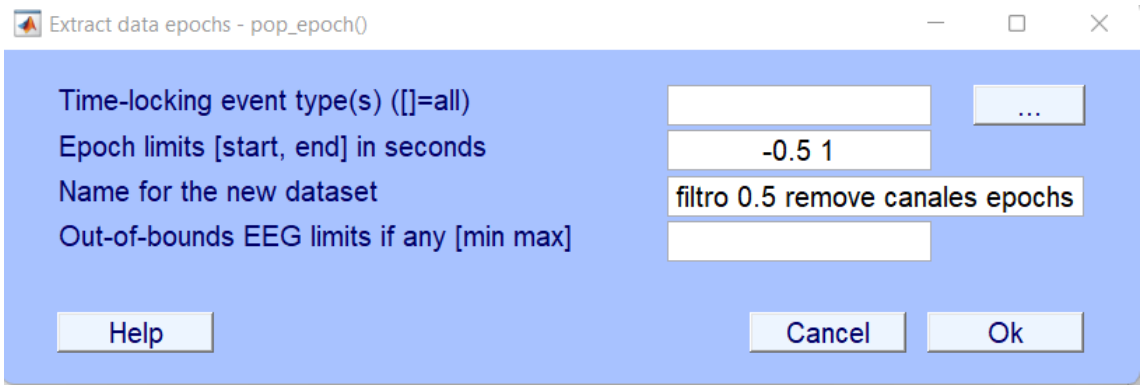


Figura 26: Extraer épocas.

Como se puede ver en la figura se puede determinar los límites de las épocas en segundos, en nuestro caso extraeremos las épocas de -500ms a 1000ms. Y tras esto, podemos hacer un análisis por épocas de las señales, que nos permita detectar mejor los artefactos que pueden tener determinados canales. Se puede apreciar en la **Figura 27:**

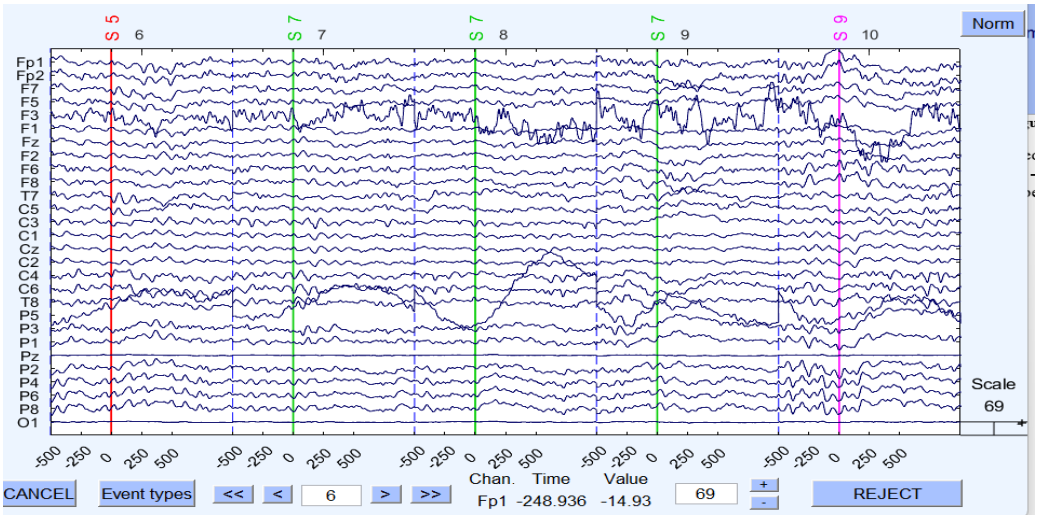


Figura 27: Señales por épocas.

Capítulo 4. Diseño y programación de algoritmos en Matlab para caracterizar de forma automática las señales de EEG desde el punto de vista de su complejidad

4.1 Implementación de algoritmos en Matlab para el preprocesado de la señal

4.1.1 Flujo de trabajo

El flujo de trabajo será el que se puede ver en la **Figura 28.** En primer lugar, se implementará la función matrizceros donde se analizará la validez de todos los datos de las señales, tras esto se generará una función generamarcas que mostrará el tiempo inicial y final de las épocas de la señal, y finalmente habrá una función matrizceros_marcas que validará época por época cada una de las señales. Finalmente, se pasará al cálculo de la entropía muestral gracias al algoritmo SampEn y a la implementación de la función entropía.

Figura 28: Flujo de trabajo,

4.1.2 Función Matrizceros

Tras realizar el primer preprocesado de datos en la aplicación de EEGLAB, procedemos a realizar el diseño y programación de algoritmos en Matlab.

En primer lugar, realizaremos la primera función, en la que tendremos los siguientes objetivos:

1. Se dividirán los canales en conjunto de épocas de 1 segundo. Para ello, se obtendrá la longitud de la señal y se dividirá por la frecuencia de muestreo (1000Hz) y ya tendremos en segundos la señal. Así obtendremos épocas de un segundo en cada señal.

2. Luego tendremos que cumplir las siguientes condiciones [45]:

    - Tensión mínima (Vmin) > -100 uV.
    - Tensión máxima (Vmax) < 100 uV.
    - Abs (V(n+1) – V(n)) < 100 uV.
    - Abs (Vmax – Vmin) < 100 uV.

3. Crearemos una matriz de unos y ceros, en la que se marcará como uno la posición correspondiente a aquellas épocas que cumplan las cuatro condiciones anteriores. En caso de que se incumpla una de las cuatro, la matriz será rellenada con un cero.

4. Por último, haremos un vector de porcentajes de cada señal. Para ello, se calculará la cantidad de épocas válidas que contiene la señal. Si este porcentaje es menor a un 50%, descartaremos la señal y la pondremos toda a cero.

Para realizar todo este proceso, se ha implementado una función en Matlab, llamada **matrizceros**, que vemos a continuación:

1. En primer lugar, tenemos cuatro parámetros de entrada y dos parámetros de salida:

    i. Parámetros de entrada:
        1. **Frec_muestreo:** Frecuencia de muestreo.
        2. **Tiempoepoca:** Tiempo de la época.
        3. **Matrizsenyal:** Señales a utilizar.
        4. **Numerocanales:** Número de canales.

    ii. Parámetros de salida:
        1. **Vectorzeros:** Épocas válidas.
        2. **Vectorporcentajes:** Porcentaje épocas válidas.

La frecuencia de muestreo que se ha utilizado en el registro de EEG que se está utilizando para la validación ha sido 1000Hz.

En cuanto a la matrizsenyal, como vimos en la **Figura 13**, dentro de los campos que contiene EEG, tenemos un campo denominado ***data***, el cual contiene todos los datos de los 28 canales. Por lo que, en este parámetro de entrada, incluiremos esta variable. Se ve en la **Figura 29:**

Figura 29: Campo data EEG.

En el caso del número de canales, escogeremos el campo denominado ***nbchan***, el cual nos indica el número de canales que contiene nuestra base de datos actualmente. Se ve en la **Figura 30:**

Figura 30: Campo nbchan EEG.

Con la función definida, pasamos a implementar la función, en este caso, el primer paso es la creación de una serie de variables. Como hemos dicho anteriormente, en primer lugar, dividiremos los canales en épocas de un segundo y, además, se inicializa el vector de ceros, que será el que se rellenará a medida que se comprueben las condiciones. En la **Figura 31** se comprueba el paso realizado:

```
function [vectorzeros, vectorporcentajes] = matrizceros (frec_muestreo, tiempoepoca, matrizsenyal, numerocanales)
%% Variables
l = length (matrizsenyal); %Longitud total de la señal
tiempo = l*1/frec_muestreo; % Longitud en tiempo
muestrasepoca = floor (tiempoepoca*frec_muestreo);
epocas = floor (l/muestrasepoca);
vectorzeros = zeros(epocas,numerocanales);

```

Figura 31: Implementación Programa Matrizceros (1).

2. Tras esto, ya pasaremos a analizar segundo a segundo las épocas para determinar si se cumplen las condiciones deseadas. Para ello se realiza un bucle, en el que entraremos en la primera columna (primer canal) e iremos recorriendo fila a fila (épocas) analizando cada una de las épocas con las condiciones a cumplir.

   Para ello, en primer lugar, se declara la variable vector. En esta variable se llama a la primera señal y se va actualizando muestra a muestra, cada vez que entra en el bucle.

   Por otro lado, se declara la variable vectordesplazado, en la cual se desplaza n+1 posiciones la variable vector.

   Tras esto, se declara una variable auxiliar, en la que damos por hecho que se cumplen cada una de las condiciones, y una vez, entra en cada una de las condiciones, si no cumple una de ellas, esta variable auxiliar pasará directamente a tener el valor de cero.

Se puede ver la implementación del código en la **Figura 32 Figura 33 y Figura 34:**

```
%% Bucles de análisis de cada segundo
 for i = 1:size(vectorzeros,2)
        x = matrizsenyal (i,:); %Mostramos la primera señal representada
        t = (0:(l-1))*1/frec_muestreo; %Pasamos las muestras a tiempo en el eje x
        plot (t, matrizsenyal (i,:), 'r'); %Mostramos la señal representada en tiempo
        p = 1; %Inicializamos valor vector inicial

        for h = 1:size(vectorzeros, 1) %filas
                vector = x (p:p+(muestrasepoca-1));
                vectordesplazado = zeros(size(vector));
                vectordesplazado (2:end) = vector(1:end-1);
                p = p+muestrasepoca;
                aux = 1;

```

Figura 32: Implementación Programa Matrizceros (2).

```
            if min (vector)>-100
                    disp ('Es correcto')
                else
                    disp ('No es correcto, ya que el mínimo es mayor que -100uV')
                    aux = 0;

                end

                    if max (vector)<100
                        disp ('Es correcto')
                    else
                        disp ('No es correcto, ya que el máximo es menor que 100uV')
                        aux = 0;
                    end

```

Figura 33: Implementación Programa Matrizceros (3).

```
                if abs (max(vector)-min(vector)) <100
                    disp ('Es correcto')
                else
                    disp ('No es correcto, ya que el absoluto de mayor menos el menor es menor que 100uV')
                    aux = 0;
                end
                if abs (vectordesplazado-vector)<100
                    disp ('Es correcto')
                else
                    disp('No es correcto, ya que el absoluto de una muestra con la anterior es mayor que 100uV')
                    aux = 0;
                end
                    vectorzeros(h,i) = aux;

        end
```

Figura 34: Implementación Programa Matrizceros (4).

3. Por último, dentro de la función se implementa, el vector de porcentajes, en el que se comprobará el porcentaje total de épocas que cumplen cada una de las condiciones.

   Para ello, se realiza un bucle, en el que se recorrerá toda la columna de la matriz de ceros y unos. Tras esto, mediante la función ***find*** de Matlab, encontraremos las posiciones donde se encuentran todos los unos, y con la función ***numel*** se cuenta el número total de unos que se han encontrado. Es decir, si hay un total de 600 unos, la función ***find*** detecta 600 filas con unos y la función ***numel***, directamente proporciona el número 600.

   Tras esto, se calcula el umbral, que será la cantidad de unos entre el total de épocas y si el umbral es mayor a 50, se quedará toda la señal igual, y si es menor al 50%, la señal la descartaremos y la pondremos toda a cero.

   Se puede ver en la **Figura 35:**

```
53          for k = 1:size(vectorzeros,2)
54              numeros = numel (find(vectorzeros(:,k)==1));
55              umbral = (numeros/epocas)*100;
56              vectorporcentajes (k) = umbral;
57              if umbral<50
58                 vectorzeros (:,k) = zeros (size(vectorzeros(:,k)))
59              else
60                 disp ('Sigue igual')
61              end
62          end
63  end
```

Figura 35: Implementación Programa Matrizceros (5).

Con esto la función matrizceros quedaría completa, y ahora pasamos a obtener los resultados. En este caso, obtendríamos una matriz de porcentajes como se ve a continuación en la **Figura 36:**

1x28 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	71.8859	73.4575	87.7183	90.9779	0	93.4226	93.5390	93.4808	91.3271	91.4435	84.4587	91.8510	92.6077	94.2375	94.2375	94.2375	90.6868	91.9092	86.7870	60.7683	71.7113
2																					
3																					
4																					
5																					

Figura 36: Vector Porcentajes.

La otra matriz que se obtiene es la matriz de ceros y como se puede comprobar, el porcentaje de la señal 5 es del 0%, por lo que la señal 5 es rellenada completamente de ceros, como se ve en la **Figura 37:**

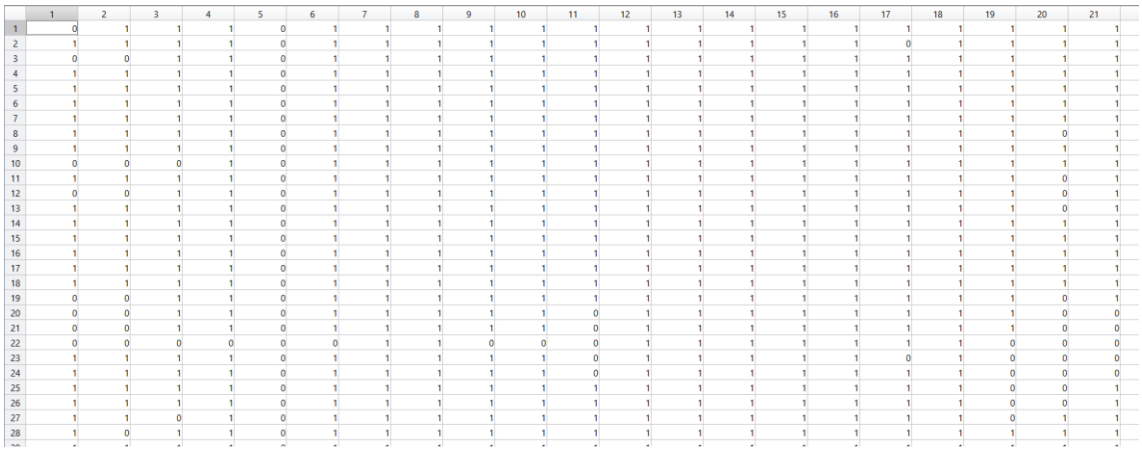


Figura 37: Vector Matrizceros.

4.1.3 Función Generamarcas

Tras esto, se procede a realizar otra función en la que tendrá los siguientes objetivos:

1. Como se ha indicado en la estructura de los datos, la frecuencia de muestreo es de 1000Hz, y va de −100 ms a 900 ms en relación con el inicio del estímulo. Por lo tanto, generaremos una función que tendremos como tiempo de inicio de la muestra -0,1s y tiempo después de la muestra de 0,9s. Cuando se realiza el registro de EEG en el marco de una experimentación psicofisiológica, la persona se ve expuesta a determinados estímulos durante la sesión (visuales, auditivos, emocionales…), y resulta de interés analizar la señal de EEG en el momento temporal que se percibe el estímulo. Es por eso por lo que se ha implementado esta función para analizar los datos.

2. Tras esto realizaremos un vector en el que marcaremos la latencia de cada uno de los eventos de la señal, en la que se marcará época a época con una diferencia de 1000 hz.

3. Tras esto, indicaremos el tipo de evento con el que estamos trabajando.

Para este proceso, se ha implementado la función llamada **generamarcas** que vemos a continuación:

1. En primer lugar, tenemos cuatro parámetros de entrada y un parámetro de salida:

    i. Parámetros de entrada:
        1. **Eventos:** Tiempo inicial, tiempo final y tipo de cada época.
        2. **Frec_muestreo:** Frecuencia de muestreo.
        3. **T_antes_muestra:** Tiempo antes de la muestra.
        4. **T_desp_muestra:** Tiempo después de la muestra.

    ii. Parámetros de salida:
        1. **Vectormarca:** Tiempo inicial, tiempo final y tipo de cada época después de aplicar los parámetros de entrada 2, 3 y 4.

    En cuanto a los eventos, igual que hemos hecho en procesos anteriores, hacemos referencia a unos de las variables que contiene el campo EEG. En esta variable tenemos el inicio y el final de cada una de las épocas y el tipo de evento que es. Como se puede ver a continuación la variable es la de la **Figura 38:**

Figura 38: Campo event EEG.

Los otros parámetros de entradas, ya descritos anteriormente son: la frecuencia de muestreo de 1000Hz, el tiempo antes de la muestra que es -0,1 segundos y el tiempo después de la muestra que es 0,9 segundos.

Mientras que el parámetro de salida, el ***vectormarca,*** definirá el tiempo inicial de la época y el tiempo final de la época después de aplicarle la frecuencia de muestreo y el tiempo antes y después de la muestra.

Con la función definida, pasamos a implementar la función, en este caso. El primer paso es la creación de una serie de variables. En las variables se generará un vector de ceros como vectormarca, que tendrá como número de filas el tamaño de todos los eventos que contiene las señales. Y el número de columnas que será de tres: el inicio del evento, el final del evento y el tipo del evento. Se puede ver en la **Figura 39**:

```
1  function [vectormarca] = generamarcas (eventos, frec_muestreo, t_antes_muestra, t_desp_muestra)
2  %% Variables
3  numerofilas = size (eventos,2);
4  numerocolumnas = 3;
5  vectormarca = zeros (numerofilas,numerocolumnas);
```

Figura 39: Implementación Programa Generamarcas (1).

2. Tras esto, se procede a implementar el bucle en el cual se irá marcando los eventos en función de la latencia. Para ello, definimos una variable que se le llamará latencia, en la cual hará referencia a cada una de las latencias que están dentro del campo eventos del EEG.

