Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERIA DE COMPUTADORES

Trabajo Fin de Grado

Implementación de una interfaz cerebro – ordenador mediante la conexión de Unicorn Hybrid Black y Raspberry Pi

Autor: Munteanu Sebastian

Tutor : José Luis Martin Sánchez

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Computadores

Trabajo Fin de Grado

Implementación de una interfaz cerebro – ordenador mediante la conexión de Unicorn Hybrid Black y Raspberry Pi

Autor: Sebastian Munteanu

Tutor/es: José Luis Martín Sánchez

TRIBUNAL:

Presidente: Carlos Julián Martín Arguedas

Vocal 1º: Alejandro Martínez Arribas

Vocal 2º: José Luis Martín Sánchez

FECHA: 27-09-2021

Al tutor por marcar el camino que he de seguir, a Martin Walchshofer por el soporte proporcionado con el Unicorn Hybrid Black y a German Esteban por los registros realizados.

Gracias

1. Resumen

El proyecto presente, consiste en el desarrollo e implementación de una conexión para el análisis de la actividad cerebral[1] entre una Raspberry Pi[2] y el casco Unicorn Hybrid Black([3] [4]). Esta conexión se realiza mediante Bluetooth, pudiendo conectar los dispositivos hasta una distancia máxima de diez metros. Utilizando dicho casco se realizará la lectura de ocho electrodos de los cuales obtendremos ese mismo número de señales encefalografías. Estos registros se enviarán a un equipo donde utilizando MATLAB, serán leídas e interpretadas. Dicha conexión funciona usando el Sistema Operativo Robotico, ROS[5], funcionando este mismo con el Protocolo de Internet, IP.

Palabras Clave

Raspberry Pi, ROS, MATLAB, Unicorn Hybrid Black.

2. Abstract

The present project consists of the development and implementation of a connection between the Unicorn Hybrid Black helmet and a Raspberry Pi. This connection is made using Bluetooth, which enables a connection with up to a maximum distance of ten meters. Using this helmet, the reading of eight electrodes will be made from which we can obtain the same number of encephalographic signals. These records will be sent to a computer where using MATLAB, they will be read and interpreted. This connection works using the Robotic Operating System, ROS, establishing the connection with the Internet Protocol, IP.

Keywords

Raspberry Pi, ROS, MATLAB, Unicorn Hybrid Black.

3. Resumen Extendido

El proyecto se origina como una propuesta para una posible implementación de conexión entre el casco Unicorn Hybrid Black y una Raspberry Pi, para la obtención de ocho posibles registros EEG. Estos se obtienen mediante el mismo número de electrodos colocados sobre la piel, penetrando el pelo mediante unos dientes que se encuentran en el extremo de los electrodos.

Dichos registros EEG, tienen una frecuencia de adquisición de 250Hz y un 24 Bits de resolución[6]. Estos electrodos también se pueden utilizar con un gel conductor para aumentar la calidad.

Debido a estas características se obtienen registros con una relación señal-ruido elevada utilizando un método no intrusivo.

Estos registros se recibirán en la Raspberry mediante el protocolo Bluetooth, donde serán encapsulados en un array para su futura transmisión.

Esta última, se hará mediante ROS. Se utilizando un talker [7] que publicara en un string los contenidos de los canales durante el tiempo de la adquisición. La cantidad de datos transmitida serán 4500 datos por cada segundo de adquisición que se ejecute, debido a la frecuencia de adquisición y la cantidad de canales existentes, es decir 18.

De manera normal, ROS funciona mediante entornos de trabajo, también conocidos como workspace. Durante para facilitar las cosas, se tratará a este sistema como una serie de bibliotecas, por tanto, durante la compilación será necesaria la mención de estas.

Estos datos serán recibidos en otro equipo donde se utilizará la toolbox ROS de MATLAB[8] para poder recibir los datos y empezar a analizarlos.

Para obtener conclusiones de los datos se aplicará un filtro paso banda Butterworth de segundo orden con unos valores de corte inferior de 0.5Hz y de corte superior de 30Hz. Esto se realiza para extraer del EEG a las ondas delta, theta, alfa y beta.

Otro filtrado que se aplicara es un filtrado notch Butterworth de segundo orden de 50Hz para eliminar el ruido debido al campo electromagnético. Dicho filtro se debería de cambiar a 60Hz si la localización de la adquisición es EE. UU.

El último paso para la obtención de los registros es eliminar los datos que no estén en el intervalo de [-100,100] microvoltios.

Por el otro lado, se han realizado las pruebas con dos modelos diferentes de Raspberry, utilizando una Model 4 B y una Raspberry Pi Zero W.

En la primera de estas se ha instalado el sistema operativo Rasbian con su versión Desktop, mientras que, en la segunda, se ha utilizado su versión headless. Esto es debido a la poca potencia que presenta la placa y los problemas que esto implica.

Debido a esto, se ha utilizado el protocolo SSH para controlar de maneara inalámbrica la placa.

4. Tabla de contenido

Contents

5. Índice de Figuras

6. Glosario de acrónimos y abreviaturas

ROS	Robot Operating System	Sistema Operativo robótico
IP	Internet Protocol	Protocolo de Internet
EEG	Electroencephalography	Electroencefalografía
ADC	Analog-to-Digital Converter	Conversor Analógico Digital
LED	Light emitting diode	Diodo emisor de luz
OS	Operating System	Sistema Operativo
RAM	Random access memory	Memoria de acceso aleatorio

7. Memoria

7.1 Introducción

Durante el transcurso de este proyecto se utilizarán diferentes dispositivos, entre los cuales se encuentran el dispositivo Unicorn Hybrid Black, una Raspberry Pi y un ordenador.

Empezando por el primero de estos dispositivos, nos podemos encontrar con un sistema completo de adquisición de señales electroencefalográficas portátil. Esto supone un avance muy notable en el mundo de la electroencefalografía.

Este sistema fabricado y distribuido por la empresa austriaca g.tec se vende preparado para trabajar con un ordenador y las aplicaciones distribuidas por la citada empresa, pero a nosotros nos ha interesado utilizarlo con o una interfaz cerebro computadora que pudiera en un futuro, controlar, por ejemplo, una silla de ruedas. Para que el Unicorn pueda realizar esta función hemos querido desarrollar un sistema que recoja las señales EEG, las procese y se comunique con un sistema robótico lo más genérico posible. De ahí que utilicemos una Raspberry Pi y publiquemos los datos mediante ROS, framework de programación de robots ampliamente estandarizado hoy en día. Utilizando dicho dispositivo podemos obtener los registros en un intervalo de tiempo inferior comparado que se utilizaban anteriormente.

El segundo de estos dispositivos es una Raspberry Pi. Esta se ha utilizado para la captación de los registros y su futura comunicación con el último dispositivo. Se ha decidido utilizar esta placa debido al bajo coste que supone y a la versatilidad que nos proporcione.

Por último, el ordenador utilizado será el encargado de procesar los datos transmitidos y aplicar los filtrados necesarios.

Estos componentes se pueden observar en el diagrama de bloques de las Figuras 1 y Figura 2. Siendo la primera de ellas la conexión online y la segunda la conexión offline. En la primera se creará la comunicación con el ordenador mientras que en la segunda se escribirán los datos en un fichero.