   Tras esto iremos rellenando el vectormarca de la siguiente manera: la latencia que hemos guardado en la variable latencia le restaremos el tiempo antes de la muestra multiplicado por la frecuencia de muestreo y obtendremos el inicio de la época. De la misma manera rellenaremos la segunda columna de la matriz, pero sumándole a la latencia, el tiempo después de la muestra porla frecuencia de muestreo. Como se puede ver a continuación en la **Figura 40:**

```
6  %% Bucle
7  for i = 1:size(vectormarca,1)
8      latencia = eventos (i).latency;
9      vectormarca (i,1) = latencia - round(t_antes_muestra*frec_muestreo);
10     vectormarca (i,2) = latencia + round(t_desp_muestra*frec_muestreo);
11
```

Figura 40:Implementación Programa Generamarcas (2).

Tras esto, simplemente nos queda comprobar de que tipo de evento estamos hablando y rellenar la tercera columna con el tipo de evento correspondiente, se haría de la manera que se puede ver en la **Figura 41:**

```
12      str = 'boundary';
13      str1 = 'S 15';
14      str2 = 'S  5';
15      str3 = 'S  7';
16      str4 = 'S  9';
17      type = eventos (i).type;
18
19      if (strcmp (type, str))
20          vectormarca (i,3) = 1;
21      else
22          disp ('No es este tipo');
23      end
24
25      if (strcmp (type, str1))
26          vectormarca (i,3) = 2;
27      else
28          disp ('No es este tipo');
29      end
```

Figura 41: Implementación Programa Generamarcas (3).

Como se puede ver en la imagen anterior, definimos cinco variables strings con el tipo de eventos que tenemos. Después, declaramos una variable llamada type, en el que se llama a la variable type dentro del campo eventos del campo de EEG y tras esto, comparamos mediante la función ***strcmp*** para ver de qué tipo es. Y cuando sea el correcto, se añadirá a la tercera columna de la matriz vectormarca. Se puede ver en la **Figura 42:**

```
30
31      if (strcmp (type, str2))
32          vectormarca (i,3) = 3;
33      else
34          disp ('No es este tipo');
35      end
36
37      if (strcmp (type, str3))
38          vectormarca (i,3) = 4;
39      else
40          disp ('No es este tipo');
41      end
42
43      if (strcmp (type, str4))
44          vectormarca (i,3) = 5;
45      else
46         disp ('No es este tipo');
47      end
48      vectormarca (1,1) = 1; %Esto lo hacemos para que el índice inicial no sea negativo.
49
50  end
```

Figura 42: Implementación Función Generamarcas (4).

Además, cabe destacar en la línea 48 del programa, que se ha declarado el índice inicial como 1 para que no sea negativo.

4.1.4 Función Matrizceros_marcas

Una vez, tenemos la matriz ***generamarcas***, donde tenemos cada una de las muestras separadas época a época, con su tiempo inicial y final, se procede a generar otra función que tendrá los siguientes objetivos:

1. Se dividirán los canales en muestras de 1 segundo. Para ello, se obtendrá la longitud de la señal y se dividirá por la frecuencia de muestreo (1000Hz) y ya tendremos en segundos la señal. Así obtendremos épocas de un segundo en cada señal.

2. Luego tendremos que cumplir las siguientes condiciones:

    - Tensión mínima (Vmin) > -100 uV.
    - Tensión máxima (Vmax) < 100 uV.
    - Abs (V(n+1) – V(n)) < 100 uV.
    - Abs (Vmax – Vmin) < 100 uV.

3. Crearemos una matriz de unos y ceros, en la que se marcará como uno aquellas épocas que cumplan las cuatro condiciones anteriores, en caso de que se incumpla una de las cuatro, la matriz será rellenada con un cero.

4. Por último, haremos un vector de porcentajes de cada señal. Por lo que, se calculará la cantidad de épocas válidas que contiene la señal. Si este porcentaje es menor a un 50%, descartaremos la señal y la pondremos toda a cero.

Como se puede comprobar, son las mismas condiciones que la primera función creada, la diferencia que tendremos, es que en este caso vamos a utilizar las marcas creadas en la función anterior, en el que iremos analizando época a época, con muestras de 1000 en 1000 Hz.

Para ello, se implementa la función ***matrizceros_marcas***, que vemos a continuación:

1. En primer lugar, tenemos cuatro parámetros de entrada y un parámetro de salida:

    i. Parámetros de entrada:
        1. **Frec_muestreo:** Frecuencia de muestreo.
        2. **Marcas:** Matriz vectormarca.
        3. **Matrizsenyal** Señales a utilizar
        4. **Numerocanales:** Número de canales.

    ii. Parámetros de salida:
        1. **Vectorzeros:** Épocas válidas.
        2. **Vectorporcentajes:** Porcentaje épocas válidas.

    Dentro de los parámetros de entrada, la única diferencia con la primera función es que escogeremos la matriz ***vectormarca*** creado anteriormente con el tiempo de época inicial y el final, en lugar de escoger todos los datos.

    Y se inicializan una serie de variables que consiste en transformar la longitud en tiempo y definir el vector de ceros y unos. Se puede ver en la **Figura 43**:

```
function [vectorzeros, vectorporcentajes] = matrizceros_marcas (frec_muestreo, marcas, matrizsenyal, numerocanales)
%% Variables
l = length (matrizsenyal); %Longitud total de la señal
tiempo = l*1/frec_muestreo; % Longitud en tiempo
epocas = size(marcas,1);
vectorzeros = zeros(epocas,numerocanales);

```

Figura 43: Implementación Función Matrizceros_marcas (1).

Tras esto, se procede a implementar el bucle de análisis de cada segundo de la señal, en la que contiene las mismas partes que la primera función. La única diferencia es que ahora analizaremos las condiciones dentro de cada una de las marcas que hemos creado anteriormente. Quedaría como la **Figura 44 y Figura 45**:

```
%% Bucles de análisis de cada segundo
 for i = 1:size(vectorzeros,2)
        x = matrizsenyal (i,:); %Mostramos la primera señal representada
        t = (0:(l-1))*1/frec_muestreo; %Pasamos las muestras a tiempo en el eje x
        plot (t, matrizsenyal (i,:), 'r'); %Mostramos la señal representada en tiempo
        p = 1; %Inicializamos valor vector inicial

        for h = 1:size(vectorzeros, 1) %filas
                vector = x (marcas(h,1):marcas(h,2));
                vectordesplazado = zeros(size(vector));
                vectordesplazado (2:end) = vector(1:end-1);
                aux = 1;

```

Figura 44: Implementación Función Matrizcero_marcas (2).

```
                if min (vector)>-100
                        disp ('Es correcto')
                    else
                        disp ('No es correcto, ya que el mínimo es mayor que -100uV')
                        aux = 0;

                end

                    if max (vector)<100
                        disp ('Es correcto')
                    else
                        disp ('No es correcto, ya que el máximo es menor que 100uV')
                        aux = 0;
                    end

                    if abs (max(vector)-min(vector)) <100
                        disp ('Es correcto')
                    else
                        disp ('No es correcto, ya que el absoluto de mayor menos el menor es menor que 100uV')
                        aux = 0;
                    end
                    if abs (vectordesplazado-vector)<100
                        disp ('Es correcto')
                    else
                        disp('No es correcto, ya que el absoluto de una muestra con la anterior es mayor que 100uV')
                        aux = 0;
                    end
                        vectorzeros(h,i) = aux;

```

Figura 45: Implementación Función Matrizcero_marcas (3).

Por último, se implementa la función de porcentajes, con el mismo umbral de la primera función. Si el número de épocas no supera un 50% de veces las condiciones, descartaremos la señal y se pondrá toda a ceros. Se puede ver en la **Figura 46:**

```
        end
        for k = 1:size(vectorzeros,2)
            numeros = numel (find(vectorzeros(:,k)==1));
            umbral = (numeros/epocas)*100;
            vectorporcentajes (k) = umbral;
            if umbral<50
               vectorzeros (:,k) = zeros (size(vectorzeros(:,k)))
            else
                disp ('Sigue igual')
            end
        end
end
```

Figura 46: Implementación Función Matrizceros_marcas (4).

4.2 Implementación algoritmo de la entropía muestral

La entropía muestral se puede medir mediante el algoritmo ***SampEn***, Esta consiste en la estimación de la probabilidad condicional de que las épocas, con una longitud *m*, que coinciden puntualmente dentro de una tolerancia *r*, coincidan también en el siguiente punto. Como se puede apreciar en la **Figura 47:**

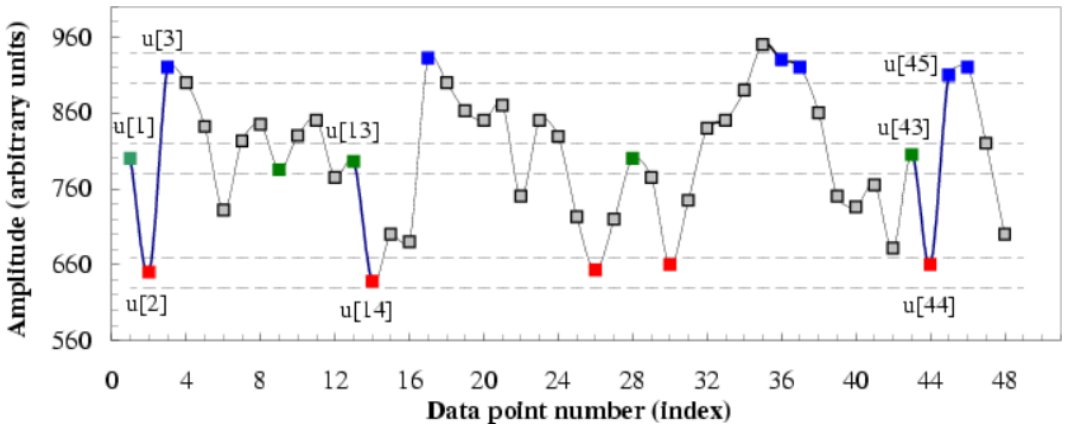


Figura 47: Entropía de la muestra [44].

4.2.1 Función SamEn

Para ello, el algoritmo de ***SampEn***, que no ha sido programado por mí, sigue una serie de procedimientos que vemos a continuación:

1. En primer lugar, tenemos tres parámetros de entrada y tres parámetros de salida:

    i. Parámetros de entrada:
        1. ***Y***: Los datos de entrada.
        2. ***M***: Longitud máxima especificada.
        3. ***R***: Tolerancia máxima.

    ii. Parámetros de salida:
        1. ***e:*** Estimaciones de entropía de muestra para m=0,1, …, M-1.
        2. ***A:*** Número de coincidencias para m = 1, …, M.
        3. ***B:*** Número de coincidencias para m = 0, …, M-1, excluyendo el último punto.

Con esto quedaría inicializada la función, y además se inicializan las siguientes variables:

- ***N:*** Longitud de los datos introducidos.
- ***P:*** Corresponde la cantidad de número de coincidencias de A dividida entre el número de coincidencias en B.
- ***Y:*** Corresponde a los datos introducidos menos la media de los datos dividido por la raíz cuadrada de la media de los datos al cuadrado.

La primera parte de la función se puede apreciar en la **Figura 48:**

```
function [e,A,B]=sampenc_b(y,M,r)


n=length(y);
lastrun=zeros(1,n);
run=zeros(1,n);
A=zeros(M,1);
B=zeros(M,1);
p=zeros(M,1);
e=zeros(M,1);


y=y-mean(y);
s=sqrt(mean(y.^2));
y=y/s;

```

Figura 48: Algoritmo SampEn (1).

2. La segunda parte, es la implementación de la entropía muestral. Por lo que, acumula ejecuciones de puntos que coinciden dentro de la tolerancia *r* hasta que no se produzca una coincidencia, al mismo tiempo realiza un seguimiento de las coincidencias en dos contadores A(k) y B(k), de longitud k hasta m. A partir de esto, el algoritmo comienza a encontrar todos los puntos que coinciden con el primer punto de una tolerancia *r*.

   A partir de aquí, los puntos que no coinciden tendrán una longitud 0, mientras que los puntos que coinciden tendrán una longitud 1, si tras esto, los puntos que tienen longitud 1 coinciden con el siguiente punto, tendrán longitud 2, y así sucesivamente hasta que no haya coincidencia que se dejará de ejecutar el programa. A su vez, si los puntos que tenían longitud 0, coinciden con el segundo punto encontrado, pasará a tener longitud 1. Este procedimiento de búsqueda de datos se realiza hasta finalizar los datos.

   El algoritmo de Matlab quedaría como en la **Figura 49:**

```
for i=1:(n-1)
   nj=n-i;
   y1=y(i);
   for jj=1:nj
      j=jj+i;
      if abs(y(j)-y1)<r
         run(jj)=lastrun(jj)+1;
         M1=min(M,run(jj));
         for m=1:M1
            A(m)=A(m)+1;
            if j<n
               B(m)=B(m)+1;
            end
         end
      else
         run(jj)=0;
      end
   end
   for j=1:nj
      lastrun(j)=run(j);
   end
end
```

Figura 49: Algoritmo SampEn (2).

3. Por último, se completan las variables inicializadas y se tendría el valor de la entropía muestral. Se puede ver en la **Figura 50:**

```
40    N=n*(n-1)/2;
41    B=[N;B(1:(M-1))];
42    p=A./B;
43    e=-log(p);
44
```

Figura 50: Algoritmo SampEn (3).

4.2.2 Función Entropía

Tras explicar cómo funciona el algoritmo ***SampEn***, se realiza una nueva función llamada ***entropia*** en la que tendremos como objetivo calcular a entropía para cada una de nuestras épocas que hemos generado antes en la función ***vectormarca***, y tras calcular la entropía para cada una de las muestras, se calculará un nuevo vector, que indicará la entropía global de toda la señal en su conjunto.

Para ello la función tendrá el siguiente procedimiento:

1. En primer lugar, tenemos cinco parámetros de entrada y cuatro parámetros de salida:

    i. Parámetros de entrada:
        1. ***Y***: Los datos de entrada.
        2. ***M***: Longitud máxima especificada.
        3. ***R***: Tolerancia máxima.
        4. ***Dimensiones:*** Las dimensiones serán las utilizadas en el vector de ceros y unos que se crearon en anteriores funciones, ya que debe coincidir con el número de muestras.
        5. ***Vectormarca:*** Hará referencia al vector de marcas creado en la función ***generamarcas.***

    ii. Parámetros de salida:
        1. ***e:*** Estimaciones de entropía de muestra para m=0,1, …, M-1.
        2. ***A:*** Número de coincidencias para m = 1, …, M.
        3. ***B:*** Número de coincidencias para m = 0, …, M-1, excluyendo el último punto.
        4. ***Vector_entropia_global:*** Vector que indicará la entropía global de toda la señal.

    Además, se inicializará la matriz de e, A y B, y a su vez, el número de filas y de columnas que tendrá, que tiene que coincidir con las dimensiones del vector de ceros creado en funciones anteriores. Quedaría como se puede visualizar en la **Figura 51**:

```
1  function [e,A,B,vector_entropia_global]=entropia(dimensiones,y,M,r,vectormarca)
2  numerofilas = size (dimensiones,1);
3  numerocolumnas = size(dimensiones,2);
4  e = zeros (numerofilas,numerocolumnas);
5  A = zeros (numerofilas,numerocolumnas);
6  B = zeros (numerofilas,numerocolumnas);
7
```

Figura 51: Implementación Función Entropía (1).

2. Tras esto, se realiza un bucle en el que se llamará a la función ***SampEn*** y se analizará muestra a muestra nuestro vector de marcas que hemos generado en las funciones anteriores. Para ello, cuando llamamos a la función ***SampEn***, se debe poner como datos nuestro vector marca.

   Además, es importante antes de llamar a la función, habrá que comprobar si esa época es válida, es decir, antes de llamar a la función debemos comprobar si en el vector de ceros y unos que se ha calculado anteriormente de esas muestras, la época cumplía las cuatro condiciones solicitadas.

   Si es así, se llevará a cabo la entropía muestral, si no es así, pasará a tener un valor de -1, que significa que no se ha calculado la entropía muestral en esa época.

   Quedaría como la **Figura 52**:

```
for h = 1:numerocolumnas
    for v = 1:numerofilas
        if (dimensiones (v,h)) == 1
            [eaux,Aaux,Baux]=sampenc_b(y(h,vectormarca(v,1):vectormarca(v,2)),M,r);
            e (v,h) = eaux(M);
            A (v,h) = Aaux(M);
            B (v,h) = Baux(M);
        else
            e (v,h) = -1;
            A (v,h) = -1;
            B (v,h) = -1;
        end
    end
end
```

Figura 52: Implementación Función Entropía (2).

3. Otro de los pasos a seguir, es la comprobación dentro de los cálculos de la entropía muestral, que no queda ninguna muestra sin valor. Por lo que para ello mediante la función ***isnan*** comprobamos si hay algún valor sin número y si es así, pondremos el valor a -1, como entropía muestral no válida.

   Quedaría como se aprecia en la **Figura 53:**

```
for h = 1:numerocolumnas
    for v = 1:numerofilas
        notnumber = isnan(e);
        if notnumber (v,h) == 1
            e (v,h) = -1;
        else
        end
    end
end

```

Figura 53: Implementación Función Entropía (3).

4. Por último, se realiza el cálculo de la entropía global muestral. Para ello, se implementa un bucle, en el cual en primer lugar mediante la función ***find***, se encuentra las posiciones en las que se encuentran las muestras diferentes a -1, es decir, las muestras que tienen un valor de entropía muestral.

   Tras esto, se crea una variable total, en la que hace el recuento del número de muestras que tienen valor de entropía muestral, después se crea un nuevo vector con el tamaño de

las muestras con valor de entropía y luego se realiza la suma de la entropía de cada muestra. Por último, se procede al cálculo de la entropía global, que será la suma de las entropías de cada muestra dividido por el total de muestras válida y se obtendrá la entropía global de la señal.

Y, está el caso, en que ninguna señal cumpla las condiciones y tenga todo ceros el vector de ceros y unos, por lo que la entropía por muestra no se puede realizar y, por lo tanto, la entropía global sería de valor -1.

Quedaría como la **Figura 54:**

```
32    for k = 1:size(e,2)
33        numeros = (find(e(:,k)~=-1));
34        total = numel(numeros);
35        enueva = e (numeros,k);
36        suma = sum(enueva);
37        entropia_global = (suma/total)*100;
38        vector_entropia_global (k) = entropia_global;
39        if numel(numeros) == 0
40        vector_entropia_global (:,k) = -1;
41        else
42        end
43    end
```

Figura 54: Implementación Función Entropía (4).

Después de ejecutar la función tendríamos los siguientes resultados en la **Figura 55 y Figura 56:**

Vector entropía (Fórmula: $-ln(\frac{A(k)}{B(k-1)})$***):***

92x28 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	-1	0.0904	0.1149	0.1169	-1	0.1090	0.0955	0.1053	0.1604	0.1867	0.2235	0.1648	0.1561	0.2095	0.2153	0.2215	0.1428	0.1776	0.1198	0.1179	0.2630
2	0.1571	0.1336	0.0520	0.1195	-1	0.1045	0.0796	0.1129	0.1183	0.0592	0.0594	0.0699	0.0887	0.1459	0.1405	0.1419	0.1131	0.1130	0.1524	-1	0.1450
3	0.0696	-1	0.0884	0.1039	-1	0.1100	0.1115	0.1263	0.1181	0.0909	0.0735	0.0883	0.0976	0.1116	0.1095	0.1099	0.2280	0.1629	0.1738	-1	0.1530
4	0.1633	0.2149	0.1406	0.1743	-1	0.1307	0.1390	0.2045	0.1861	0.1881	0.1874	0.2008	0.2104	0.1793	0.1798	0.1828	0.2105	0.1734	0.2984	-1	0.1498
5	-1	0.1158	-1	-1	-1	0.0495	0.0577	0.0631	0.1078	0.1511	0.1428	0.1262	0.1485	0.1214	0.1265	0.1212	0.0949	-1	0.2968	-1	0.1366
6	0.1581	0.1799	0.1298	0.1915	-1	0.1861	0.1869	0.1745	0.1797	0.2075	0.1672	0.1804	0.1892	0.1608	0.1607	0.1613	0.1877	0.1656	0.2285	-1	0.1609
7	0.0984	0.1305	0.0936	0.1004	-1	0.1326	0.1474	0.1699	0.1506	0.1310	0.1327	0.1037	0.1545	0.1189	0.1188	0.1183	0.1449	0.1277	0.0766	-1	0.1309
8	0.1612	0.1848	0.1147	0.1440	-1	0.1143	0.1270	0.1688	0.1919	0.1613	0.1865	0.2022	0.2018	0.2116	0.2144	0.2053	0.1769	0.1711	0.1840	-1	0.1899
9	0.1793	0.1481	0.1311	0.1743	-1	0.1161	0.1014	0.1331	0.1046	0.1160	0.1049	0.1902	0.1729	0.1986	0.1945	0.1977	0.1216	0.1165	0.1217	-1	0.1055
10	0.2069	0.1036	0.0993	0.1371	-1	0.1414	0.1045	0.0723	0.0887	0.0776	0.0949	0.1402	0.0834	0.1463	0.1505	0.1499	0.0625	0.0574	0.0731	-1	0.0807
11	-1	-1	0.0703	0.0706	-1	0.0639	0.0988	0.0830	0.1374	0.0865	0.1564	0.2467	0.1847	0.1298	0.1287	0.1264	0.2263	0.1935	0.1504	-1	-1
12	0.1556	0.1457	0.1266	0.1370	-1	0.0892	0.0863	0.1086	0.1142	0.1361	0.1270	0.1054	0.1538	0.1670	0.1700	0.1649	0.1863	0.1548	0.1382	-1	0.1451
13	0.2310	0.1772	0.1355	0.1838	-1	0.1418	0.1069	0.1071	0.1557	0.1470	0.1766	0.1740	0.2284	0.2007	0.1958	0.1970	0.2087	0.1856	0.2074	0.1603	0.1928
14	0.1015	0.1229	0.0891	0.1281	-1	0.0780	0.0974	0.0818	0.1000	0.0469	0.1400	0.1336	0.1465	0.1595	0.1617	0.1551	0.1612	0.1530	0.1135	-1	0.1328
15	0.2003	0.2340	0.1923	0.1928	-1	0.1390	0.1665	0.1887	0.1564	0.1183	0.1375	0.1240	0.2280	0.1721	0.1736	0.1723	0.2095	0.1553	0.1707	-1	0.1772
16	0.1538	0.1780	0.1511	0.2185	-1	0.1664	0.1473	0.1258	0.1243	0.0960	0.1697	0.1752	0.1300	0.1510	0.1467	0.1613	0.1086	0.1957	0.1715	0.1037	0.0852
17	0.1961	0.2225	0.1025	0.1563	-1	0.2167	0.1992	0.2077	0.2071	0.1210	0.1419	0.1217	0.1693	0.1376	0.1351	0.1353	0.1555	0.1562	0.1559	-1	0.1853
18	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
19	0.1714	0.2026	0.1347	0.1690	-1	0.1655	0.1826	0.2148	0.2316	0.2117	0.1662	0.1577	0.1967	0.2017	0.2094	0.2104	0.2027	0.0958	0.1632	-1	0.2066
20	0.1412	0.1841	0.1289	0.1623	-1	0.1356	0.1644	0.1716	0.1627	0.1268	0.1531	0.1583	0.2196	0.1376	0.1375	0.1347	0.1821	0.1990	0.1854	0.1907	0.1245
21	0.1350	0.1623	0.0785	0.1460	-1	0.1205	0.0844	0.0951	0.1372	0.0848	0.1175	0.2103	0.1862	0.2024	0.1986	0.1883	0.1922	0.1897	-1	-1	0.0840
22	0.1301	0.1612	0.1470	0.1415	-1	0.1416	0.1335	0.1590	0.1269	0.1061	0.0927	0.1395	0.2237	0.1484	0.1498	0.1561	0.1525	0.1439	0.1609	-1	0.1724
23	0.2369	0.2626	0.2140	0.1549	-1	0.1251	0.1182	0.1224	0.1717	0.1945	0.1571	0.2089	0.2478	0.1810	0.1770	0.1764	0.2114	0.2096	0.1781	-1	0.1279
24	0.1863	0.2165	0.1333	0.1607	-1	0.0869	0.0881	0.0937	0.2661	0.2581	0.1520	0.1958	0.1731	0.1483	0.1444	0.1460	0.0966	0.1674	0.1784	-1	0.1562
25	-1	-1	0.1073	0.1869	-1	0.2142	0.2333	0.3265	0.3008	0.0958	0.1695	0.1366	0.1978	0.1117	0.1072	0.1106	0.2309	0.2271	-1	0.1997	0.2755
26	0.2586	0.2411	0.0872	0.1400	-1	0.1318	0.1533	0.1317	0.1549	0.0985	0.1004	0.1590	0.1994	0.1460	0.1425	0.1514	0.1459	0.1432	-1	-1	0.1050
27	0.1543	0.1539	0.1689	0.1354	-1	0.1013	0.0738	0.0836	0.1132	0.1178	0.1340	0.1565	0.1112	0.1349	0.1393	0.1365	0.1762	0.1588	0.1792	-1	0.1026
28	-1	0.3082	0.1581	0.2085	-1	0.2182	0.2117	0.2577	0.3381	0.2127	0.1565	0.1257	0.1785	0.1682	0.1784	0.1721	0.1594	0.1561	0.2067	0.1821	0.1556

Figura 55: Vector entropía.

Vector entropía global:

1x28 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	21.9413	22.8304	13.2772	18.1456	-1	13.8764	13.9161	15.4995	19.5245	16.4372	14.4003	15.7680	15.8681	15.0427	15.0553	15.0227	15.7408	16.6355	18.4840	15.9072	12.8711

Figura 56: Vector Entropía Global.

Estos serían los resultados que se obtendrían de la entropía. Los valores que da el algoritmo hay que estudiarlos entre señales. Los valores bajos indican señales más similares, es decir patrones más parecidos, en cambio los valores altos, todo lo contrario. Depende de cómo funciona el cerebro, o la tarea que se está realizando, la señal será más o menos entrópica.

Capítulo 5. App Designer

5.1 Interfaz Gráfica

Tras la implementación de los algoritmos de preprocesado y la utilización del SampEn para la entropía muestral, se ha creado una interfaz gráfica en App Designer para facilitar al usuario el acceso a dichos algoritmos, sin la necesidad de utilizar Matlab.

Para ello se ha creado un Panel llamado ***Interfaz Gráfica EEG*** y dentro de éste se ha dividido la aplicación en cinco subapartados, que serán explicados detalladamente más adelante: ***Carga Señales, Filtrado Señal, Filtrado Latencia, Señales Después Filtrado y Entropía Muestral.***

5.1.1 Carga Señales

En primer lugar, tenemos la pestaña de ***Carga Señales***, en la que tendremos un panel de carga, dónde se tiene la posibilidad de seleccionar un archivo .mat para cargar los archivos. A su vez, tendremos un área de texto, donde se verá que elemento se ha cargado en la aplicación. A continuación, se va a proceder a explicar la implementación de manera detallada.

En primer lugar, se utiliza la función ***BotonCargarArchivoMatPushed***, que corresponde a un botón, ya que al ser pulsado será cuando realice la carga de archivos.

La primera línea de código devuelve el nombre del documento y dónde se encuentra el archivo gracias al comando ***uigetfile***, además se le exige que el archivo sea procedente de Matlab. Tras esto, se cargan todos los documentos dentro de la variable app.fichero. Y, por último, dentro de un área de texto se indica cuál es el documento cargado. Se puede ver el código en la **Figura 57:**

```
function BotonCargarArchivoMatPushed(app, event)
    [file, path] = uigetfile('*.mat');
    app.fichero = load(fullfile(path,file));
    msgbox('El documento cargado es correcto')
    set (app.DocumentoCargado, 'Value', file)
end
```

Figura 57: Implementación Carga Señales (1).

De manera representativa, se puede ver en la **Figura 58:**

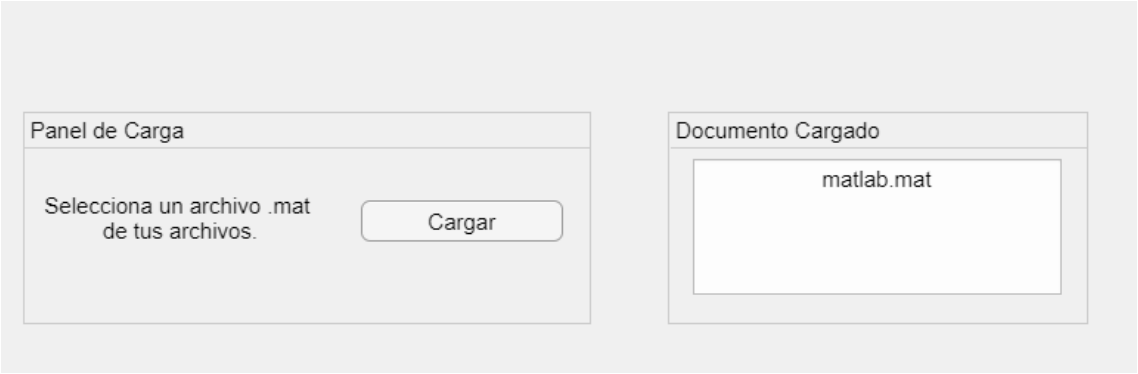


Figura 58: Representación Carga Señales (1).

Tras cargar el archivo de Matlab, ya se puede comenzar a utilizar todos los datos que contiene dicho archivo. Dentro de esta pestaña de la carga de señales, también existe la posibilidad de representar las señales.

Se podrá representar de distintas formas. La primera forma es la representación de todas las señales, mientras que la segunda forma es la representación de las señales una a una, pudiendo indicar la frecuencia de muestreo que desea el usuario.

Para la primera forma de representación tenemos un botón llamado ***Representar todas las señales***, en la que tendremos el código que se puede ver en la **Figura 59**.

Se compone en primera instancia de una inicialización de la gráfica donde se va a representar las señales, y a continuación, contiene un bucle ***for***, cuya función es representar una a una las señales del fichero.

```
function RepresentartodaslassealesButtonPushed(app, event)
    cla (app.Grafica, "reset")
    hold (app.Grafica,"on")%Los ejes
    ylim (app.Grafica, [-5000 5000])
    for i = 1:app.fichero.EEG.nbchan
    plot (app.Grafica, app.fichero.EEG.data(i,:))
    end
```

Figura 59: Implementación Carga Señales (2).

Por otro lado, tenemos la siguiente forma de representación, en la que el usuario podrá elegir tanto la frecuencia de muestreo, como la señal a representar. Para ello, se ha implementado el código de la **Figura 60**.

Dentro del código, tenemos en primer lugar un ***Drop Down***, en la que va a permitir elegir cuál señal se quiere representar. Para ello se realiza un bucle ***for***, el cual añadirá todas las señales que estén dentro del fichero. Tras esto, se define un botón llamado ***Representar Señal***, en el cual se inicializa la gráfica nuevamente y se realiza la representación de la gráfica teniendo en cuenta la frecuencia de muestreo que ha indicado el usuario.

```
function ListaSenyalesDropDownOpening(app, event)
    string vector_signal;
    c = 1;
    for f = 1:app.fichero.EEG.nbchan
            vector_signal (f,c) = ("Señal " + f);
            vector_data (f,c) = f;
    end
    app.ListaSenyales.Items = vector_signal;
    app.ListaSenyales.ItemsData = vector_data;
end

% Button pushed function: RepresentarSignal
function RepresentarSignalPushed(app, event)
    cla (app.Grafica, "reset")
    hold (app.Grafica,"on")
    t = (0:(length(app.fichero.EEG.data)-1))*1/app.FrecuenciaMuestreo.Value;
    plot (app.Grafica, t, app.fichero.EEG.data(app.ListaSenyales.Value,:))
```

Figura 60: Implementación Carga Señales (3).

La representación quedaría como se puede apreciar en la **Figura 61 y Figura 62**. En primer lugar, es el caso de la representación de todas las señales y el segundo caso, sería el de la representación de una señal.

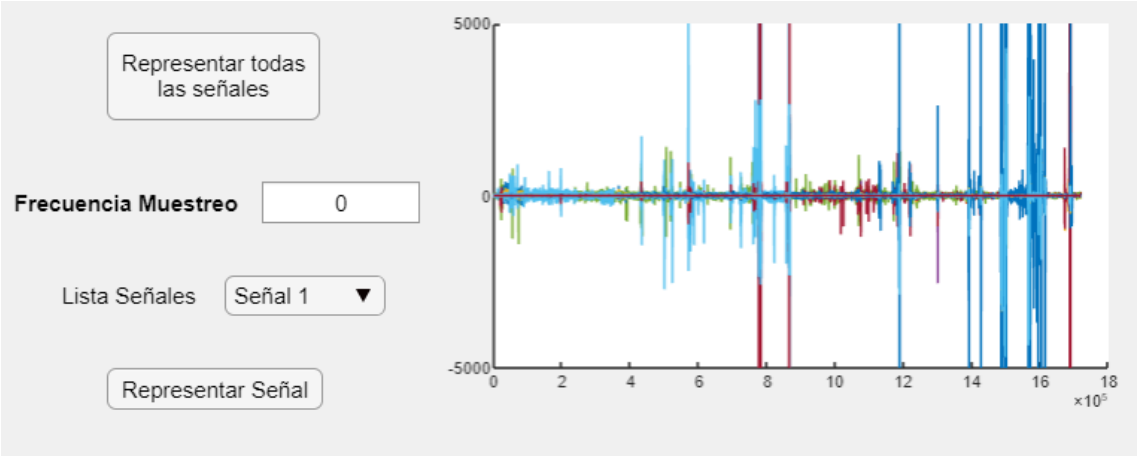


Figura 61: Representación Todas Señales.

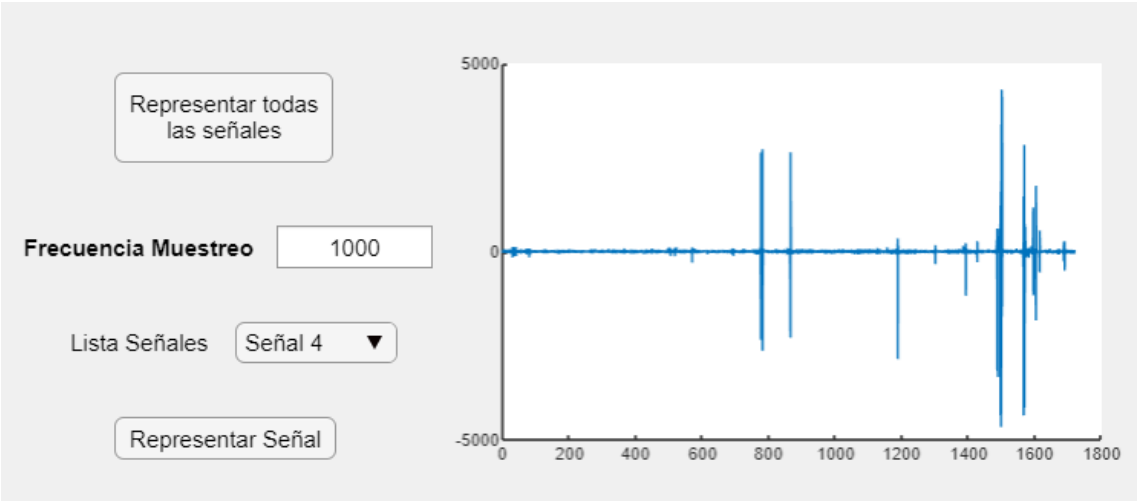


Figura 62: Representación de una señal con su Frecuencia de Muestreo.