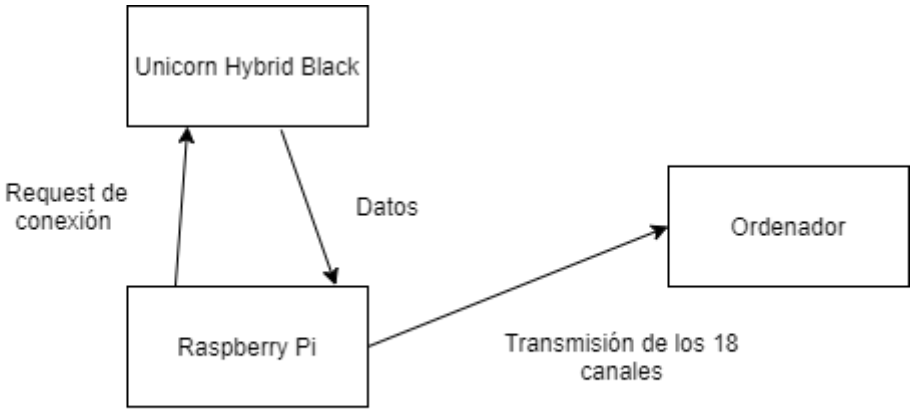


Figura 1 Conexión online de los componentes

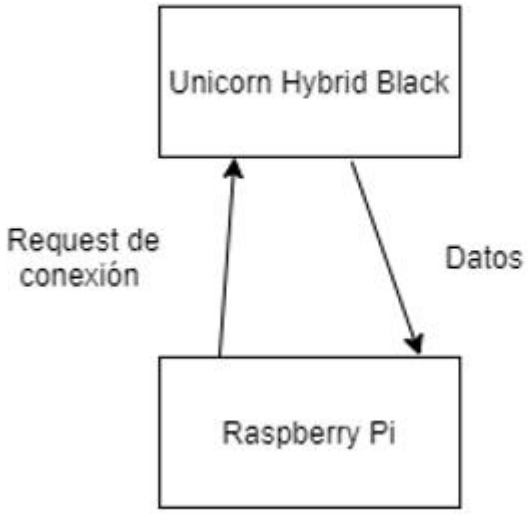


Figura 2 Conexión offline de los componentes

Durante el transcurso de los siguientes apartados se explicarán los pasos que se han realizar para la correcta ejecución de la conexión entre todos los componentes, así como los pasos necesarios para configurarlos.

7.2 Preparación Hardware

7.2.1 Preparación Unicorn Hybrid Black

Iniciando con el componente más importante, nos encontramos con el Unicorn Hybrid Black. Este sistema está compuesto por ocho canales de adquisición de los registros EEG's, los cuales se corresponden con el mismo numero de electrodos, tres canales de acelerómetro, tres canales de giróscopo. Este sistema se ha de montar en una malla especifica de acuerdo a la metodología especificada a continuación.

Durante este apartado se especifican los siguientes pasos que se han de seguir para montar el dispositivo Unicorn Hybrid Black:

- Montaje de los electrodos en la malla
- Posición de los electrodos utilizando el sistema 10-20
- Asignación de los electrodos en la malla

En el Unicorn Hybrid Black se pueden observar ocho electrodos de goma de gaucho para la toma de los EEG, dos clips que han de ser conectado con paneles adhesivos a los huesos mastoideos, un LED que permite ver el estado actual del casco y el botón encendido/apagado. Esto se puede apreciar en la Figura 3.

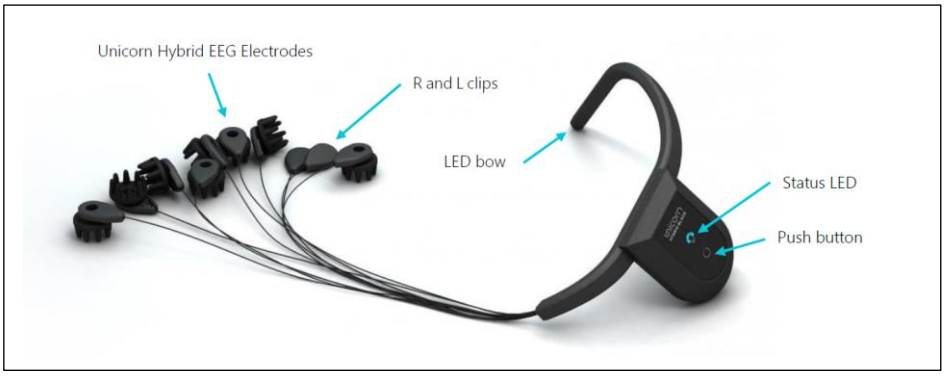


Figura 3 Componentes Unicorn Hybrid Black

Empezando con los electrodos cuya tarea es la toma de los registros EEG, estos han de ser introducidos, hasta el segundo anillo en los agujeros de la malla proporcionada por el distribuidor del casco.

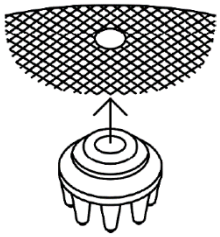


Figura 4 Montaje en la malla

Estos dos anillos junto a los agujeros donde se montan se pueden apreciar en la Figura 4.

El siguiente paso, es la conexión de los cables del Unicorn Hybrid Black con la malla.

Para la obtención de unos registros óptimos se

utiliza el sistema 10-20[9], el cual nos indica la posición en la que se deben de encontrar los electrodos. Los agujeros del casco se corresponden con las posiciones Fz, C3, Cz, C4, Pz, PO7, Oz and PO8 de la Figura 5.

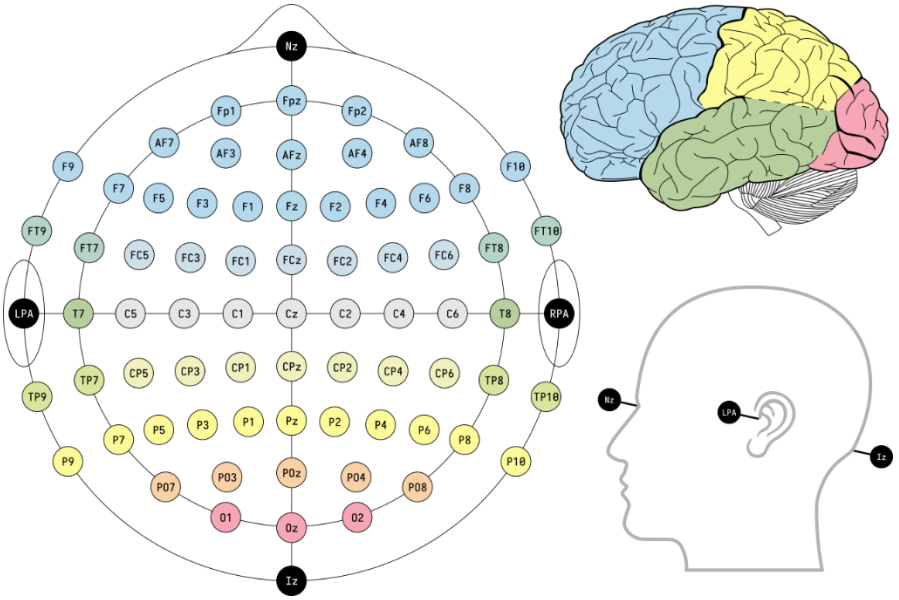


Figura 5 Posiciones del sistema 10-20

A continuación, se ha de unir los cables del Unicorn Hybrid Black con los electrodos de la malla.

En la Figura 6, se muestra la posición en la que han de ser colocados tanto los cables correspondientes a los ocho electrodos, como los clips pegajoso L y R. Se recomienda aplicar gel conductivo a los ocho electrodos para aumentar la calidad de los electrodos.