Y, en la **Figura 63**, se puede ver como quedaría por completo la pestaña de ***Carga Señales.***

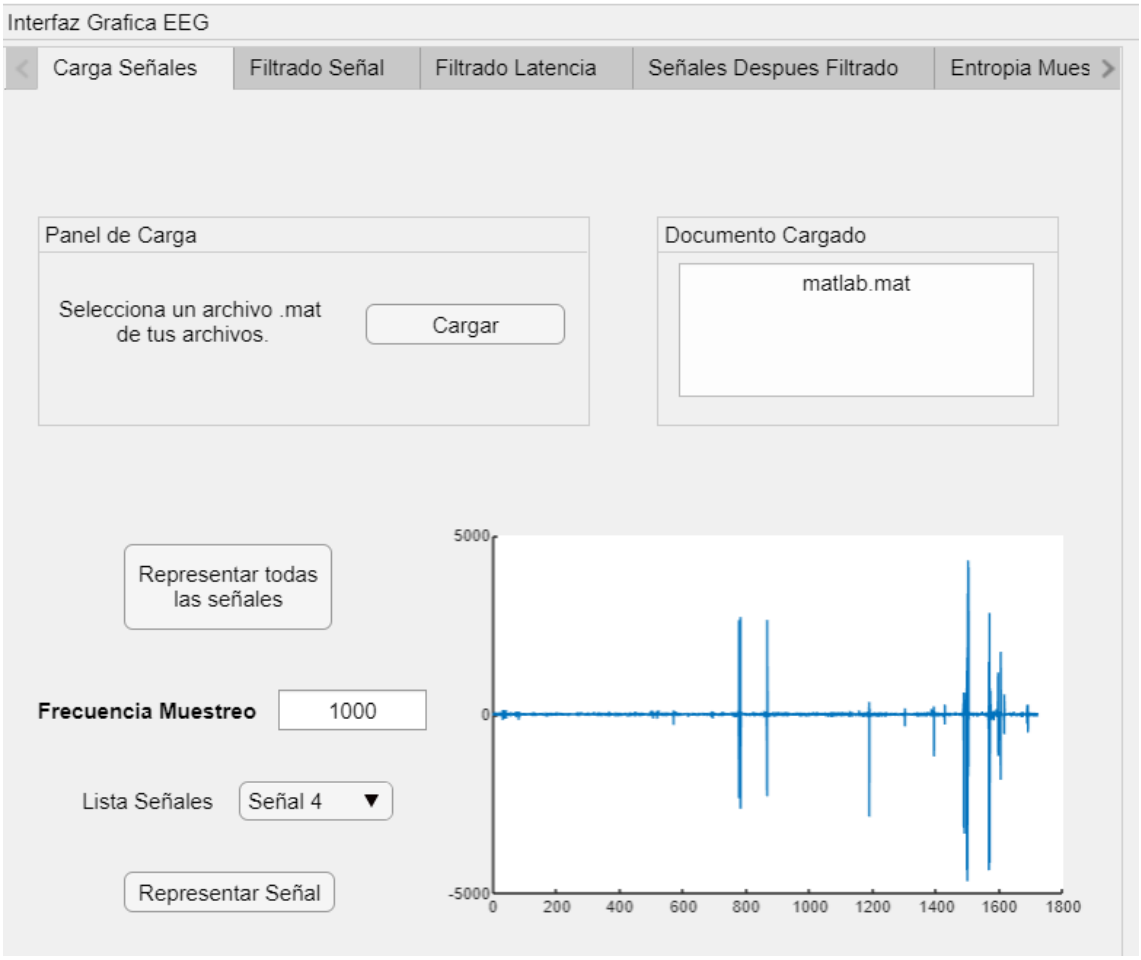


Figura 63: Carga Señales Completa.

5.1.2 Filtrado Señal

En segundo lugar, se encuentra la pestaña ***Filtrado Señal***, en la que tendremos seis áreas de texto numéricas. La primera se utilizará para indicar el tiempo de época que se quiere utilizar en el filtrado de la señal, las cuatro siguientes será para que el usuario escoja los valores de tensión que se adaptan mejor a su filtrado, y, por último, se tendrá un área para indicar el umbral, es decir, el % de veces que deben cumplirse las condiciones para que la señal sea considerada válida.

Por último, se tendrá dos botones de representación de tablas. La primera de ellas será la tabla de validez de épocas, que será igual que la vista en la **Figura 36** y la tabla % Validez Épocas que es la misma que la vista en la **Figura 35.** A continuación, se va a explicar detalladamente el código implementado en esta pestaña.

Como se puede ver en la **Figura 64, Figura 65 y Figura 66,** el código que se ha implementado para rellenar las tablas se ejecutará al pulsar los botones de representación de tabla.

Además, el código es el mismo que el que se ha utilizado para la implementación de los algoritmos en Matlab, ya explicado anteriormente. Los cambios que se han producido en la implementación del código es que el tiempo de época, las condiciones y el umbral, ya no es un valor fijo, sino que depende del valor que proporciona el usuario en la aplicación.

```
function TablaValidezEpocasRepresentarPushed(app, event)
l = length (app.fichero.EEG.data); %Longitud total de la señal
tiempo = l*1/app.FrecuenciaMuestreo.Value; % Longitud en tiempo
muestrasepoca = floor (app.TiempoEpoca.Value*app.FrecuenciaMuestreo.Value);
epocas = floor (l/muestrasepoca);
app.TablaEpocas.ColumnName = app.ListaSenyales.Items;
app.TablaEpocas.Data = zeros(epocas,app.fichero.EEG.nbchan);
ValorMinimo = app.Condicion2VMin.Value;
ValorMaximo = app.Condicion1VMax.Value;
Cond3 = app.Condicion3absVn1Vn.Value;
Cond4 = app.Condicion4absVMaxVMin.Value;
size1 = size(app.TablaEpocas.Data,2);
size2 = size(app.TablaEpocas.Data, 1);

for i = 1:size1 %Columnas
    x = app.fichero.EEG.data (i,:);
    p = 1;%Inicializamos valor vector inicial
        for h = 1:size2 %Filas
            vector = x (p:p+(muestrasepoca-1));
            vectordesplazado = zeros(size(vector));
            vectordesplazado (2:end) = vector(1:end-1);
            p = p+muestrasepoca;
            aux = 1;
                if min (vector)> ValorMinimo
                     %disp ('Es correcto')
                else
                    %disp ('No es correcto, ya que el mínimo es mayor que -100uV')
                    aux = 0;

                end
```

Figura 64: Implementación Filtrado señal (1).

```
                if max (vector)<ValorMaximo
                    %disp ('Es correcto')
                else
                    %disp ('No es correcto, ya que el máximo es menor que 100uV')
                    aux = 0;
                end

                if abs (max(vector)-min(vector)) <Cond3
                    %disp ('Es correcto')
                else
                    %disp ('No es correcto, ya que el absoluto de mayor menos el menor es menor que 100uV')
                    aux = 0;
                end
                if abs (vectordesplazado-vector)<Cond4
                    %disp ('Es correcto')
                else
                    %disp('No es correcto, ya que el absoluto de una muestra con la anterior es mayor que 100uV')
                    aux = 0;
                end
                    app.TablaEpocas.Data(h,i) = aux;

        end
end
end
```

Figura 65: Implementación Filtrado Señal (2).

```
function TablaValidezEpocasPorcentajeRepresentarPushed(app, event)
    l = length (app.fichero.EEG.data); %Longitud total de la señal
    tiempo = l*1/app.FrecuenciaMuestreo.Value; % Longitud en tiempo
    muestrasepoca = floor (app.TiempoEpoca.Value*app.FrecuenciaMuestreo.Value);
    epocas = floor (l/muestrasepoca);
    app.TablaPorcentajeEpocas.ColumnName = app.ListaSenyales.Items;

    for k = 1:size(app.TablaEpocas.Data,2)
    numeros = numel (find(app.TablaEpocas.Data(:,k)==1));
    umbral = (numeros/epocas)*100;
    app.TablaPorcentajeEpocas.Data (k) = umbral;
    if umbral<app.Umbral.Value
       app.TablaEpocas.Data (:,k) = zeros (size(app.TablaEpocas.Data(:,k)))
    else
        disp ('Sigue igual')
    end
    end
end
```

Figura 66: Implementación Filtrado Señal (3).

Si ejecutamos la aplicación y rellenamos las áreas numéricas con los datos de nuestro filtrado, tendríamos la pestaña completa de la manera que se puede apreciar en la **Figura 67.**

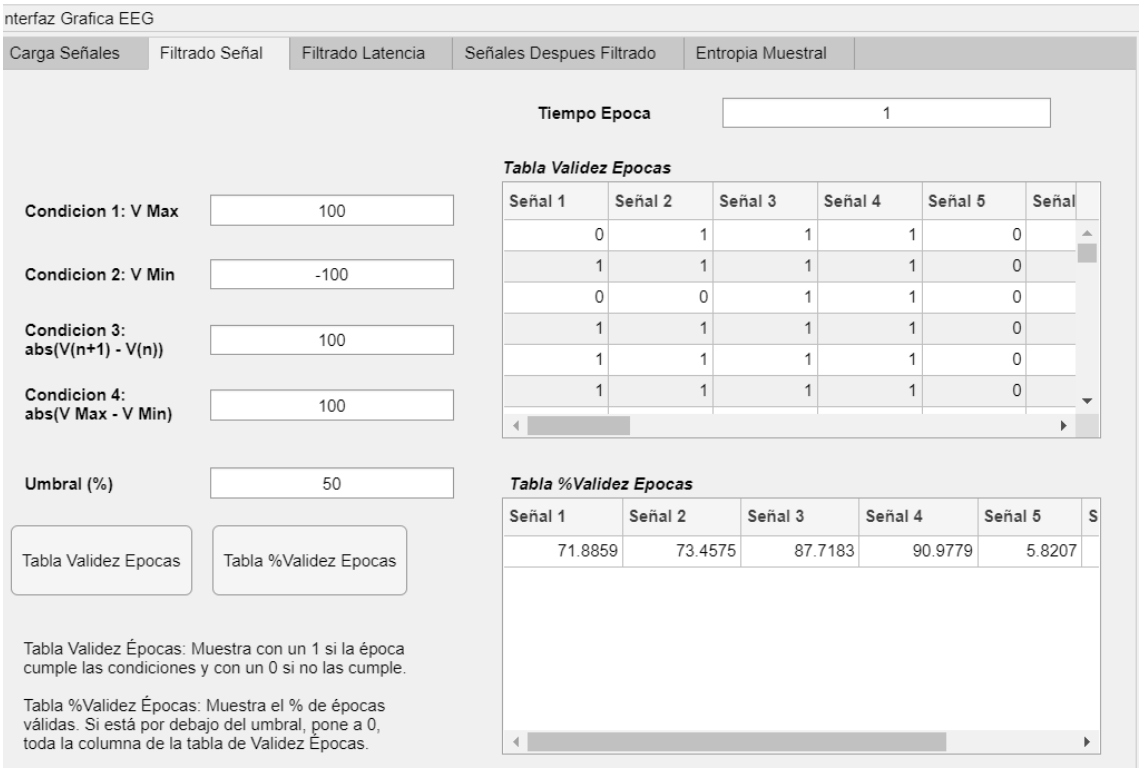


Figura 67: Filtrado Señal Completa.

5.1.3 Filtrado Latencia

En tercer lugar, se encuentra la pestaña ***Filtrado Latencia***, en la que tendremos dos áreas de texto numéricas. La primera se utilizará para indicar el tiempo antes de la muestra y el segundo para indicar el tiempo después de la muestra. Tras esto, se encuentran tres botones en los que se completarán tres tablas. La tabla de latencia donde indicará el tiempo inicial, el tiempo final de cada época y el tipo de señal. Y las tablas de validez de épocas si cumplen las condiciones y el porcentaje de épocas que cumplen las condiciones.

Como se puede ver en la **Figura 68, Figura 69**, **Figura 70 Figura 71 y Figura 72** el código que se ha implementado para rellenar las tablas se ejecutará al pulsar los botones de representación de las tablas.

Además, el código es el mismo que el que se ha utilizado para la implementación de los algoritmos en Matlab, ya explicado anteriormente. Los cambios que se han producido en la implementación del código es que el tiempo de antes de la muestra y el tiempo después de la muestra, ya no es fijo, sino será el indicado por el usuario en las áreas numéricas.

```
function TablaLatenciaRepresentarButtonPushed(app, event)
    numerofilas = size (app.fichero.EEG.event,2);
    numerocolumnas = 3;
    app.TablaLatencia.ColumnName = {'Tiempo 1';'Tiempo 2'; 'Tipo'};
    app.TablaLatencia.Data = zeros (numerofilas,numerocolumnas);
    for i = 1:size(app.TablaLatencia.Data,1)
     latencia = app.fichero.EEG.event (i).latency;
     app.TablaLatencia.Data (i,1) = latencia - round(app.TiempoAntesMuestra.Value*app.FrecuenciaMuestreo.Value);
     app.TablaLatencia.Data (i,2) = latencia + round(app.TiempoDespuesMuestra.Value*app.FrecuenciaMuestreo.Value);

     str = 'boundary';
     str1 = 'S 15';
     str2 = 'S  5';
     str3 = 'S  7';
     str4 = 'S  9';
     type = app.fichero.EEG.event (i).type;

     if (strcmp (type, str))
         app.TablaLatencia.Data (i,3) = 1;
     else
         disp ('No es este tipo');
     end

     if (strcmp (type, str1))
         app.TablaLatencia.Data (i,3) = 2;
     else
         disp ('No es este tipo');
     end
```

Figura 68: Implementación Filtrado Latencia (1).

```
     if (strcmp (type, str2))
         app.TablaLatencia.Data (i,3) = 3;
     else
         disp ('No es este tipo');
     end

     if (strcmp (type, str3))
         app.TablaLatencia.Data (i,3) = 4;
     else
         disp ('No es este tipo');
     end

     if (strcmp (type, str4))
         app.TablaLatencia.Data (i,3) = 5;
     else
         disp ('No es este tipo');
     end
     app.TablaLatencia.Data (1,1) = 1;

     end
end
```

Figura 69: Implementación Filtrado Latencia (2).

```
function TablaValidezEpocasLatenciaRepresentarPushed(app, event)
    l = length (app.fichero.EEG.data); %Longitud total de la señal
    tiempo = l*1/app.FrecuenciaMuestreo.Value; % Longitud en tiempo
    epocas = size(app.TablaLatencia.Data,1);
    app.TablaEpocasLatencia.ColumnName = app.ListaSenyales.Items;
    app.TablaEpocasLatencia.Data = zeros(epocas,app.fichero.EEG.nbchan);

    for i = 1:size(app.TablaEpocasLatencia.Data,2)
        x = app.fichero.EEG.data (i,:);
        p = 1; %Inicializamos valor vector inicial

    for h = 1:size(app.TablaEpocasLatencia.Data, 1) %filas
    %disp (size(vectorzeros,1));

        vector = x (app.TablaLatencia.Data(h,1):app.TablaLatencia.Data(h,2));
        vectordesplazado = zeros(size(vector));
        vectordesplazado (2:end) = vector(1:end-1);
        aux = 1;

        if min (vector)>app.Condicion2VMin.Value
                 disp ('Es correcto')
            else
                disp ('No es correcto, ya que el mínimo es mayor que -100uV')
                aux = 0;

        end
```

Figura 70: Implementación Filtrado Latencia (3).

```
        if max (vector)<app.Condicion1VMax.Value
            disp ('Es correcto')
        else
            disp ('No es correcto, ya que el máximo es menor que 100uV')
            aux = 0;
        end

        if abs (max(vector)-min(vector)) <app.Condicion3absVn1Vn.Value
            disp ('Es correcto')
        else
            disp ('No es correcto, ya que el absoluto de mayor menos el menor es menor que 100uV')
            aux = 0;
        end
        if abs (vectordesplazado-vector)<app.Condicion4absVMaxVMin.Value
            disp ('Es correcto')
        else
            disp('No es correcto, ya que el absoluto de una muestra con la anterior es mayor que 100uV')
            aux = 0;
        end
            app.TablaEpocasLatencia.Data(h,i) = aux;

    end
    end
end
```

Figura 71: Implementación Filtrado Latencia (4).

```
function TablaValidezEpocasPorcentajesRepresentarLatenciaButtonPushed(app, event)
    epocas = size(app.TablaLatencia.Data,1);
    app.TablaPorcentajeEpocasLatencia.ColumnName = app.ListaSenyales.Items;

    for k = 1:size(app.TablaEpocasLatencia.Data,2)
        numeros = numel (find(app.TablaEpocasLatencia.Data(:,k)==1));
        umbral = (numeros/epocas)*100;
        cla (app.TablaPorcentajeEpocasLatencia, "reset")
        app.TablaPorcentajeEpocasLatencia.Data (1,k) = umbral;
            if umbral<app.Umbral.Value
                app.TablaEpocasLatencia.Data (:,k) = zeros (size(app.TablaEpocasLatencia.Data(:,k)));
            else
                disp ('Sigue igual')
            end
    end
end
```

Figura 72: Implementación Filtrado Latencia (5).