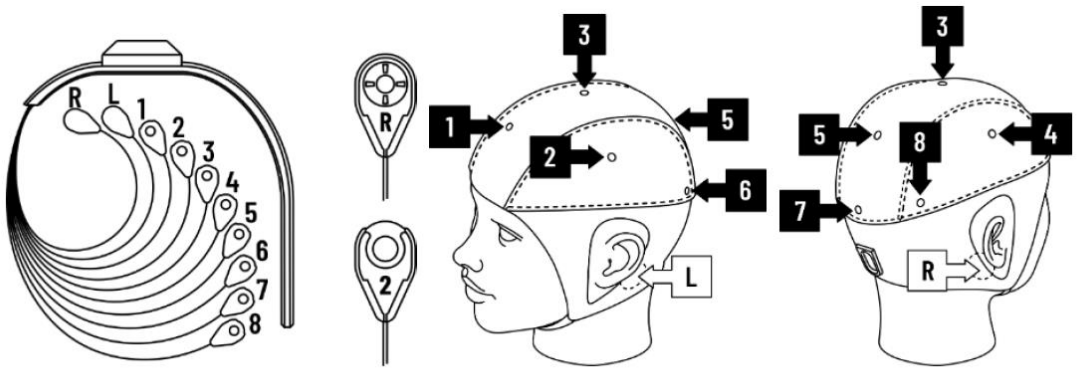


Figura 6 Posición de los electrodos en la malla

Visualizando el LED del estado podemos obtener el nivel de batería observando el color de dicho diodo. El color cian corresponde a un nivel de batería alto, el amarillo a un nivel bajo, el rojo a un nivel nulo y el color verde corresponde con el estado de carga.

También podemos visualizar la operación que está realizando segundo la acción que está realizando. Si se encuentra brillando se corresponde a un estado sin conexión bluetooth, parpadeando lentamente corresponde a una conexión abierta bluetooth, un parpadeo continuo muestra que se está llevando a cabo la adquisición de los datos, por último, si el LED no brilla, significa que el casco esta apagado.

Por último, para encender y apagar el casco se ha de mantener presionado el botón de encendido durante dos segundos.

7.2.2 Preparación Raspberry 4

En este apartado el único punto que se tratará será la instalación del sistema operativo dentro de la Raspberry Pi.

En primer lugar, se ha de formatear una tarjeta de memoria microSD con el formato de fichero FAT32. A continuación, se ha de grabar el OS Raspbian en dicha tarjeta. Para realizar esto, se puede utilizar una herramienta como balenaEtcher [11]. Al terminar dicho proceso, se puede introducir la microSD en la Raspberry para empezar el proceso de preparación software.

Durante las pruebas una de las placas utilizadas ha sido una Raspberry Pi 4 modelo B con una capacidad de 4GB de RAM y 64GB de almacenamiento. Existe la posibilidad de utilizar otro modelo mientras disponga de un puerto jack de 3.5mm, conexión bluetooth y los diferentes puertos necesarios para la conexión con los dispositivos de entrada y salida, es decir, ratón, teclado y pantalla.

7.2.3 Preparación Raspberry Pi Zero W

Como variante para el uso de Raspberry Pi, también se ha probado utilizando una placa con menor potencia comparada con la Raspberry Pi 4, siendo esta una modelo Zero W.

Sin embargo, debido a esto es necesario configurarla sin entorno gráfico.

Durante este punto se realizarán las siguientes configuraciones:

- Configuración Headless
- Configuración Wi-fi
- Configuración SSH

En primer lugar, se ha de realizar la instalación del sistema operativo de la misma manera que se ha realizado con la Raspberry Pi 4.

En este momento empiezan los cambios. En primer lugar, hay que configurar un archivo para realizar la conexión a internet directamente.

Se ha de crear un archivo llamado “wpa_supplicant.conf” dentro del boot, teniendo este como contenido el texto que se muestra en la Figura 7. Sin embargo, hay que cambiar el código de country, el SSID y PSK a los adecuados. Siendo el primero de estos “ES” estando en España, y los otro respectivamente son el nombre de la red y la contraseña de esta.

```
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
  ssid="NETWORK-NAME"
  psk="NETWORK-PASSWORD"
}
```

Figura 7 Configuración del fichero wpa_supplicant

Si se ha configurado de manera adecuada la placa se conectará a dicha red unos minutos después de enchufar el cable de corriente.

El siguiente paso es habilitar el protocolo SSH. Para esto se ha de crear un archivo llamado "ssh" sin embargo no ha de tener extensión txt.

En este momento ya se puede establecer la conexión con esta placa desde el ordenador.

Esto se puede hacer desde la terminal escribiendo ssh@IP donde se ha de sustituir IP con la dirección IP de la placa. A continuación, será necesaria la contraseña, siendo por defecto esta "raspberry".

Si no han existido problemas durante este proceso, en este momento tenemos acceso a la placa desde el ordenador remoto. También es recomendable utilizar el programa WinSCP para realizar intercambios de ficheros de una manera sencilla.

7.2.4 Preparación equipo MATLAB

Para el procesamiento de los datos, se ha decidido utilizar el entorno MATLAB con la versión 2020a. Todos los programas y configuración se han realizado para esta versión, sin embargo, es posible utilizar otras versiones de este software, aunque existe la posibilidad de necesitar cambios.

7.3 Configuración Software

7.3.1 Configuración del entorno Rasbian

Durante el transcurso de este apartado se explicarán los pasos que hay que seguir para configurar el entorno Rasbian que se ha instalado previamente. Es necesario realizar dicha configuración independientemente de la placa que se este utilizando. Los pasos realizados serán los siguientes:

- Configuración IP's de ROS
- Configuración de localización de las librerías
- Prueba mediante roscore
- Librerías y posible problema

El siguiente paso, es la instalación de una distribución de ROS. Se ha seleccionado ROS Melodic Morenia[12] [13] debido al soporte que ofrece para el sistema operativo Rasbian junto al tiempo de vida de dicha distribución de ROS, acabando esta en el año 2025.

Utilizando las librerías disponibles dentro de ROS podemos crear una conexión mediante el protocolo IP para comunicarnos con el entorno de MATLAB que se configurara en el siguiente apartado. Sin embargo, para poder realizar esto es necesario encontrarse en la misma red de wifi y configurar el fichero bashrc correspondiente.

En primer lugar, es recomendable conectarse a una red hotspot creada desde el equipo que recibirá los datos enviados. Esto es debido a diferentes posibles problemas de conexión y perdida de paquetes encontrados en ciertas redes wifi. Utilizando esta configuración, se establecerá una conexión directa desde la Raspberry al equipo, evitando el trayecto hacia el router. En la Figura 9 se aprecia el esquema de la configuración de red sugerida, mientras que en la Figura 8 se presenta el esquema si se utilizase una conexión mediante un router. La creación de la red mencionada se explicará en el siguiente apartado.

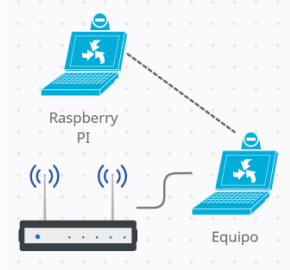


Figura 9 Conexión mediante hotspot

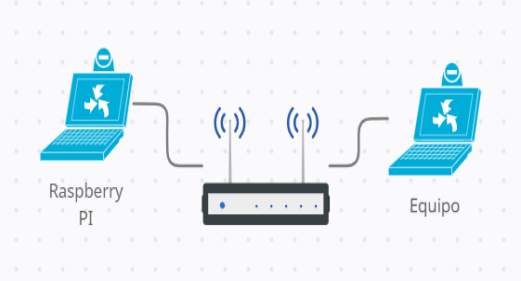


Figura 8 Conexión mediante router

El último paso para configurar la conexión de ROS dentro de la Raspberry Pi es la configuración del fichero mencionado anteriormente, es decir el bashrc. En primer lugar, será necesario conocer la dirección IP de la Raspberry. Para obtener esta dirección, se utilizará el comando "ifconfig" en una terminal. En la Figura 10, se puede apreciar la salida de dicho comando.