Si ejecutamos la aplicación y rellenamos las áreas numéricas con los datos de nuestro filtrado, tendríamos la pestaña completa de la manera que se puede apreciar en la **Figura 73.**

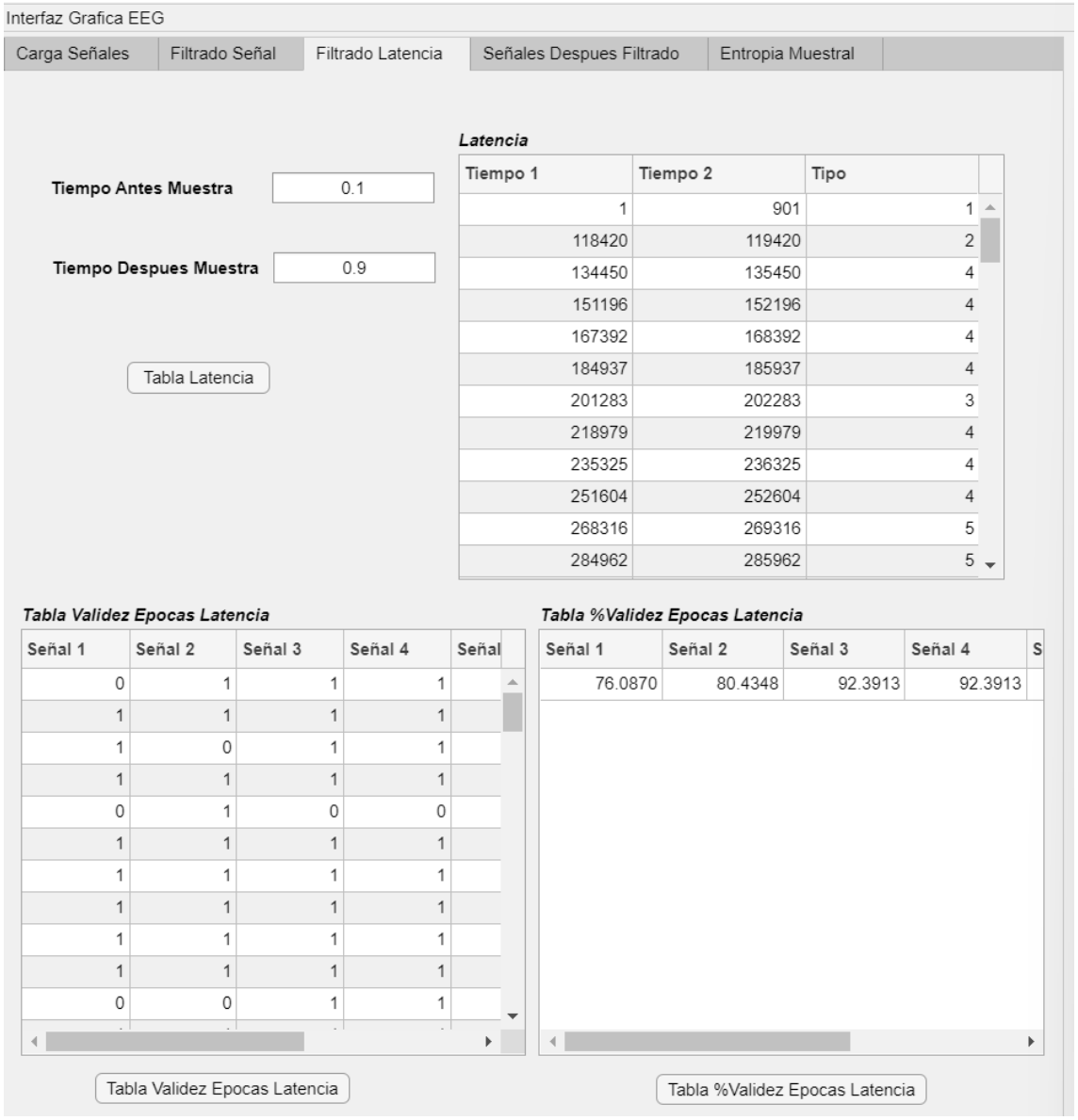


Figura 73: Filtrado Latencia Completa.

5.1.4 Señales Después Filtrado

En cuarto lugar, se encuentra la pestaña ***Señales Después Filtrado***, en la que tenemos directamente un botón, el cual comprueba el número de señales han sido eliminadas tras no cumplir el umbral necesario de épocas válidas e indica cuales señales han sido eliminadas. Tras esto, hay una nueva gráfica, donde solo se podrá representar las señales válidas, mientras que las señales no válidas, dará un error, ya que han sido eliminadas.

Como se puede ver en la **Figura 74,** el código implementado se ejecutará al pulsar los botones correspondientes.

El código en este caso, se declaran dos variables, un double, que se irá rellenando señal a señal y un string, el cual se irá rellenando con las señales eliminadas de la forma: ***"Señal + k + Eliminada"*** y con las señales no eliminadas: ***"Señal + k".***

```
function ComprobarSignalsEliminadasButtonPushed(app, event)
    string vector_signal2
    double vector_data2
    aux = 0;
    h = 1;
    for k = 1:app.fichero.EEG.nbchan
        if (app.TablaPorcentajeEpocasLatencia.Data (1,k)<app.Umbral.Value)
            aux = aux + 1;

            vector_eliminado (1,aux) = k;

            vector_signal2 (k,h) = ("Señal " + k + " Eliminada.")
            vector_data2 (k,h) = k;
        else
            vector_signal2 (k,h) = ("Señal " + k);
            vector_data2 (k,h) = k;
        end
    end
    for k = 1:aux
        app.Ylassenyalesson.Value = (app.Ylassenyalesson.Value + " Señal " + vector_eliminado(1,k));
    end
    app.vector_eliminado = vector_eliminado;
    app.SignalsFiltradas.Items = vector_signal2;
    app.SignalsFiltradas.ItemsData = vector_data2;
    app.Elnumerodesenyaleseliminadasson.Value = num2str(aux);
end
```

Figura 74: Implementación Señales Después Filtrado (1).

Por último, se implementa el código para representar las señales, que sigue el mismo procedimiento que las anteriores gráficas, se puede comprobar en la **Figura 75.**

```
function RepresentarButtonPushed(app, event)
    cla (app.GraficaFinal, "reset")
    hold (app.GraficaFinal,"on")
    t = (0:(length(app.fichero.EEG.data)-1))*1/app.FrecuenciaMuestreo.Value;
    %if (app.SignalsFiltradas.Value ~= app.vector_eliminado)
    %disp (length (find(app.vector_eliminado == app.SignalsFiltradas.Value)))
    if length (find(app.vector_eliminado == app.SignalsFiltradas.Value)) == 0
        plot (app.GraficaFinal, t, app.fichero.EEG.data(app.SignalsFiltradas.Value,:));
    else
        msgbox('La señal está eliminada, no se puede representar', 'Error', 'Error');
    end
end
```

Figura 75: Implementación Señales Después Filtrado (2).

Si pulsamos los botones, podemos comprobar lo explicado anteriormente en la **Figura 76 y Figura 77.**

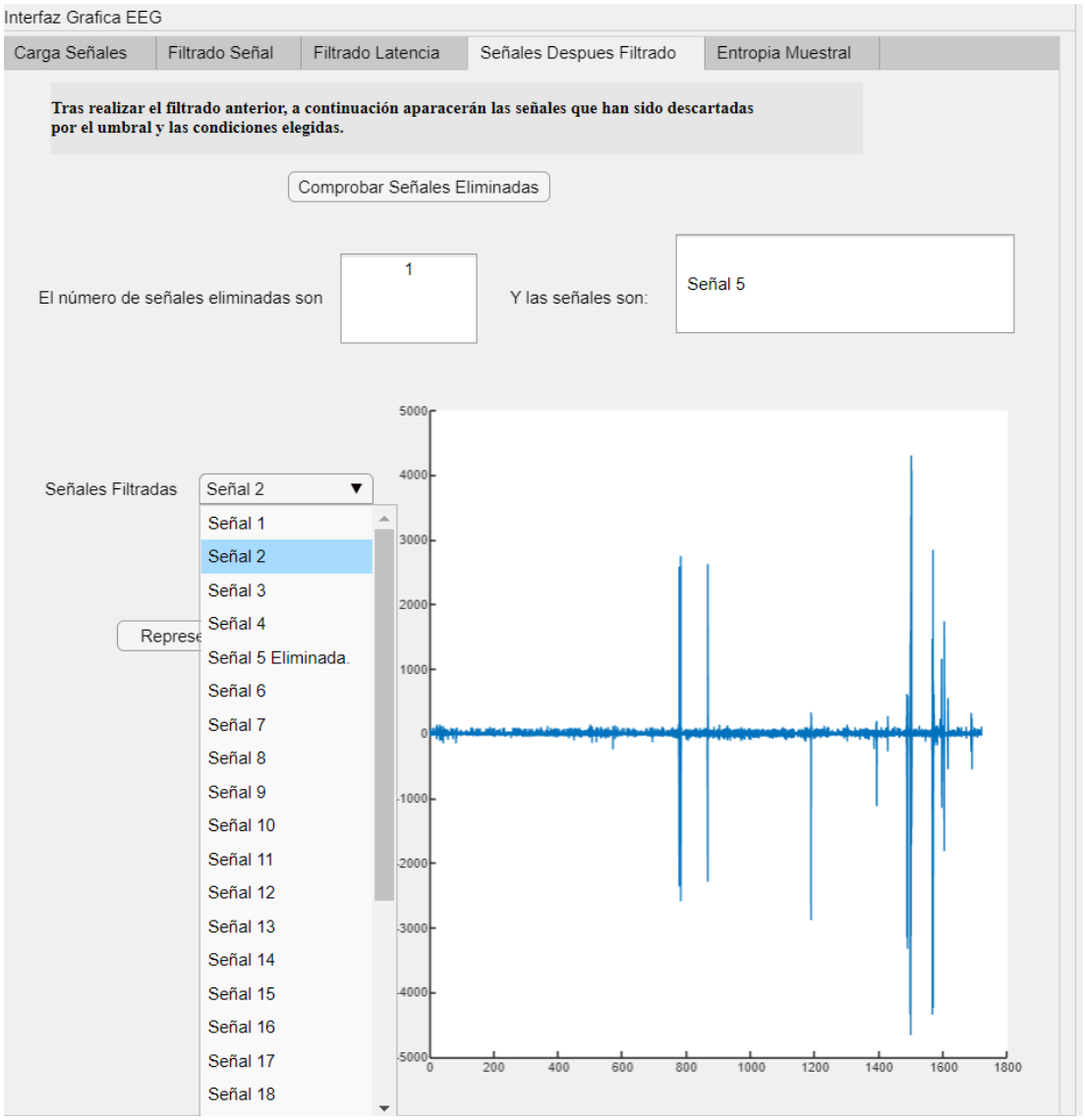


Figura 76: Señales Después Filtrado Completa (1).

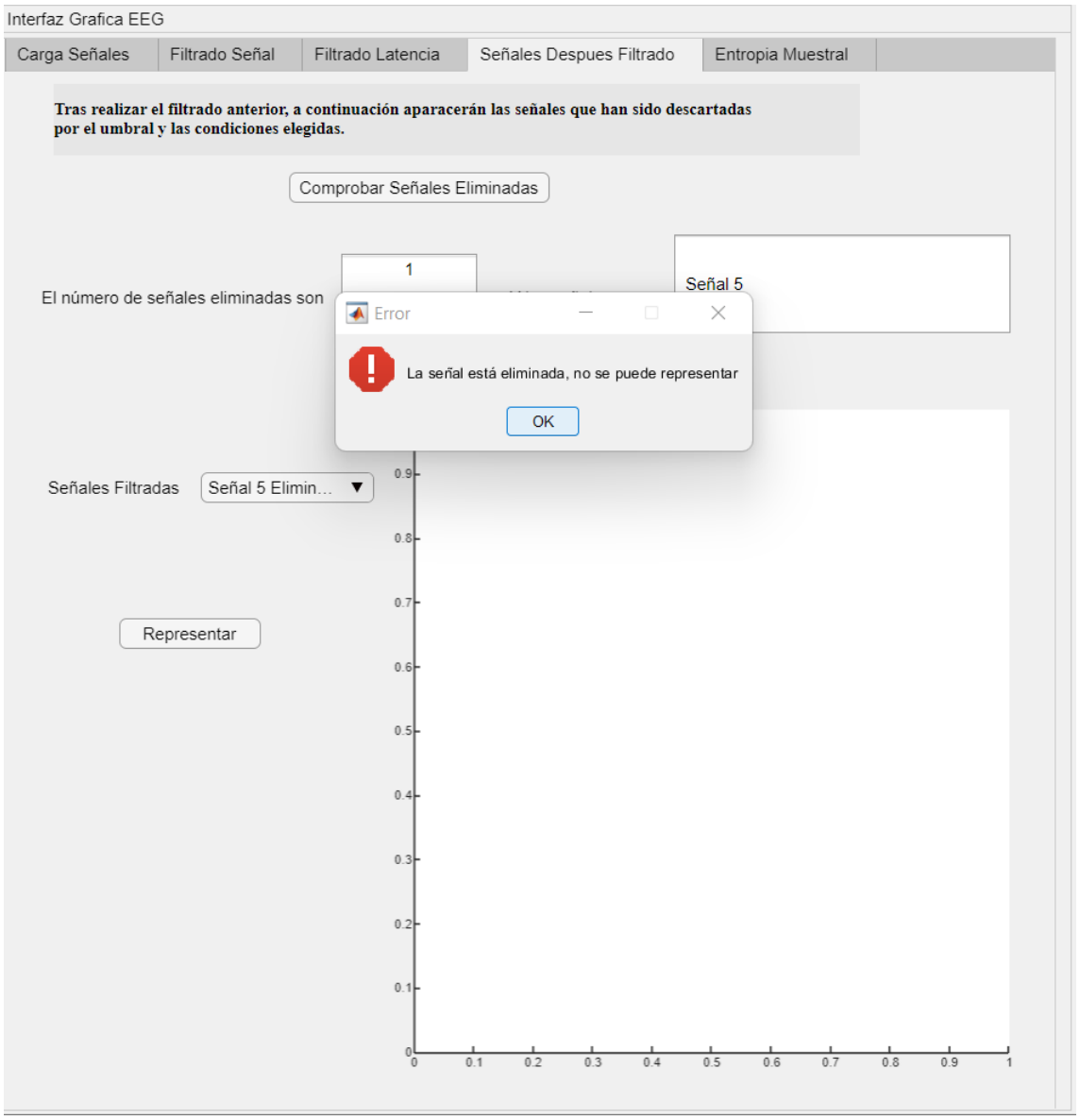


Figura 77: Señales Después Filtrado Completa (2).

5.1.5 Entropía Muestral

Por último, se encuentra la pestaña ***Entropía Muestral***, en la que tenemos dos áreas numéricas con la longitud máxima del modelo y la tolerancia de coincidencia, que serán indicadas por el usuario. Por otro lado, tenemos tres botones, el primero calcula la entropía, el segundo calcula la entropía global, y el último guarda los datos, generando un archivo de Excel, donde se podrán ver los resultados obtenidos en la entropía muestral.

Como se puede ver en la **Figura 78 y Figura 79,** el código implementado se ejecutará al pulsar los botones correspondientes.

El código en este caso, para el primer botón del cálculo de la entropía es una adaptación del mismo código utilizado en Matlab, lo único que el valor de M y r serán proporcionados por el usuario que esté utilizando la aplicación.

```
function CalcularEntropiaButtonPushed(app, event)
    numerofilas = size (app.TablaEpocasLatencia.Data,1);
    numerocolumnas = size(app.TablaEpocasLatencia.Data,2);
    EntradaY = app.fichero.EEG.data;
    app.ATable.Data = zeros (numerofilas,numerocolumnas);
    app.BTable.Data = zeros (numerofilas,numerocolumnas);
    app.EntropiaMuestras.Data = zeros (numerofilas,numerocolumnas);
    app.ATable.ColumnName = app.ListaSenyales.Items;
    app.BTable.ColumnName = app.ListaSenyales.Items;
    app.EntropiaMuestras.ColumnName = app.ListaSenyales.Items;
    ValorR = app.r.Value;
    ValorM = app.M.Value;
    for h = 1:numerocolumnas
        for v = 1:numerofilas
            if (app.TablaEpocasLatencia.Data (v,h)) == 1
             %which ('EntradaY');
             %which ('ValorR');
             %which ('ValorM');
             %which ('sampenc_b');
            [eaux,Aaux,Baux]=sampenc_b(EntradaY(h,app.TablaLatencia.Data(v,1):app.TablaLatencia.Data(v,2)),ValorM,ValorR);
            app.ATable.Data (v,h) = Aaux(ValorM);
            app.BTable.Data (v,h) = Baux(ValorM);
            app.EntropiaMuestras.Data (v,h) = eaux(ValorM);
            else
            app.ATable.Data (v,h) = -1;
            app.BTable.Data (v,h) = -1;
            app.EntropiaMuestras.Data (v,h) = -1;
            end
        end
    end
```

Figura 78: Implementación Entropía Muestral (1).

```
    for h = 1:numerocolumnas
        for v= 1:numerofilas
            notnumber = isnan (app.EntropiaMuestras.Data);
            if notnumber (v,h) == 1
                app.EntropiaMuestras.Data (v,h) = -1;
            else
            end
        end
    end
end
```

Figura 79: Implementación Entropía Muestral (2).

Para el segundo botón, el del cálculo de la entropía global es también una adaptación del código utilizado en Matlab. Se puede ver en la **Figura 80:**

```
function CalcularEntropiaGlobalButtonPushed(app, event)
app.EntropiaGlobal.ColumnName = app.ListaSenyales.Items;
    for k = 1:size(app.EntropiaMuestras.Data,2)
        numeros = (find(app.EntropiaMuestras.Data(:,k)~=-1));
        total = numel(numeros);
        enueva = app.EntropiaMuestras.Data (numeros,k);
        suma = sum(enueva);
        entropia_global = (suma/total)*100;
        app.EntropiaGlobal.Data (k) = entropia_global;
        if numel(numeros) == 0
        app.EntropiaGlobal.Data (:,k) = -1;
        else
        end
    end
end
```

Figura 80: Implementación Entropía Muestral (3).

El último botón, el que guarda los datos en formato Excel, en primer lugar, inicializaremos cuatro variables, donde guardaremos los resultados de las tablas calculadas al pulsar los anteriores botones. Tras esto, se guardarán los resultados en formato ***.xlsx***, y utilizando el comando writematrix, guardará los datos en formato Excel en el directorio del usuario. Se puede apreciar en la **Figura 81:**

```
        function GuardardatosButtonPushed(app, event)
            data = app.EntropiaGlobal.Data;
            data1 = app.ATable.Data;
            data2 = app.BTable.Data;
            data3 = app.EntropiaMuestras.Data;
            filename = 'EntropiaGlobal.xlsx';
            filename1 = 'A.xlsx';
            filename2 = 'B.xlsx';
            filename3 = 'EntropiaEpocas.xlsx';
            writematrix(data,filename)
            writematrix(data1,filename1)
            writematrix(data2,filename2)
            writematrix(data3,filename3)
        end
    end
```

Figura 81: Implementación Entropía Muestral (4).

En la **Figura 82 y Figura 83**, se puede ver como quedaría la interfaz gráfica, pulsando los botones.

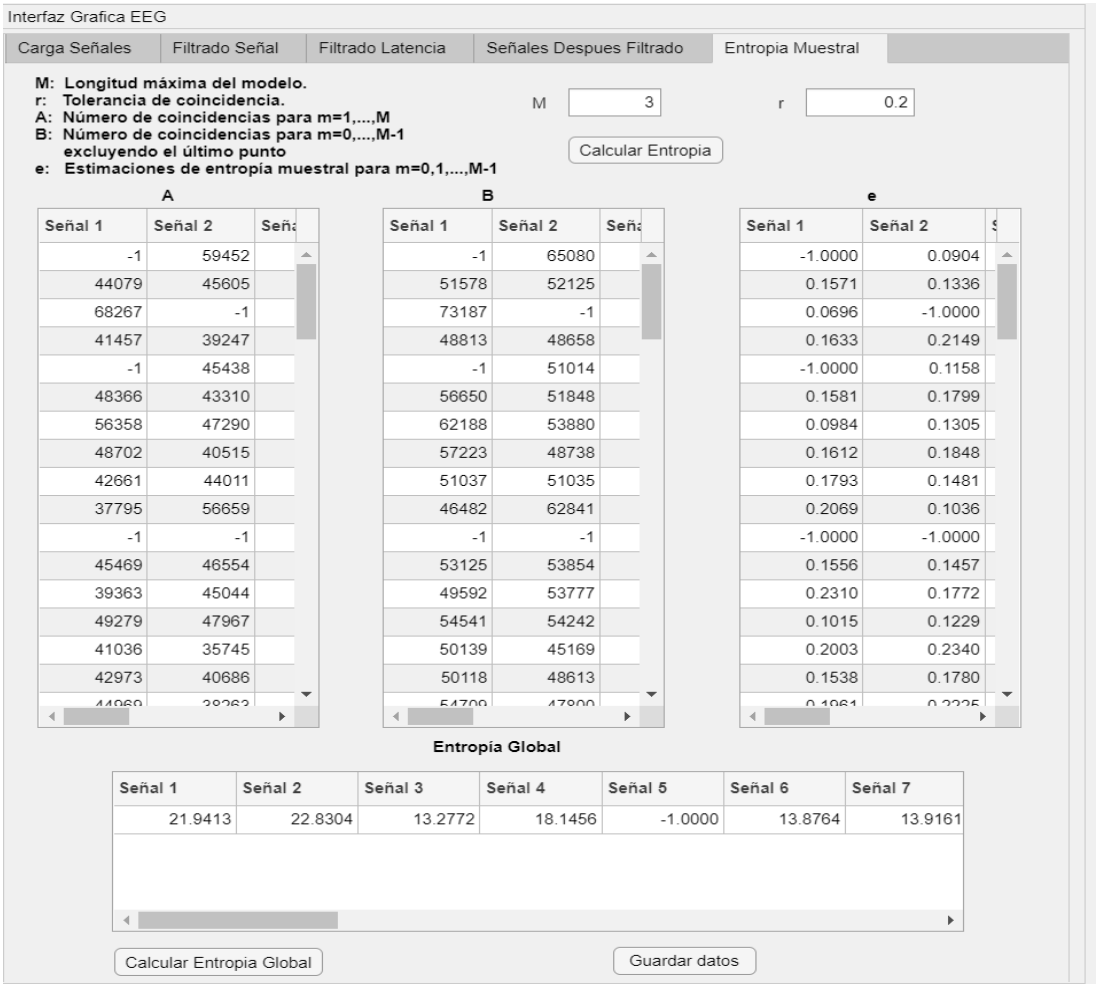
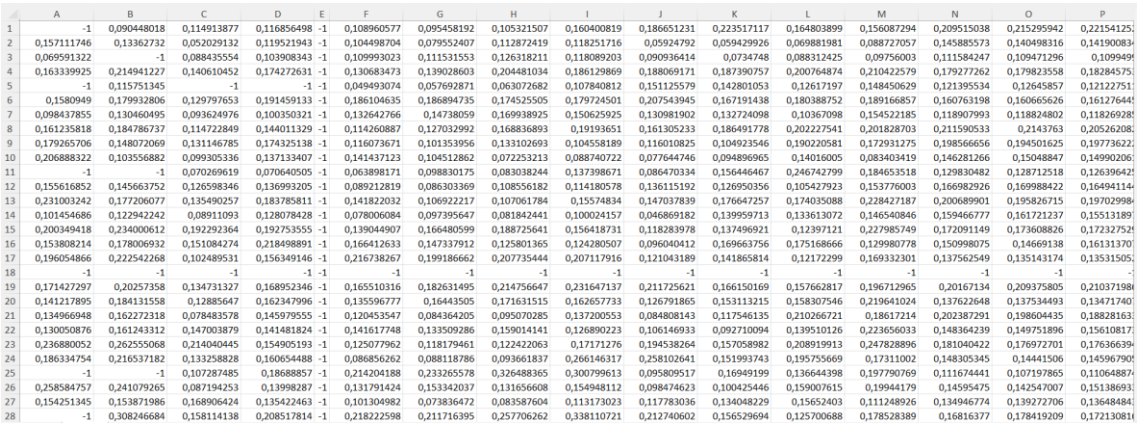


Figura 82: Entropía Muestral Completa.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	-1	0,090448018	0,114913877	0,116856498	-1	0,108960577	0,095458192	0,105321507	0,160400819	0,186651231	0,223517117	0,164803899	0,156087294	0,209515038	0,215295942	0,22154125
2	0,157111746	0,13362732	0,052029132	0,119521943	-1	0,104498704	0,079552407	0,112872419	0,118251716	0,05924792	0,059429926	0,069881981	0,088727057	0,145885573	0,140498316	0,14190083
3	0,069591322	-1	0,088435554	0,103908343	-1	0,109993023	0,111531553	0,126318211	0,118089203	0,090936414	0,0734748	0,088312425	0,09756003	0,111584247	0,109471296	0,109949
4	0,163339925	0,214941227	0,140610452	0,174272631	-1	0,130683473	0,139028603	0,204481034	0,186129869	0,188069171	0,187390757	0,200764874	0,210422579	0,179277262	0,179823558	0,18284575
5	-1	0,115751345	-1	-1	-1	0,049493074	0,057692871	0,063072682	0,107840812	0,151125579	0,142801053	0,12617197	0,148450629	0,121395534	0,12645857	0,12122751
6	0,1580949	0,179932806	0,129797653	0,191459133	-1	0,186104635	0,186894735	0,174525505	0,179724501	0,207543945	0,167191438	0,180388752	0,189166857	0,160763198	0,160665626	0,16127644
7	0,098437855	0,130460495	0,093624976	0,100350321	-1	0,132642766	0,14738059	0,169938925	0,150625925	0,130981902	0,132724098	0,10367098	0,154522185	0,118907993	0,118824802	0,11826928
8	0,161235818	0,184786737	0,114722849	0,144011329	-1	0,114260887	0,127032992	0,168836893	0,19193651	0,161305233	0,186491778	0,202227541	0,201828703	0,211590533	0,2143763	0,20526208
9	0,179265706	0,148072069	0,131146785	0,174325138	-1	0,116073671	0,101353956	0,133102693	0,104558189	0,116010825	0,104923546	0,190220581	0,172931275	0,198566656	0,194501625	0,19773622
10	0,206888322	0,103556882	0,099305336	0,137133407	-1	0,141437123	0,104512862	0,072253213	0,088740722	0,077644746	0,094896965	0,14016005	0,083403419	0,146281266	0,15048847	0,14990206
11	-1	-1	0,070269619	0,070640505	-1	0,063898171	0,098830175	0,083038244	0,137398671	0,086470334	0,156446467	0,246742799	0,184653518	0,129830482	0,128712518	0,12639642
12	0,155616852	0,145663752	0,126598346	0,136993205	-1	0,089212819	0,086303369	0,108556182	0,114180578	0,136115192	0,126950356	0,105427923	0,153776003	0,166982926	0,169988422	0,16494114
13	0,231003242	0,177206077	0,135490257	0,183785811	-1	0,141822032	0,106922217	0,107061784	0,15574834	0,147037839	0,176647257	0,174035088	0,228427187	0,200689901	0,195826715	0,19702998
14	0,101454686	0,122942242	0,08911093	0,128078428	-1	0,078006084	0,097395647	0,081842441	0,100024157	0,046869182	0,139959713	0,133613072	0,146540846	0,159466777	0,161721237	0,15513189
15	0,200349418	0,234000612	0,192292364	0,192753555	-1	0,139044907	0,166480599	0,188725641	0,156418731	0,118283978	0,137496921	0,12397121	0,227985749	0,172091149	0,173608826	0,17232752
16	0,153808214	0,178006932	0,151084274	0,218498891	-1	0,166412633	0,147337912	0,125801365	0,124280507	0,096040412	0,169663756	0,175168666	0,129980778	0,150998075	0,14669138	0,16131370
17	0,196054866	0,222542268	0,102489531	0,156349146	-1	0,216738267	0,199186662	0,207735444	0,207117916	0,121043189	0,141865814	0,12172299	0,169332301	0,137562549	0,135143174	0,13531505
18	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-
19	0,171427297	0,20257358	0,134731327	0,168952346	-1	0,165510316	0,182631495	0,214756647	0,231647137	0,211725621	0,166150169	0,157662817	0,196712965	0,20167134	0,209375805	0,21037198
20	0,141217895	0,184131558	0,12885647	0,162347996	-1	0,135596777	0,16443505	0,171631515	0,162657733	0,126791865	0,153113215	0,158307546	0,219641024	0,137622648	0,137534493	0,13471740
21	0,134966948	0,162272318	0,078483578	0,145979555	-1	0,120453547	0,084364205	0,095070285	0,137200553	0,084808143	0,117546135	0,210266721	0,18617214	0,202387291	0,198604435	0,18828163
22	0,130050876	0,161243312	0,147003879	0,141481824	-1	0,141617748	0,133509286	0,159014141	0,126890223	0,106146933	0,092710094	0,139510126	0,223656033	0,148364239	0,149751896	0,15610817
23	0,236880052	0,262555068	0,214040445	0,154905193	-1	0,125077962	0,118179461	0,122422063	0,17171276	0,194538264	0,157058982	0,208919913	0,247828896	0,181040422	0,176972701	0,17636639
24	0,186334754	0,216537182	0,133258828	0,160654488	-1	0,086856262	0,088118786	0,093661837	0,266146317	0,258102641	0,151993743	0,195755669	0,17311002	0,148305345	0,14441506	0,14596790
25	-1	-1	0,107287485	0,18688857	-1	0,214204188	0,233265578	0,326488365	0,300799613	0,095809517	0,16949199	0,136644398	0,197790769	0,111674441	0,107197865	0,11064887
26	0,258584757	0,241079265	0,087194253	0,13998287	-1	0,131791424	0,153342037	0,131656608	0,154948112	0,098474623	0,100425446	0,159007615	0,19944179	0,14595475	0,142547007	0,15138693
27	0,154251345	0,153871986	0,168906424	0,135422463	-1	0,101304982	0,073836472	0,083587604	0,113173023	0,117783036	0,134048229	0,15652403	0,111248926	0,134946774	0,139272706	0,13648484
28	-1	0,308246684	0,158114138	0,208517814	-1	0,218222598	0,211716395	0,257706262	0,338110721	0,212740602	0,156529694	0,125700688	0,178528389	0,16816377	0,178419209	0,17213081

Figura 83: Excel Generado Entropía.

Capítulo 6. Conclusiones y líneas futuras.

6.1 Conclusiones

En este proyecto se ha llevado a cabo distintos filtrados, implementaciones de algoritmos y la creación de una interfaz gráfica que haga más sencillo al usuario el uso de dichos algoritmos para el análisis de la complejidad de la señal.

Enfocándonos en los objetivos, hemos visto la utilidad de EEGLAB para realizar un preprocesado de la señal, que nos ha permitido realizar satisfactoriamente un primer filtrado, donde hemos podido detectar las señales que no nos proporcionaban información o que simplemente presentaban artefactos.

Por otro lado, en la implementación de los algoritmos de Matlab, hemos podido diseñar y programar de manera correcta algoritmos que nos han permitido detectar señales defectuosas, que a simple vista parecían correctas. También, ha sido posible comprobar que el análisis época a época es mucho más preciso para poder descartar las señales que no cumplen las condiciones deseadas por el usuario.

Además, se ha podido realizar un estudio completo de la complejidad de la señal utilizando el algoritmo de la entropía muestral.

Por último, gracias a la implementación de la interfaz gráfica, el usuario tiene a su disposición una aplicación que le permite, de manera intuitiva y sencilla, realizar todo el preprocesado de los datos y a su vez poder utilizar la técnica de procesado de entropía muestral sin la necesidad de utilizar ningún lenguaje de programación.

En definitiva, se han cumplido cada uno de los objetivos propuestos para alcanzar el objetivo final buscado en el proyecto: el desarrollo de una interfaz gráfica que facilite el uso de los algoritmos de complejidad para el análisis de las señales de EEG.

6.2 Líneas Futuras

Como líneas futuras podemos distinguir las siguientes:

- Un análisis más profundo de los valores de la entropía, ya que el proyecto queda fuera del estudio cualitativo. Por tanto, una buena línea futura sería estudiar entre señales los valores proporcionados por el algoritmo.

- Otra de las líneas futuras vendría relacionada con la interfaz cerebro-computadora (“Brain-computer interface” – BCI). El sistema BCI registra la actividad cerebral mediante electrodos específicos, tras esto se realiza un procesado de la señal donde se extraen las características más relevantes de la señal en tiempo real, es decir, interactuando con la realidad. Tras esto, se clasifican e interpretan las características obtenidas y, por último, las respuestas son enviadas al dispositivo de salida.

  Como línea futura, se puede hacer BCI con señales de complejidad para hacer un análisis de la propia complejidad.

Capítulo 7. Bibliografía

[1] D. F. Rodríguez, «Anatomía del cerebro,» 30 Abril 2019. [En línea]. Available: https://asociacioneducar.com/anatomia-cerebro.

[2] D. J. N. A. Flores, «Anatomía y fisiología del cerebro,» 12 Marzo 2013. [En línea]. Available: https://es.slideshare.net/Logan_sv/anatoma-y-fisiologa-del-cerebro.

[3] D. D. Geffner, «El cerebro organización y función,» [En línea]. Available: https://www.svneurologia.org/libro%20ictus%20capitulos/cap2.pdf.

[4] D. T. A. Pérez, «Los hemisferios cerebrales y sus funciones,» [En línea]. Available: https://www.lechepuleva.es/nutricion-y-bienestar/los-hemisferios-cerebrales-y-sus-funciones.

[5] A. Triglia, «Partes del cerebro humano (y funciones),» *Psicología y Mente,* 24 Julio 2016.

[6] J. M. Uriarte, «Cerebro,» 13 Agosto 2019. [En línea]. Available: https://www.caracteristicas.co/cerebro/.

[7] S. Silgado, «Hemisferios cerebrales derecho e izquierdo: características, funciones y diferencias,» 9 Octubre 2020. [En línea]. Available: https://www.psicologia-online.com/hemisferios-cerebrales-derecho-e-izquierdo-caracteristicas-funciones-y-diferencias-5260.html#anchor_0.

[8] M. Menéndez, «Diferencias entre sistema nervioso central y periférico,» 31 Julio 2018. [En línea]. Available: https://www.psicologia-online.com/diferencias-entre-sistema-nervioso-central-y-periferico-4002.html.

[9] Micaela, «Entendiendo el cerebro: ¿cómo funciona el sistema nervioso periférico?,» [En línea]. Available: https://www.dacer.org/entendiendo-el-cerebro-como-funciona-el-sistema-nervioso-periferico/.

[10] A. Torres, «¿Qué es el lobulo frontal y cómo funciona?,» 17 Diciembre 2015. [En línea]. Available: https://psicologiaymente.com/neurociencias/lobulo-frontal-cerebro.

[11] A. Torres, «Lóbulo Parietal: características y funciones,» 18 Octubre 2016. [En línea]. Available: https://psicologiaymente.com/neurociencias/lobulo-parietal.