El dato necesario se encuentra en wlan0, siendo este la dirección inet, es decir, en dicho caso 192.168.1.125.

Este dato es dinámico, siempre que se cambie la red de internet en la que se encuentre, este cambiará y será necesario repetir estos pasos.

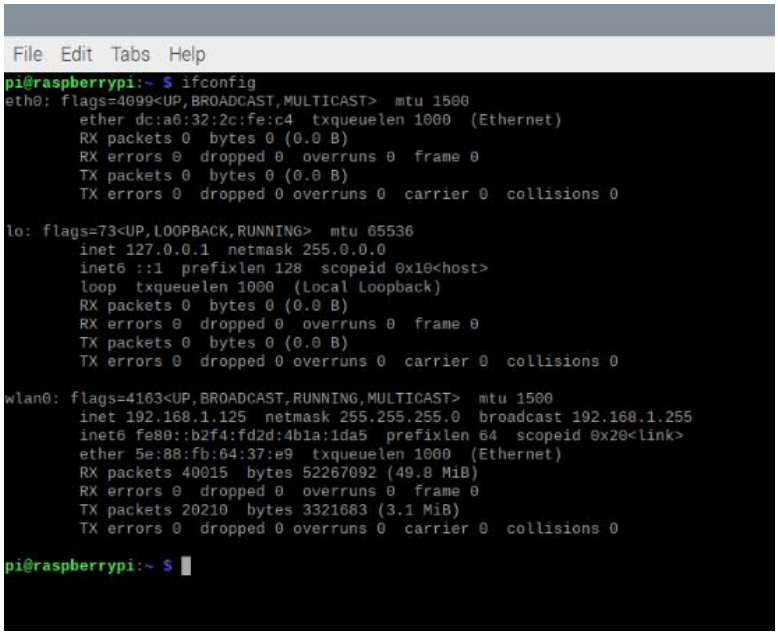


Figura 10 Ejecución del comando ifconfig

A continuación, se ha de editar el fichero bashrc. Para realizar esto se ha utilizado el editor GNU nano[14]. Esto se realizará utilizando el comando "nano .bashrc" encontrándonos en el directorio del usuario, siendo este /home/pi en este caso.

Se ha de añadir al final del fichero las siguientes líneas:

-export LD_LIBRARY_PATH=/opt/ros/medolic/lib

-export ROS_IP=192.168.1.125

-export ROS_MASTER_URI=http://192.168.1.125:11311

La primera de estas, indica el directorio donde se encuentran las librerías necesarias para la compilación de los ficheros que se explicarán en un futuro apartado.

En la segunda y tercera será necesario utilizar la dirección IP obtenida anteriormente. Será necesario sustituir dichas IP's siempre que cambien. El formato de estas siempre será el mismo, es decir, simplemente se ha de sustituir la dirección de 32bits con la de la máquina. El puerto 11311 es el utilizado por defecto, este se puede cambiar por uno disponible, pero se ha de tener en cuenta cuando se realice la conexión de ROS. Se han de cerrar todas las terminales que se están ejecutando en este momento para que se actualicen con los cambios realizados al fichero.

En este punto se puede comprobar si la instalación y configuración se ha realizado con éxito ejecutando el comando "roscore". Si no ha habido ningún error, la salida de este comando ha de ser parecida a la Figura 11.

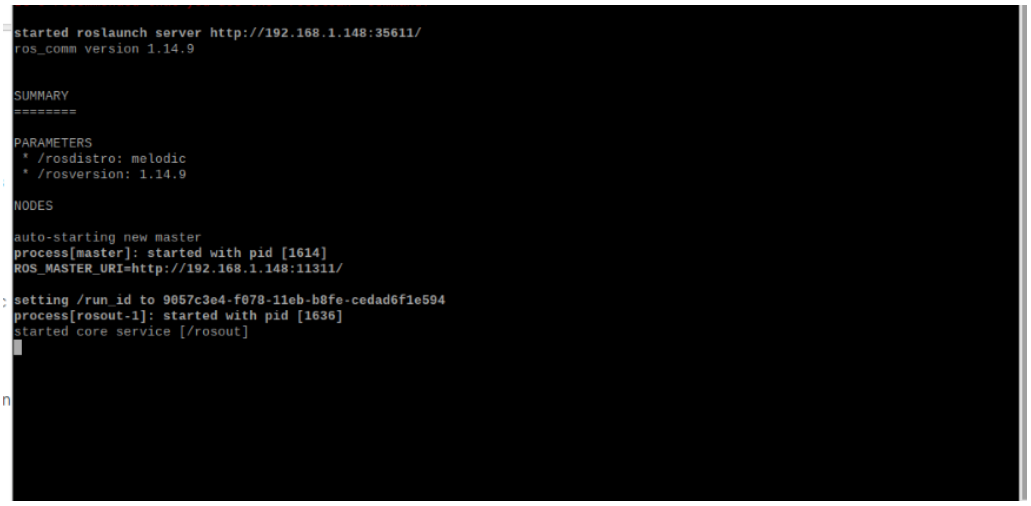


Figura 11 Ejecución del comando roscore

El último comando utilizado, sirve para realizar comunicaciones entre nodos de ROS. Siempre que se quiera utilizar ROS, ha de existir una sesion de roscore encendida.

El siguiente paso será obtener las librerías del Unicorn Hybrid Black correspondiente a la Raspberry Pi[15]. Los ficheros libunicorn.so y unicorn.h se han movido al directorio mencionado en el fichero bashrc. Estos seran utilizados para la compilacion de los programas creados, debido a esto, para facilitar la compilacion se han movido al mismo directorio donde se encuentran las librearias de ROS. Se recomienda utilizar el comando “sudo cp” para realizar una copia en dicho directorio.

En caso de que se trate de una instalación limpia del sistema operativo, existe la posibilidad de que durante la compilación no se encuentre la librería libbluetooth.so.3. Si se diese dicho caso, será necesario ejecutar el comando “sudo apt install libbluetooth-dev”.

7.3.2 Configuración del entorno MATLAB y red Hotspot

Empezando por la configuración de MATLAB, para utilizar este entorno será necesaria la instalación de la toolbox de ROS[8] .

Por el otro lado, se creará la red hotspot[16] dentro de este equipo para conectarse con la Raspberry tal y como se ha mencionado anteriormente.

La configuración de esta red creada se puede observar en la Figura 12.

Figura 12 Red hotspot

7.4 Librería Unicorn Hybrid Black

Dentro de las librerías descargadas anteriormente, nos podemos encontrar diferentes funciones, tipos de datos y estructuras. Utilizando estas podremos conectarnos al dispositivo y manejar el mismo, es decir, elegir la cantidad de datos que se analizan, los canales utilizados y la frecuencia de adquisición. Se ha realizado el diagrama de flujo mostrado en la Figura 13 para ejemplificar el curso del programa.