[12] A. Triglia, «Lóbulo occipital: anatomía, características y funciones,» 18 Octubre 2016. [En línea]. Available: https://psicologiaymente.com/neurociencias/lobulo-occipital.

[13] O. C. Mimenza, «Lóbulo temporal: estructura y funciones,» 22 Octubre 2016. [En línea]. Available: https://psicologiaymente.com/neurociencias/lobulo-temporal.

[14] J. M. Uriarte, «Cerebelo,» 10 Marzo 2020. [En línea]. Available: https://www.caracteristicas.co/cerebelo/.

[15] Gradior, «Médula espinal: qué es, partes y cuáles son sus funciones,» 27 Julio 2021. [En línea]. Available: https://www.gradior.es/que-es-la-medula-espinal/.

[16] Equipo de la torre, «Qué es y cómo se estructura el sistema nervioso,» 24 Agosto 2021. [En línea]. Available: https://www.neurocirugiaequipodelatorre.es/que-es-y-como-se-estructura-el-sistema-nervioso.

[17] C. S. MD, «Bulbo raquídeo (médula oblongada),» 21 Marzo 2022. [En línea]. Available: https://www.kenhub.com/es/library/anatomia-es/bulbo-raquideo-medula-oblongada.

[18] El blog de la Fundación Pasqual Maragall, «¿Cómo es y cómo funciona nuestro cerebro?,» 11 Julio 2021. [En línea]. Available: https://blog.fpmaragall.org/como-es-y-como-funciona-nuestro-cerebro.

[19] A. Z. Fernandes, «Sistema nervioso,» 2 Junio 2022. [En línea]. Available: https://www.significados.com/sistema-nervioso/.

[20] R. Pacientes, «Sistema Nervioso: Concepto y Funciones,» [En línea]. Available: https://rochepacientes.es/esclerosis-multiple/sistema-nervioso.html.

[21] B. Szymik, «¿Qué hay en tu cerebro?,» 31 Mayo 2017. [En línea]. Available: https://askabiologist.asu.edu/que-hay-en-tu-cerebro.

[22] Á. C. Domínguez, «Procesamiento de señales para la detección anticipada de Crisis Epilépticas en el EEG,» 2017. [En línea]. Available: https://biblus.us.es/bibing/proyectos/abreproy/91168/fichero/MEMORIA+TFG.pdf.

[23] R. B. Navarro, «Instrumentación Biomédica,» [En línea]. Available: http://www.hca.es/huca/web/enfermeria/html/f_archivos/electroencefalografia.pdf?fbclid=IwAR3phj1ifwaCuPO8-udtqTg3AEHP716M-DzemN_ohP1RP7v9dXYLeBK76wQ.

[24] Mayo Clinic, «EEG (electroencefalograma),» [En línea]. Available: https://www.mayoclinic.org/es-es/tests-procedures/eeg/about/pac-20393875#:~:text=Un%20electroencefalograma%20es%20una%20prueba,el%20tiempo%2C%20incluso%20mientras%20duermes..

[25] TOPDOCTORS, «Electroencefalograma,» [En línea]. Available: https://www.topdoctors.es/diccionario-medico/electroencefalograma#.

[26] L. Palacios, «Breve historia de la electroencefalografía,» 18 Febrero 2002. [En línea]. Available: http://www.acnweb.org/acta/2002_18_2_104.pdf.

[27] T. T. García, «Manual básico para enfermeros en electroencefalografía,» 2011. [En línea]. Available: http://www.sspa.juntadeandalucia.es/servicioandaluzdesalud/huvvsites/default/files/revistas/ED-094-07.pdf.

[28] Sinc Salud, «Las ondas delta de la corteza cerebral intervienen en la toma de decisiones,» 25 Septiembre 2013. [En línea]. Available: https://www.agenciasinc.es/Noticias/Las-ondas-delta-de-la-corteza-cerebral-intervienen-en-la-toma-de-decisiones.

[29] NeuroFeedBack, «¿Qué son las ondas cerebrales?,» 24 Julio 2019. [En línea]. Available: https://www.neurofeedback.cat/que-son-las-ondas-cerebrales/.

[30] E. Sánchez, «¿Qué sabemos de las misteriosas ondas delta?,» 1 Febrero 2022. [En línea]. Available: https://lamenteesmaravillosa.com/que-sabemos-de-las-misteriosas-ondas-delta/.

[31] Bitbrain, «Colocación de electrodos EEG en un Layout Fijo vs. Variable,» 30 Abril 2020. [En línea]. Available: https://www.bitbrain.com/es/blog/colocacion-electrodos-eeg.

[32] V. G. Gil, «Identificación biométrica basada en el electroencefalograma,» 2017. [En línea]. Available: https://idus.us.es/bitstream/handle/11441/66516/TFG_Victor%20Gonz%C3%A1lez%20Gil.pdf?sequence=1&isAllowed=y.

[33] P. Eduardo López-Larraz, «What is BCI? An introduction to brain-computer interface using EEG signals,» 3 March 2020. [En línea]. Available: https://www.bitbrain.com/blog/brain-computer-interface-using-eeg-signals.

[34] L. J. G. Figueroa, «Análisis de señales EEg para detección de eventos oculares, musculares y cognitivos,» 12 Septiembre 2016. [En línea]. Available: https://oa.upm.es/44379/1/TFM_LEONARDO_JOSE_GOMEZ_FIGUEROA.pdf.

[35] A. P. Escudero, «Complejidad cerebral explicada mediante leyes sencillas,» Marzo/Abril 2010. [En línea]. Available: https://www.investigacionyciencia.es/revistas/mente-y-cerebro/dormir-para-aprender-500/complejidad-cerebral-explicada-mediante-leyes-sencillas-853.

[36] O. H. M. Ross, «ResearchGate,» [En línea]. Available: https://www.researchgate.net/figure/Posicion-de-los-electrodos-acorde-al-Sistema-Internacional-10-20-Figura-vista-desde-a_fig1_274457110.

[37] R. B. Navarro, «Instrumentación Biomédica,» [En línea]. Available: https://www.pardell.es/electroencefalografo.html.

[38] M. E. A. Perona, «Elecroencefalografía,» [En línea]. Available: http://dea.unsj.edu.ar/bioinstrumentacion2/apunteseeg.pdf.

[39] BANANA SOFT, «Un entorno de código abierto para el procesamiento de señales electrofisiológicas,» [En línea]. Available: https://banana-soft.com/es/un-entorno-de-codigo-abierto-para-el-procesamiento-de-senales-electrofisiologicas.

[40] G. C. Espinosa, «Programación de interfaz gráfica en app designer para el control vectorial de motores de imanes permanentes,» Julio 2018. [En línea]. Available: https://oa.upm.es/53343/1/TFG_GUILLERMO_CID_ESPINOSA.pdf.

[41] S. B. Gómez, «Desarrollo de un sistema para análisis de señales electroencefalográficas,» 19 Junio 2017. [En línea]. Available: https://oa.upm.es/52807/1/TFG_SERGIO_BLANCO_GOMEZ.pdf.

[42] C. d. l. F. Gutiérrez, «Eliminación de artefactos cardiacos en señales de electroencefalograma mediante filtro adaptativo,» Marzo 2020. [En línea]. Available: https://oa.upm.es/67402/1/TFG_CLAUDIA_DE_LA_FUENTE_GUTIERREZ.pdf.

[43] J. M. a. C. H. DK Lake (dlake at virginia dot edu), «Sample Entropy Estimation Using SAMPEN,» [En línea]. Available: https://archive.physionet.org/physiotools/sampen/.

[44] M. Costa, «Background,» 24 Junio 2005. [En línea]. Available: https://archive.physionet.org/physiotools/mse/tutorial/node1.html.

[45] M. A. Muñoz, «Affective Modulation of Brain and Autonomic Responses in Patients With Fibromyalgia,» [En línea]. Available: https://www.academia.edu/34150655/Affective_Modulation_of_Brain_and_Autonomic_Responses_in_Patients_With_Fibromyalgia.

[46] Sanitas, «Taller Teórico-Práctico de electroencefalografía y polisomnografía,» 2014. [En línea]. Available: https://www.sanitas.es/media/profesionalessanitarios/doc/archivo/pdf_taller_electroencefalografia_polisomnografia/son280042.pdf.

[47] Neuromarketing, «4 tecnologías que se usan en estudios Neuromarketing,» [En línea]. Available: https://neuromarketing.la/2015/12/herramientas-utilizadas-por-el-neuromarketing/.

[48] Pixabay, [En línea]. Available: https://pixabay.com/es/images/search/electrodos%20adheridos/.

Capítulo 8. Anexo

8.1 Introducción a App Designer

App Designer es un entorno de desarrollo interactivo de Matlab que permite el diseño y la programación de una aplicación. Este entorno permite la programación rápida gracias a la posibilidad de utilizar un editor integrado que permite colocar los componentes visuales directamente en la pantalla.

En primer lugar, existen dos formas de acceder a App Designer.

1) Dentro del Command Windows de Matlab, escribiendo *appdesigner*.

2) En la pestaña *Apps* – *Design App*. (**Ver Figura 84**).

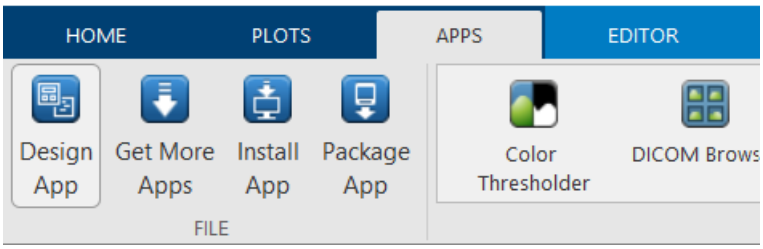


Figura 84: Acceso App Designer.

Tras acceder al entorno de App Designer, podemos diferenciar dos entornos diferentes de trabajo: ***Design View y Code View.*** Se puede apreciar en la **Figura 85:**

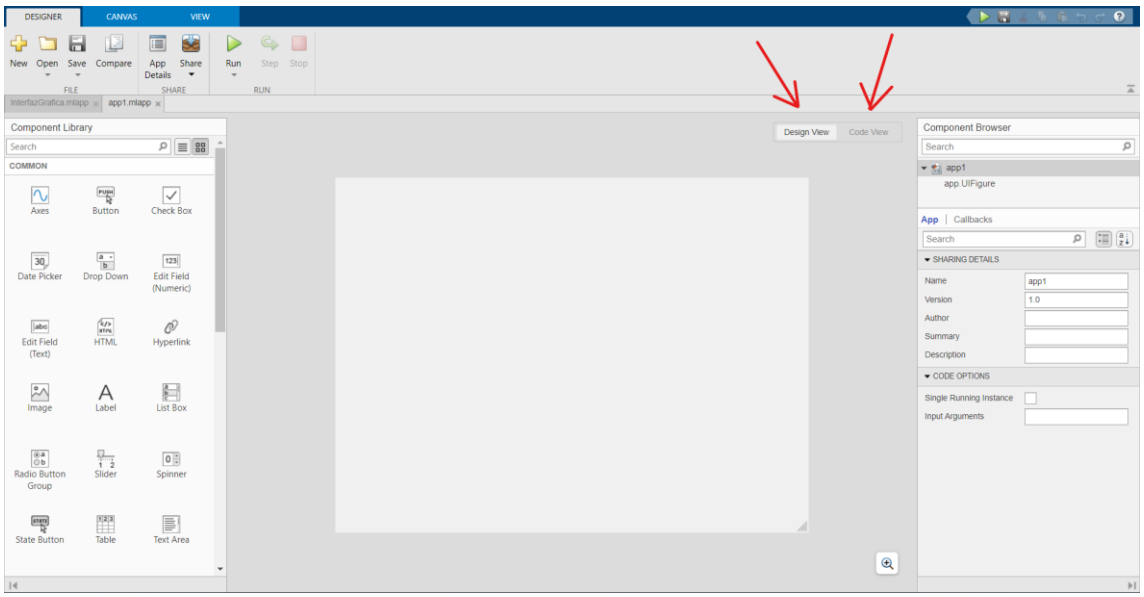


Figura 85: Entorno de trabajo.

Dentro del entorno de ***Design View*** podemos distinguir los siguientes paneles y funciones. Se puede ver en la **Figura 86:**

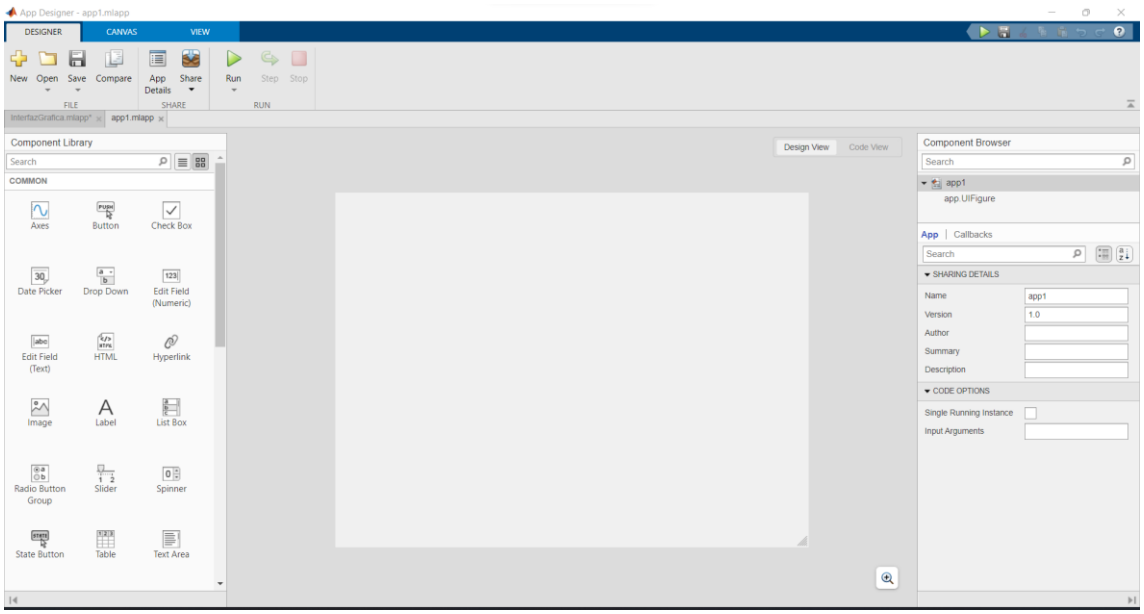
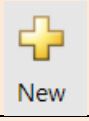
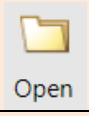
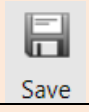
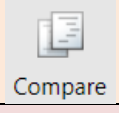
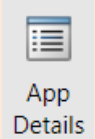


Figura 86: Entorno Design View.

1) **TOOLSTRIP**: Esta parte es donde se encuentran todas las herramientas y se divide en tres pestañas. (Ver **Tabla 3, Tabla 4 y Tabla 5)**

Pestaña DESIGNER	
File	
	**New.** La función es crear una nueva aplicación.
	**Open.** Abre aplicaciones ya existentes.
	**Save.** Guarda las modificaciones realizadas en la aplicación.
	**Compare.** Permite realizar comparaciones de dos aplicaciones diferentes.
Share	
	**App Details.** Permite ver los detalles de la aplicación.

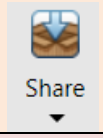
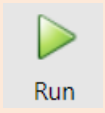
	**Share.** Crea un documento para compartir la aplicación con otros usuarios.
Run	
	**Run.** Ejecuta la aplicación.

Tabla 3: Herramientas Pestaña Designer.

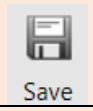
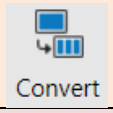
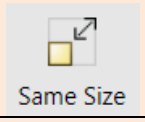
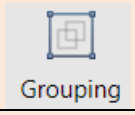
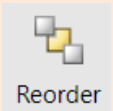
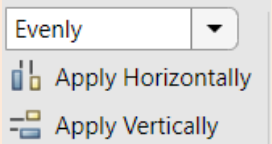
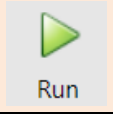
Pestaña CANVAS	
File	
	**Save.** Guarda las modificaciones realizadas en la aplicación.
	**Convert.** Convierte una aplicación en 2 o 3 paneles con auto reflujo, es decir, la aplicación proporciona reflujo automático y cambio de tamaños.
Align	
	**Align.** Alinea todos los componentes seleccionados dentro de la aplicación.
Arrange	
	**Same Size.** Permite igualar las dimensiones de los componentes que se han seleccionado.
	**Grouping.** Permite agrupar los componentes que se han seleccionado.
	**Reorder.** Permite reordenar los componentes que se han seleccionado.
Space	
	**Space.** Esta función permite distribuir los componentes que se han seleccionado. Tiene dos formas de hacerlo, de forma uniforme (evently) o introduciendo la distancia que desea el usuario dentro de la casilla.
Run	
	**Run.** Ejecuta la aplicación.

Tabla 4: Herramientas Pestaña Canvas.

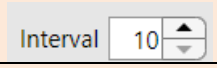
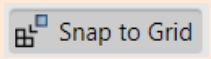
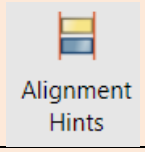
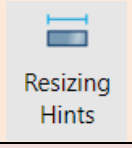
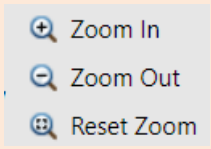
Pestaña VIEW	
Display	
	**Grid.** Permite activar la opción de vista en cuadrícula.
	**Interval:** Se utiliza para modificar el tamaño de la matrícula.
	**Snap to Grid:** Permite ajustar la posición de las líneas de la cuadrícula.
	**Alignment Hints.** Ayuda al ajuste de la alineación de los componentes.
	**Resizing Hints.** Ayuda al ajuste de las dimensiones de los componentes.
Zoom	
	**Fit to View.** Ajusta el diseño en la pantalla de diseño.
	**Zoom.** Control del zoom de la pantalla de diseño.

Tabla 5: Herramientas Pestaña View.

2) **COMPONENT LIBRARY:** En esta parte se encuentran los diferentes componentes que se pueden utilizar en el desarrollo del diseño de la aplicación. Tiene distintos tipos: common, containers, figure tools y instrumentation. Para pode rutilizarlos, basta con arrastrarlos al editor del entorno de ***design view.*** Se puede ver en la **Figura 87:**

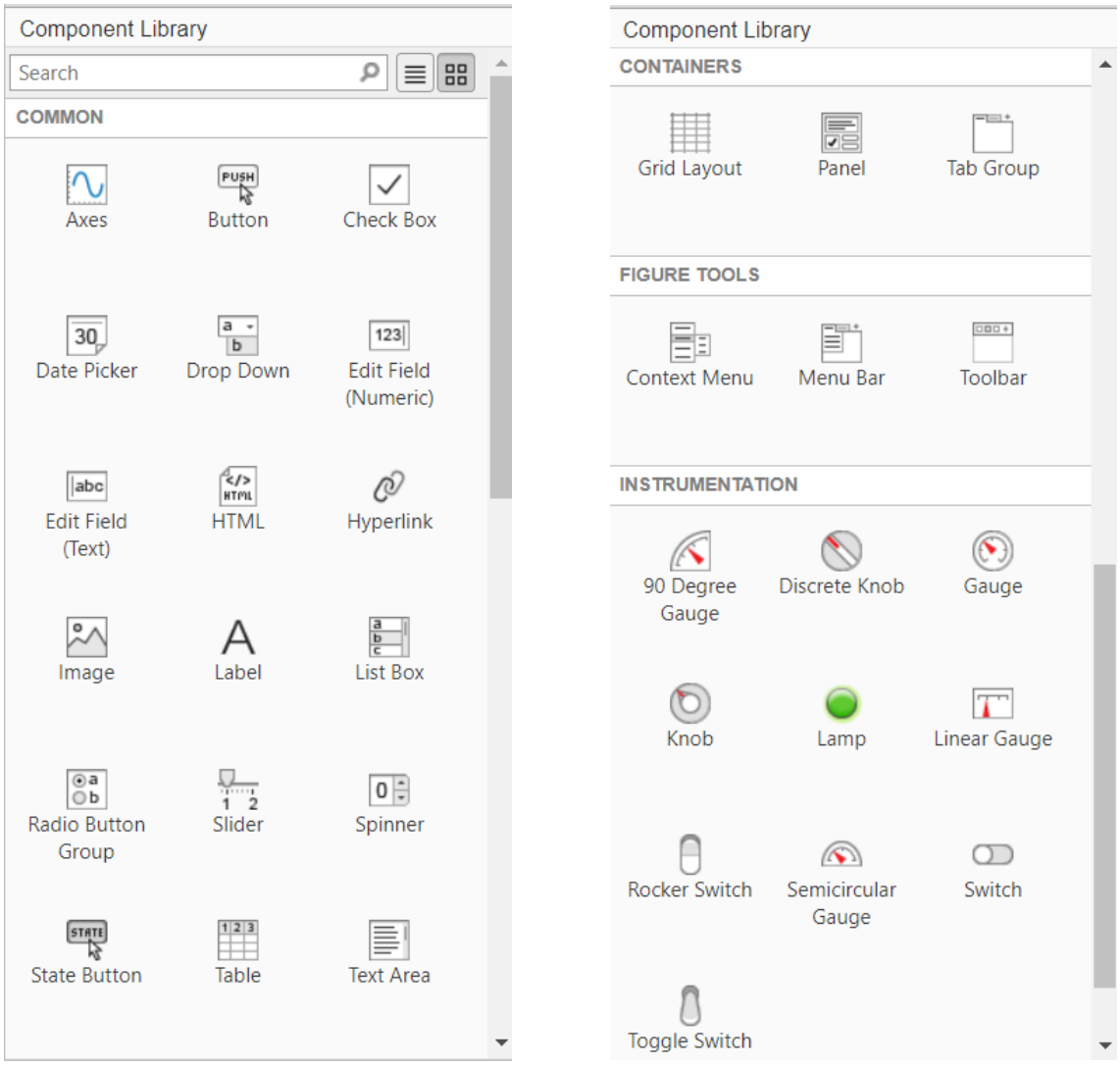


Figura 87: Component Library.

3) **DESIGN EDITOR:** Es el área donde se arrastran los diferentes elementos de control y constituye una representación de lo que se verá en la pantalla al ejecutar la aplicación. Se puede ver en la **Figura 88:**

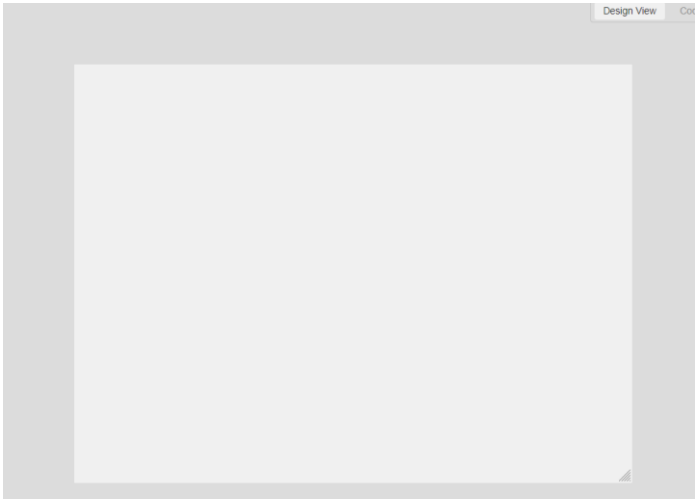


Figura 88: Design Editor.

4) **COMPONENT BROWSER:** Es la última parte del entorno. En primer lugar, aparece una lista que se genera al crear diferentes objetos en la aplicación. Además, contiene una parte de propiedades, donde puedes ver las características de los componentes utilizados y modificarlas. Se puede ver en la **Figura 89:**

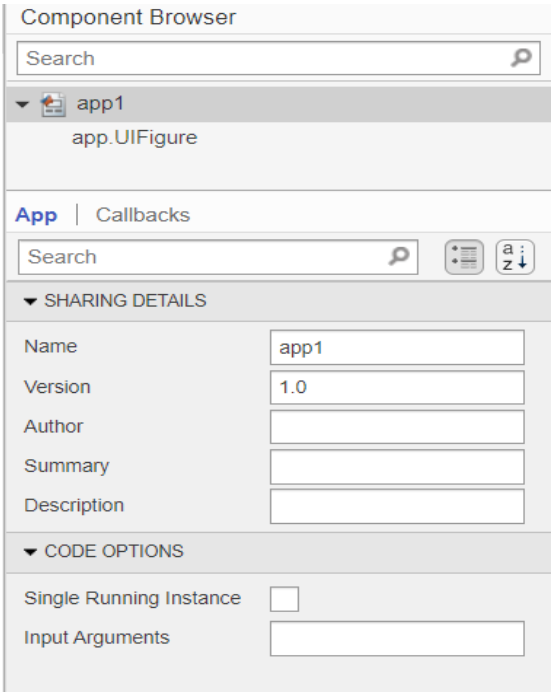
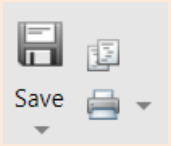
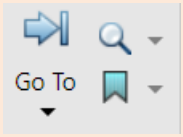
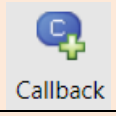
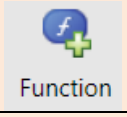
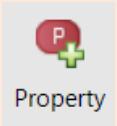
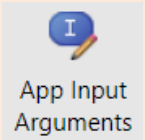
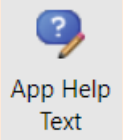


Figura 89: Component Browser.

Desde el entorno de ***code view*** podemos distinguir los siguientes paneles y funciones:

1) **TOOLSTRIP**: Esta parte es donde se encuentran todas las herramientas y se divide en tres pestañas. (Ver **Tabla 6 y Tabla 7)**

La primera es la pestaña **DESIGNER**, ya vista anteriormente en el anterior entorno.

Pestaña EDITOR	
File	
	**Save.** Guarda las modificaciones realizadas en la aplicación. Y además incluye la opción de imprimir y de comparar con otras aplicaciones.
Navigate	
	**Go to.** Permite localizar la línea de código o la función que se indique. Además, con la lupa podemos encontrar en el código las palabras seleccionadas. La otra herramienta, es para poner marcadores dentro del código.
Insert	
	**Callback.** Permite crear un callback.
	**Function.** Permite crear una función.
	**Reorder.** Permite crear una propiedad
	**Reorder.** Permite a la aplicación tomar argumentos de entrada.
	**Reorder.** Permite añadir texto de ayuda a la aplicación.
Code	
	**Comment.** Crea y elimina comentarios.

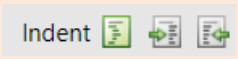
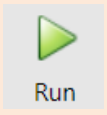
	**Indent.** Permite introducir sangría en el código.
Run	
	**Run.** Ejecuta la aplicación.

Tabla 6: Herramientas Pestaña Editor.

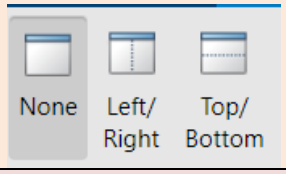
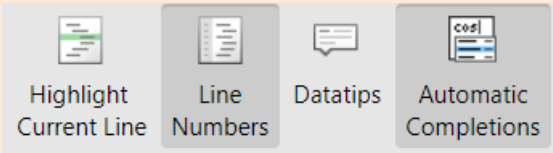
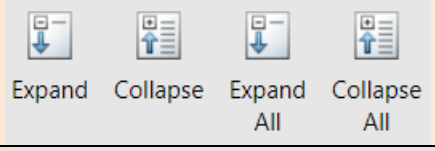
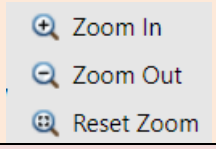
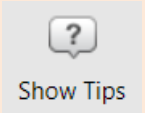
Pestaña VIEW	
Split Document	
	**Split Document.** Permite dividir el documento a la hora de programar.
Display	
	**Display.** Es un monitor que contiene funciones como mostrar el número de la línea del código, líneas vacías, consejos de datos y terminar automáticamente el código.
Code folding	
	**Code Folding.** Permite contraer o extender el código que se está implementando.
Zoom	
	**Zoom.** Control del zoom de la pantalla de diseño.
Resources	
	**Show Tips.** Habilita los consejos en el code view.

Tabla 7: Herramientas Pestaña View.

2) **CODE BROWSER:** Muestra los diferentes callbacks, funciones y propiedades creados dentro de la aplicación. Se puede ver en la **Figura 90:**

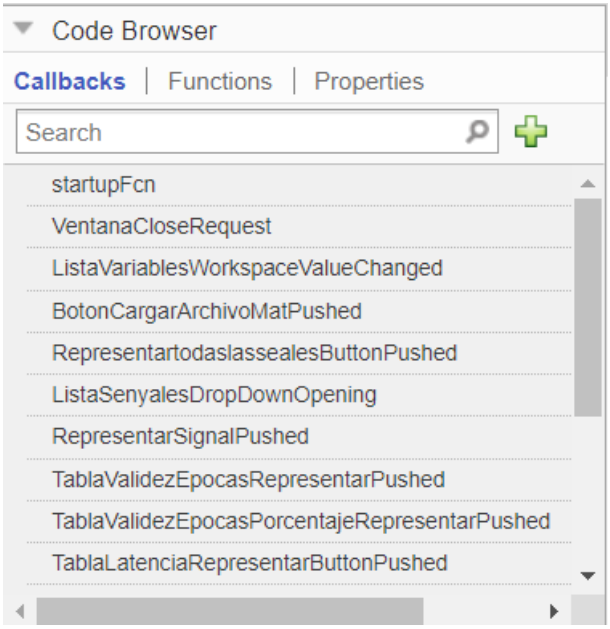


Figura 90: Code Browser.

3) **APP LAYOUT:** Es una vista de cómo están situados los distintos componentes que forman la aplicación creada. Se puede ver en la **Figura 91:**

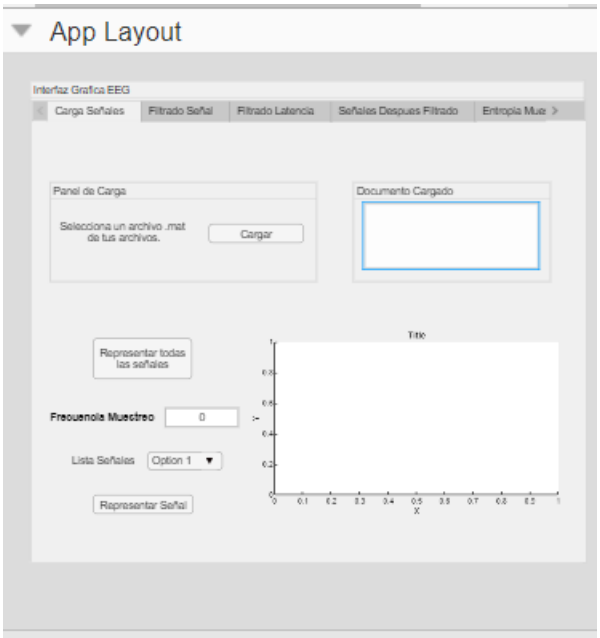


Figura 91: App Layout.

4) **CODE EDITOR:** Donde se realiza la implementación del código de la aplicación. Se puede ver en la **figura 92:**

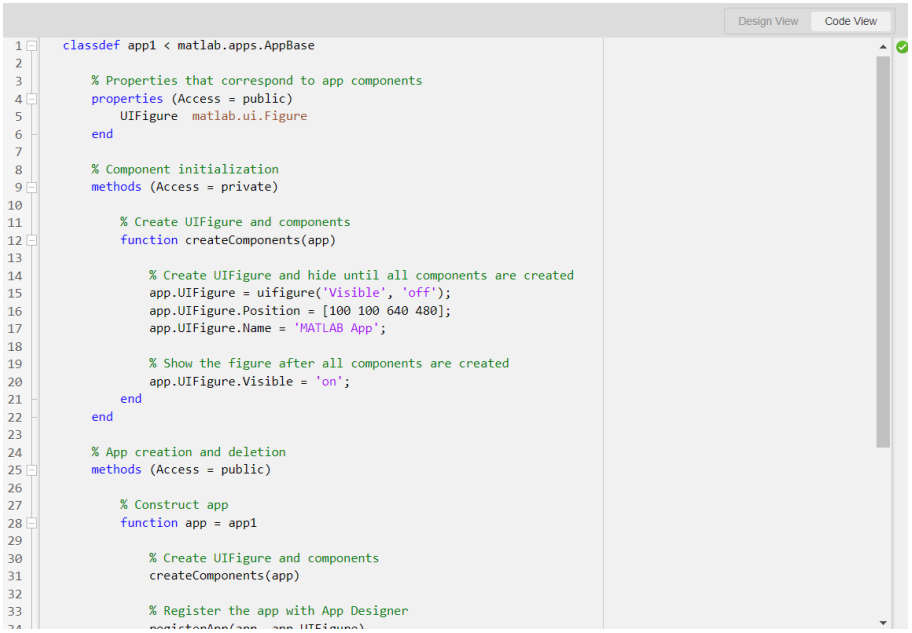


Figura 92: Code Editor.

5) **COMPONENT BROWSER:** Al igual que la anterior, ventana que contiene aparte de los diferentes componentes creados en la aplicación, distintas propiedades sobre los componentes que pueden ser modificadas. Se puede ver en la **figura 93:**

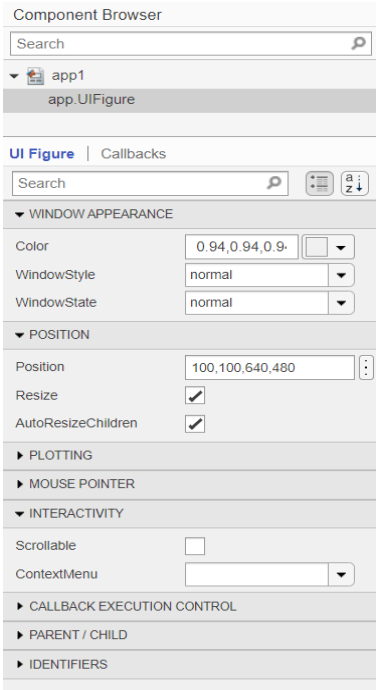


Figura 93: Component Browser.

Tras esto, se procede a explicar los botones utilizados en nuestro proyecto y la función que contiene cada uno de ellos. (Ver **Tabla 8**)

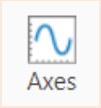
Componentes	
	**Axes.** Se utiliza para representar datos.
	**Button.** Los Button ejecutan un callback cuando se pulsa sobre ellos.
	**Drop Down.** Permite elegir entre dos o más opciones dentro de la aplicación.
	**Edit Field (Numeric).** Permite introducir datos de tipo numérico.
	**Edit Field (Text).** Permite introducir datos de tipo texto.
	**Label.** Se utiliza para poner información sobre la aplicación.
	**Table.** Se utiliza para representar datos en tablas.
	**Text Area.** Permite introducir datos de tipo texto.
	**Panel.** Se utiliza para agrupar todos los contenidos de la aplicación.
	**Tab Group.** Se utiliza para crear distintas pestañas de información dentro de la aplicación.

Tabla 8: Componentes App Designer.