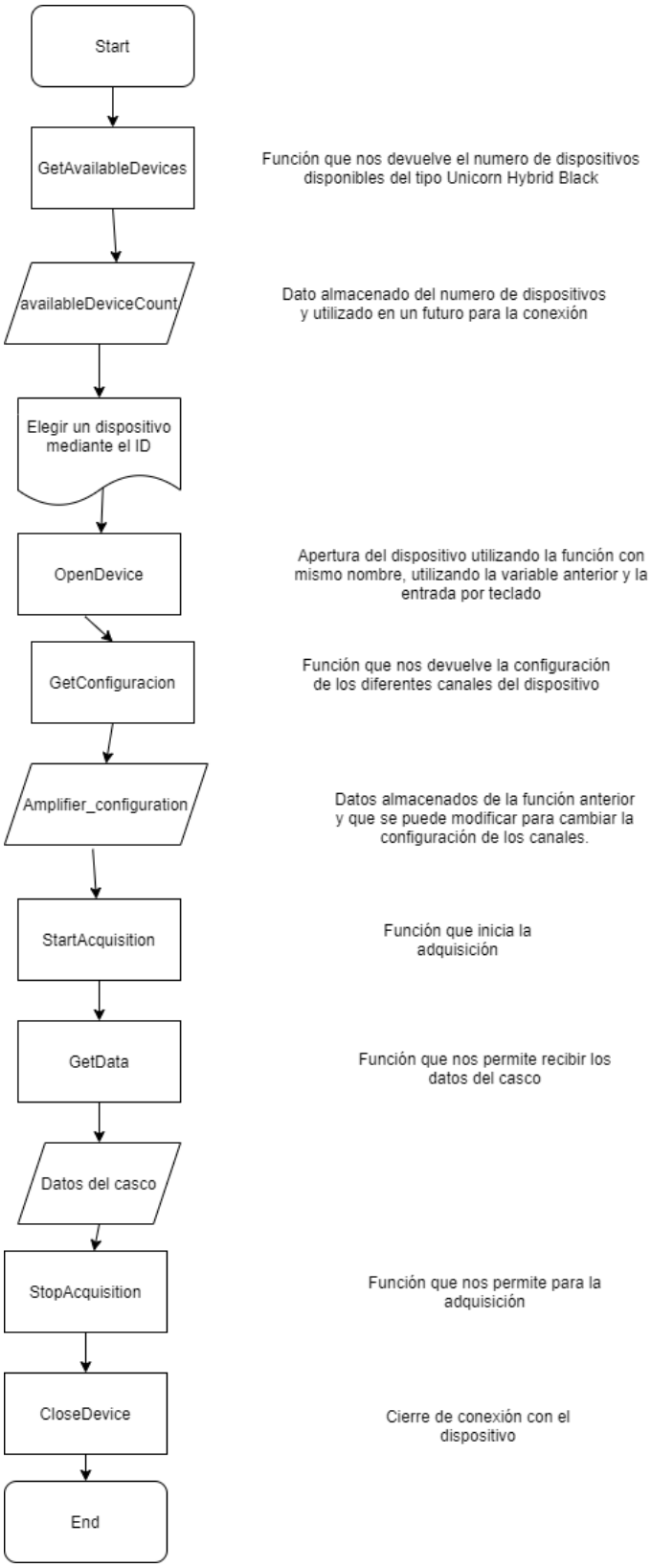


Figura 13 Diagrama de flujo

Entrando en detalle, los contenidos de las librerías utilizadas en este proyecto son las siguientes:

7.4.1 Tipos

UNICORN_DEVICE_SERIAL

Tipo de datos que almacena el número de serie del dispositivo. Este tipo de datos se utiliza junto a la funcion "`UNICORN_GetAvailableDevices`" como primer parámetro para obtener los dispositivos disponibles.

UNICORN_HANDLE

Tipo de datos que actúa como manejador del dispositivo. Este tipo se utilizará para la creación de una variable que se utilizará para abrir la conexión con el dispositivo, obtener la configuración del dispositivo y obtener los datos provenientes del mismo.

7.4.2 Estructuras

UNICORN_AMPLIFIER_CHANNEL

Estructura de datos que contiene información sobre un solo canal del amplificador.

Los componentes de esta estructura son :

name: tipo character de longitud 32 que contiene el nombre del canal

unit: tipo character de longitud 32 que contiene la unidad del canal, siendo en este caso por defecto en los canales correspondientes a los EEG's microvoltios.

range: Se trata de un array con tamaño dos con tipo de datos float que contiene el rango del canal. El primero contiene el valor mínimo y el segundo el valor máximo. Por defecto en los canales correspondientes a los EEG's estos valores son [-75000,75000].

enabled: Se trata de un booleano que indica si el canal esta activo o no.

UNICORN_AMPLIFIER_CONFIGURATION

Este segundo tipo de estructuras se corresponde con un array de la estructura mencionada anteriormente, es decir, guarda la configuración de todos los canales utilizados en la transmisión.

7.4.3 Funciones

UNICORN_GetAvailableDevices

Esta función escanea en búsqueda de dispositivos Unicorn Hybrid Black utilizando el protocolo Bluetooth estando emparejados o no con el dispositivo.

El primer parámetro de entrada de esta función es un puntero de un array del tipo UNICORN_DEVICE_SERIAL que guarda los dispositivos disponibles encontrados una vez ejecutada la búsqueda. Si se utiliza el valor NULL, solamente se devolverá el segundo parámetro.

El siguiente parámetro de entrada, es un puntero a una variable que guarda el número de dispositivos disponibles.

El último parámetro de entrada, se utiliza para hacer una búsqueda rápida o repetir la búsqueda anterior. Si el valor es "TRUE" se realizará lo primero y por el otro lado, un valor "FALSE", realizará la segunda acción.

Por último, la función devuelve un código de error en caso de no encontrar un dispositivo compatible.

UNICORN_OpenDevice

Esta segunda función, tiene como tarea establecer la conexión con los dispositivos encontrados anteriormente.

El primer parámetro de entrada es el número de serie del dispositivo que se desea conectar. Una opción para obtener este es usar la función anteriormente mencionada, utilizando la variable del primer parámetro.

El segundo parámetro de entrada es un puntero al manejador asociado con la sesión, en caso de realizarse esta con éxito. Esta variable es del tipo "UNICORN_HANDLER".

Por último, se devuelve un código de error en caso de fallar la operación de apertura del Unicorn Hybrid Black.

UNICORN_GetNumberOfAcquieredChannels

Esta función, determina el índice de los canales para la adquisición de los datos.

El primer parámetro de entrada es el manejador asociado con la sesión utilizado anteriormente.

El siguiente parámetro de entrada, es un puntero a la variable que recibe el índice de los canales.

Por defecto, se reciben los siguientes datos:

EEG: 1|2|3|4|5|6|7|8

Accelometer: X|Y|Z

Gyroscope: X|Y|Z

Counter

Battery Level

Validation Indicator

Por último, se devuelve un código de error en caso de fallar la operación.

UNICORN_GetConfiguration

Esta función, se encarga de recibir la configuración del amplificador.

El primer parámetro de entrada es el manejador asociado con la sesión utilizado anteriormente.

El siguiente parámetro, es un puntero a la variable que recibe la configuración del amplificador. Esta última variable, es una estructura del tipo “UNICORN_AMPLIFIER_CONFIGURATION”.

Por último, se devuelve un código de error en caso de fallar la operación.

UNICORN_StartAcquisition

La tarea de esta función es la inicialización de la adquisición de los datos, tanto en modo de medida como en modo de pruebas.

El primer parámetro de entrada es el manejador asociado con la sesión utilizado anteriormente.

El segundo parámetro, cambia el modo de medida a modo de pruebas y viceversa. Si se envía una señal de “TRUE”, la señal proviniendo de los canales 1-8, será una señal de prueba cuadrática. En contraposición, si se utiliza un valor de “FALSE” se cambiará a modo de medición.

Por último, se devuelve un código de error en caso de fallar el inicio de la adquisición de los datos.

UNICORN_GetData

Esta funcion, escribe un numero especifico de datos provenientes del Unicorn Hybrid Black en un buffer de tamaño conocido. Este buffer, es verificado para comprobar que tenga el tamaño suficiente para almacenar el numero solicitado de datos.

El primer parámetro de entrada es el manejador asociado con la sesión utilizado anteriormente.

El siguiente parámetro de entrada, es el número de escaneos que se desea hacer, siendo obligatoriamente mayor que cero. Cada escaneo consiste en un numero en coma flotante de 32-bits por cada canal adquirido.

El tercer parámetro de entrada es el buffer de destino en los que se almacenaran los datos adquiridos. Este buffer necesita tener suficiente memoria para almacenar el número de canales que se desean adquirir multiplicado por el número de muestras deseadas en cada trama.

El último parámetro de entrada, se necesita introducir el número de float's que se introducirán en el buffer.

Por último, se devuelve un código de error en caso de fallar la operación.

UNICORN_StopAcquisition

La tarea de esta funcion es parar la actual sesión de adquisición de datos

El parámetro de entrada es el manejador asociado con la sesión utilizado anteriormente.

Por último, se devuelve un código de error en caso de fallar la operación.

UNICORN_CloseDevice

Esta funcion cierra la conexión con el dispositivo abierto anteriormente.

El parámetro de entrada es el manejador asociado con la sesión utilizado anteriormente.

Por último, se devuelve un código de error en caso de fallar el cierre.

7.5 Ficheros Raspberry

Se han desarrollado diferentes archivos para la obtención de los datos a partir del ejemplo[15] de adquisición procedente del fabricante. Estos archivos se han creado para cumplir las funciones de grabación de una trama completa o para una conexión real.

Durante la toma de estos registros, el programa abrirá un socket bluetooth para realizar la conexión. Un posible problema es que esto falle.

Entre las posibles soluciones se haya en habilitar un agente bluetooth. Esto se puede realizar mediante el uso del comando “bluetoothctl” en una terminal continuando con el comando “default-agent”. De esta manera, se creará un manejador por defecto para realizar la conexión.

7.5.1 Trama Completa

En primer lugar, nos podemos encontrar con el archivo encargado de guardar en el fichero una trama completa, es decir, desde el momento en el que se inicia la grabación del EEG hasta que acaba el mismo. Este archivo es una modificación del fichero proporcionado por los desarrolladores del mismo.

Los cambios realizados consisten en la reproducción sonara y acústica de un estímulo para la captación de la onda P300, la creación de un nuevo canal artificial que representara con un “1” cuando se ha realizado el estímulo y con un “0” cuando no se aplique este. De esta manera existirán un total de 18 canales utilizados durante el proyecto. Otro cambio realizado es, la manera en la que se escriben los datos en el fichero. En la versión original los archivos se escriben en binario y en la versión modificada se realiza mediante en decimal.

Para la compilación de este script se ha utilizado el siguiente comando:

“g++ TramaCompleta.cpp -I /home/pi/Desktop/Unicorn/ -L /home/pi/Desktop/Unicorn/ -lunicorn -o AdquisicionCompleta”

Por último, cabe mencionar que será necesario un fichero de audio llamado “” dentro del directorio de los archivos que se ejecutará para crear el estimulo auditivo.

7.5.2 Tiempo Real

Este fichero es una modificación del anterior. En este caso, en lugar de escribir en un fichero se realizará la escritura en un Publisher de ROS. Por tanto, para este archivo serán necesarias las bibliotecas de ROS. En el Publisher los datos se escriben en una cadena de caracteres separados por el carácter ";". Estos datos enviados pueden ser de los canales que sean necesarios, es decir, pueden ser los 8 canales de EEG's o los 18 canales totales.

Estos se enviarán una vez, al entorno MATLAB donde será recibidos por el script correspondientes utilizando un Publisher típico de ROS. Este se muestra en la

Para la compilación de este script se ha utilizado el siguiente comando:

"g++ TiempoReal.cpp -o TiempoR -I /opt/ros/melodic/include -L /opt/ros/melodic/lib -lroscpp -lrostime -lrosconsole -lunicorn -lroscpp_serialization"

Como se puede apreciar se han utilizado las librerías provenientes de ROS llamadas "lroscpp", "lrostime", "lrosconsole", "lroscpp_serialization" durante la realización de este apartado. En caso de añadir diferentes funciones se ha de revisar este comando de compilación.

Por último, cabe mencionar que será necesario un fichero de audio llamado "" dentro del directorio de los archivos que se ejecutará para crear el estimulo auditivo.

7.6 Ficheros Matlab

Continuando con los archivos existentes en MATLAB, se han desarrollado dos tipos, para cumplir dos funciones diferentes.

7.6.1 Lectura

El primero de ellos, tiene como tarea recibir el fichero generado desde la Raspberry Pi en el que se incluyen los datos de los 18 canales provenientes del Unicorn Hybrid Black.

Entrando dentro del funcionamiento de este archivo, se ha divido el código en diferentes categorías para facilitar el entendimiento de este.

En primer lugar, nos encontramos con el código que se muestra en la Figura 14 correspondiéndose esta misma con la apertura y lectura del fichero. Los datos escritos se pueden leer de una manera sencilla debido al cambio en la escritura del archivo. Por tanto, lo único necesario será su apertura, lectura y separación en diferentes celdas. Debido a la separación de los datos habrá que dividirlos mediante el carácter ";".

```
fileID = fopen('data.bin');
formatSpec = '%c';
A = fscanf(fileID,formatSpec);
A = split(A,";");
```

Figura 14 Apertura del fichero

La siguiente parte, es la declaración de las variables necesarias. En la Figura 15 aparece todas las utilizadas durante el transcurso de este mismo

```
%%Declaracion de variables necesarias
Fs=250; %frecuencia de adquisicion utilizada
BpI=0.5; %frecuencia inferior del filtro bandpass
BpS=30; %frecuencia superior del filtro bandpass
tam=round(length(A)/18);  %preparamos el tamaño del array para optimizar tiempos
B1=zeros(tam,1,'single');
B2=zeros(tam,1,'single');
B3=zeros(tam,1,'single');
B4=zeros(tam,1,'single');
B5=zeros(tam,1,'single');
B6=zeros(tam,1,'single');
B7=zeros(tam,1,'single');
B8=zeros(tam,1,'single');
B9=zeros(tam,1,'single');
B10=zeros(tam,1,'single');
B11=zeros(tam,1,'single');
B12=zeros(tam,1,'single');
B13=zeros(tam,1,'single');
B14=zeros(tam,1,'single');
B15=zeros(tam,1,'single');
B16=zeros(tam,1,'single');
B17=zeros(tam,1,'single');
B18=zeros(tam,1,'single');
d = designfilt('bandpassiir','FilterOrder',2, ...
    'HalfPowerFrequency1',BpI,'HalfPowerFrequency2',BpS, ...
    'SampleRate',Fs); %% creacion del filtro bandpass

[b,a] = butter(2, [49 51]./(Fs/2), 'stop'); %%creacion del filtro notch en 50 hz
```

Figura 15 Creación de variables

En primer lugar, se crea una variable llamada “Fs” que contiene la frecuencia de adquisición del casco, el cual, tiene como valor máximo 250Hz. Esta será utilizada para los filtros que se aplicaran en un futuro.

En segundo lugar, se crean las variables “BpI” y “BpS” que representan los valores de corte para el filtrado Butterworth bandpass de segundo orden que se aplicara a las señales EEG’s. Estos valores son de 0.5-30Hz.

Las siguientes variables creadas servirán para almacenar los datos de los diferentes canales. En primer lugar, se calcula el tamaño de cada array dividiendo la cantidad de datos provenientes del fichero entre el número de canales. En este caso, se utilizan los 17 canales mas el canal extra del array que representa el estimulo creado. Si se decide por cambiar la cantidad de estos, será necesario cambiar este valor.

Las 17 variables siguientes se corresponden con los array’s utilizados para almacenar los datos mas el canal extra del array. Se crean los array’s con un tamaño fijo para optimizar los tiempos de ejecución.

Por último, se crean los dos filtros utilizados durante el proyecto. El primero, es el filtro bandpass y el segundo es el filtro notch. Ambos son filtro Butterworth de segundo orden. El filtrado notch depende de la ubicación que se encuentre el usuario del Unicorn Hybrid Black. Por ejemplo, si se utiliza en un país como Estados Unidos, el filtrado notch deberá ser de 60Hz para eliminar el ruido de la corriente eléctrica.

Al ubicarse el ejemplo en Europa, es necesario utilizar un filtrado de 50Hz.

El siguiente paso, es la división de los datos en los array's ya mencionados. Esto se logrará mediante el código mostrado en la Figura 16. Como se puede apreciar, se utiliza una variable llamada "access" para acceder a la posición del array que se desea. En este caso, en cada iteración del bucle, la variable aumente en 18. Esto es debido a la cantidad de canales que se han utilizado.

Por ejemplo, si se desean utilizar solamente 8 canales, será necesario quitar todas las líneas existentes desde el acceso a la variable "B9" incluida esta y cambiar el último aumento de "access" en cada iteración del bucle a un valor total de 11.

```
for i = 1:( tam )
B1(i)= str2double(A(access));
access=access+1;
B2(i)=str2double(A(access));
access=access+1;
B3(i)= str2double(A(access));
access=access+1;
B4(i)= str2double(A(access));
access=access+1;
B5(i)= str2double(A(access));
access=access+1;
B6(i)= str2double(A(access));
access=access+1;
B7(i)= str2double(A(access));
access=access+1;
B8(i)= str2double(A(access));
access=access+1;
B9(i)= str2double(A(access));
access=access+1;
B10(i)= str2double(A(access));
access=access+1;
B11(i)= str2double(A(access));
access=access+1;
B12(i)= str2double(A(access));
access=access+1;
B13(i)= str2double(A(access));
access=access+1;
B14(i)= str2double(A(access));
access=access+1;
B15(i)= str2double(A(access));
access=access+1;
B16(i)= str2double(A(access));
access=access+1;
B17(i)= str2double(A(access));
access=access+1;
B18(i)= str2double(A(access));
access=access+1;
```

Figura 16 División de los array's

A continuación, se muestran los datos obtenidos utilizando el código mostrado en la Figura 17 y Figura 18.

```
%se muestran los 8
figure
sgtitle('Raw data')
subplot(4,2,1);
plot(B1);
title('Canal 1')
subplot(4,2,2);
plot(B2);
title('Canal 2')
subplot(4,2,3);
plot(B3);
title('Canal 3')
subplot(4,2,4)
plot(B4)
title('Canal 4')
subplot(4,2,5);
plot(B5);
title('Canal 5')
subplot(4,2,6);
plot(B6);
title('Canal 6')
subplot(4,2,7);
plot(B7);
title('Canal 7')
subplot(4,2,8)
plot(B8)
title('Canal 8')
```

Figura 17 Plot de los primero 8 canales

```
%se muestran los canales del 9-18
figure
sgtitle('Raw data 9-18')
subplot(5,2,1);
plot(B9);
title('Canal 9')
subplot(5,2,2);
plot(B10);
title('Canal 10')
subplot(5,2,3);
plot(B11);
title('Canal 11')
subplot(5,2,4);
plot(B12);
title('Canal 12')
subplot(5,2,5);
plot(B13);
title('Canal 13')
subplot(5,2,6)
plot(B14);
title('Canal 14')
subplot(5,2,7);
plot(B15);
title('Canal 15')
subplot(5,2,8);
plot(B16);
title('Canal 16')
subplot(5,2,9)
plot(B17)
title('Canal 17')
subplot(5,2,10)
plot(B18)
title('Canal 18')
```

Figura 18 Plot de los canales del 9 al 18

Continuando con el archivo, el siguiente paso será aplicar el filtro bandpass creado con anterioridad a los diferentes canales. En la Figura 19 aparece el código para la aplicación del filtro y en la Figura 20 se muestra como imprimir por pantalla el código

```
C1= filter(d,B1);
C2= filter(d,B2);
C3= filter(d,B3);
C4= filter(d,B4);
C5= filter(d,B5);
C6= filter(d,B6);
C7= filter(d,B7);
C8= filter(d,B8);
```

Figura 19 Filtro Bandpass

```
figure
sgtitle('Filtrado Bandpass')
subplot(4,2,1);
plot(C1);
title('Canal 1');
subplot(4,2,2);
plot(C2);
title('Canal 2');
subplot(4,2,3);
plot(C3);
title('Canal 3');
subplot(4,2,4)
plot(C4);
title('Canal 4');
subplot(4,2,5);
plot(C5);
title('Canal 5');
subplot(4,2,6);
plot(C6);
title('Canal 6');
subplot(4,2,7);
plot(C7);
title('Canal 7');
subplot(4,2,8)
plot(C8);
title('Canal 8');
```

Figura 20 Plot filtro Bandpass

El siguiente paso será aplicar el filtro notch. Se puede apreciar en la Figura 21 y la Figura 22 como se aplica el filtrado y se muestra su resultado.

```
D1=filter(b,a,C1);
D2=filter(b,a,C2);
D3=filter(b,a,C3);
D4=filter(b,a,C4);
D5=filter(b,a,C5);
D6=filter(b,a,C6);
D7=filter(b,a,C7);
D8=filter(b,a,C8);
```

Figura 21 Filtro Notch

```
figure
sgtitle('Filtrado Notch')
subplot(4,2,1);
plot(D1);
title('Canal 1')
subplot(4,2,2);
plot(D2);
title('Canal 2')
subplot(4,2,3);
plot(D3);
title('Canal 3')
subplot(4,2,4)
plot(D4);
title('Canal 4')
subplot(4,2,5);
plot(D5);
title('Canal 5')
subplot(4,2,6);
plot(D6);
title('Canal 6')
subplot(4,2,7);
plot(D7);
title('Canal 7')
subplot(4,2,8)
plot(D8);
title('Canal 8')
```

Figura 22 Plot filtro Notch

Por último, para obtener los registros EEG's es necesario realizar un último filtrado para eliminar todos los datos que no estén en el rango de valores de [-100,100] microvoltios. Para esto se han eliminado todos los valores que no están en dicho intervalo después de aplicar los filtrados anteriores. Tal y como aparece en la Figura 23.

```
for i = 1:(length(D1)-1)

if ~(all(D1(i) >= BottomLimit & D1(i) <= UpperLimit))
   D1(i)=0;
end
if ~(all(D2(i) >= BottomLimit & D2(i) <= UpperLimit))
   D2(i)=0;
end
if ~(all(D3(i) >= BottomLimit & D3(i) <= UpperLimit))
   D3(i)=0;
end
if ~(all(D4(i) >= BottomLimit & D4(i) <= UpperLimit))
   D4(i)=0;
end
if ~(all(D5(i) >= BottomLimit & D5(i) <= UpperLimit))
   D5(i)=0;
end
if ~(all(D6(i) >= BottomLimit & D6(i) <= UpperLimit))
   D6(i)=0;
end
if ~(all(D7(i) >= BottomLimit & D7(i) <= UpperLimit))
   D7(i)=0;
end
if ~(all(D8(i) >= BottomLimit & D8(i) <= UpperLimit))
   D8(i)=0;
end

end
```

Figura 23 Reducción de los valores

El siguiente paso, sera eliminar todos los valores que sean 0 de los arrays. Esto se hara de la manera mostrada en la Figura 24.

```
D1 = nonzeros(D1);
D2 = nonzeros(D2);
D3 = nonzeros(D3);
D4 = nonzeros(D4);
D5 = nonzeros(D5);
D6 = nonzeros(D6);
D7 = nonzeros(D7);
D8 = nonzeros(D8);
```

Figura 24 Eliminación de los valores 0

Por último, se llamará a la funcion "DivisionWave()" para realizar el análisis del registro EEG.

7.6.2 Conexión en Tiempo Real

El segundo tipo de archivo creado tiene como función establecer una conexión en tiempo real con la Raspberry Pi. Como se ha mencionado anteriormente, esto se realiza mediante la toolbox de ROS.

En primer lugar, es necesario realizar la conexión con la Raspberry Pi. Como se ha mencionado previamente, se ha creado una conexión mediante una red hotspot con la placa. El siguiente paso es conocer la dirección IP tanto de la Raspberry Pi, como del equipo usuario de MATLAB.

Anteriormente se mostró como obtener la primera de estas. Al tratarse de un sistema operativo Windows, para la obtener la segunda, es necesario utilizar el comando "ipconfig" en una terminal. La dirección IPv4 del equipo se pueden observar tal y como aparece en la Figura 25.

```
Adaptador de LAN inalámbrica Wi-Fi:

   Sufijo DNS específico para la conexión. . : home
   Vínculo: dirección IPv6 local. . . : fe80::7861:3a85:eb3:f5fc%12
   Dirección IPv4. . . . . . . . . . . . . . : 192.168.1.103
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 192.168.1.1
```

Figura 25 IP Windows

Antes de continuar, se ha inicializar el comando “roscore” en una de las terminales para poder establecer la conexión.

Una vez obtenido este dato e inicializado el “roscore” en la Raspberry Pi, ya se puede inicializar la conexión. En la Figura 26 aparece en código necesario para realizar eso. En la primera y tercera línea, se ha de usar la dirección IP de la Raspberry Pi, mientras que en la segunda se ha de usar la del equipo MATLAB.

```
setenv('ROS_MASTER_URI','http://192.168.1.125:11311')
setenv('ROS_IP','192.168.1.103')
rosinit('192.168.1.125')
```

Figura 26 Ajustes IP's de MATLAB

El siguiente paso es la creación de las variables necesaria durante el proyecto. Estas son las mismas que han sido explicadas en el fichero anterior.

Desde este punto es necesario que desde la Raspberry Pi se haya ejecutado la creación del Publisher. El Publisher creado anteriormente tiene el identificador de chatter_pub, debido a esto, se creará un suscriber de la manera mostrada en la Figura 27 para poder realizar la lectura de las publicaciones.

```
sub=rossubscriber('/chatter_pub');
```

Figura 27 Suscriber ROS

El siguiente paso ocurrirá al recibir el mensaje enviado por el Publisher. Hasta ese momento será necesario esperar. Esto se realizará mediante el comando mostrado en la Figura 28.

```
msg = receive(sub);
```

Figura 28 Mensaje de ROS

Cuando se reciba un mensaje se procederá a dividir el contenido del mensaje en los diferentes canales. Al escribirse estos separados por el carácter de punto y coma, será primero necesario quitar estos de la manera mostrada en la Figura 29.

```
A = strsplit(msg.Data,';');
access=1;
[~,tamb]=size(A);
tamb=tamb/18;
```

Figura 29 Descomposición del mensaje de ROS

Desde este punto, todo será igual al fichero de MATLAB llamado Lectura.

7.6.3 DivisionWave

El último ficho de MATLAB que ha ser mencionado es la funcion “DivisionWaveComplete”.

El fichero es una modificación de un archivo de Mathworks[17].

Esta tiene como tarea dividir el registro en las diferentes ondas que lo forman. Entre ellas se encuentran las ondas Alpha, Beta, Theta y Gamma.

La segunda tarea de esta funcion es pasar al dominio frecuencial las ondas mencionadas anteriormente

La entrada de esta funcion es un registro EEG.

En primer lugar, se utilizan las funciones “wavedec()”, “detcoef()”, “appcoef()” y “wrcoef()” para la división en las diferentes ondas, tal y como se muestra en la Figura 30.

```
waveletFunction = 'db8';
                [C,L] = wavedec(s,8,waveletFunction);

                cD1 = detcoef(C,L,1);
                cD2 = detcoef(C,L,2);
                cD3 = detcoef(C,L,3);
                cD4 = detcoef(C,L,4);
                cD5 = detcoef(C,L,5); %GAMMA
                cD6 = detcoef(C,L,6); %BETA
                cD7 = detcoef(C,L,7); %ALPHA
                cD8 = detcoef(C,L,8); %THETA
                cA8 = appcoef(C,L,waveletFunction,8); %DELTA
                D1 = wrcoef('d',C,L,waveletFunction,1);
                D2 = wrcoef('d',C,L,waveletFunction,2);
                D3 = wrcoef('d',C,L,waveletFunction,3);
                D4 = wrcoef('d',C,L,waveletFunction,4);
                D5 = wrcoef('d',C,L,waveletFunction,5); %GAMMA
                D6 = wrcoef('d',C,L,waveletFunction,6); %BETA
                D7 = wrcoef('d',C,L,waveletFunction,7); %ALPHA
                D8 = wrcoef('d',C,L,waveletFunction,8); %THETA
                A8 = wrcoef('a',C,L,waveletFunction,8); %DELTA
```

Figura 30 División Wavelet

A continuación, se muestran estas ondas en el dominio del tiempo utilizando el código mostrado en la Figura 31.

```
Gamma = D5;
figure; subplot(5,1,1); plot(1:1:length(Gamma),Gamma);title('GAMMA');

Beta = D6;
subplot(5,1,2); plot(1:1:length(Beta), Beta); title('BETA');


Alpha = D7;
subplot(5,1,3); plot(1:1:length(Alpha),Alpha); title('ALPHA');

Theta = D8;
subplot(5,1,4); plot(1:1:length(Theta),Theta);title('THETA');
D8 = detrend(D8,0);

Delta = A8;

subplot(5,1,5);plot(1:1:length(Delta),Delta);title('DELTA');
```

Figura 31 Plot en el dominio del tiempo

El siguiente paso será la conversión al dominio frecuencial de las ondas. En este caso, será necesario utilizar el código mostrado en la Figura 32 y la Figura 33.

```
D5 = detrend(D5,0);
xdft = fft(D5);
freq = 0:N/length(D5):N/2;
xdft = xdft(1:length(D5)/2+1);
figure;subplot(511);plot(freq,abs(xdft));title('GAMMA-FREQUENCY');
xlim([0 100])
[~,~] = max(abs(xdft));

D6 = detrend(D6,0);
xdft2 = fft(D6);
freq2 = 0:N/length(D6):N/2;
xdft2 = xdft2(1:length(D6)/2+1);

subplot(512);plot(freq2,abs(xdft2));title('BETA');
xlim([0 100])
[~,~] = max(abs(xdft2));

D7 = detrend(D7,0);
xdft3 = fft(D7);
freq3 = 0:N/length(D7):N/2;
xdft3 = xdft3(1:length(D7)/2+1);

subplot(513);plot(freq3,abs(xdft3));title('ALPHA');
xlim([0 100])
```

Figura 32 Plot en el dominio frecuencial 1

```
[~,~] = max(abs(xdft3));



 xdft4 = fft(D8);
freq4 = 0:N/length(D8):N/2;
xdft4 = xdft4(1:length(D8)/2+1);

subplot(514);plot(freq4,abs(xdft4));title('THETA');
xlim([0 100])
[~,~] = max(abs(xdft4));


A8 = detrend(A8,0);
xdft5 = fft(A8);
freq5 = 0:N/length(A8):N/2;
xdft5 = xdft5(1:length(A8)/2+1);

subplot(515);plot(freq5,abs(xdft5));title('DELTA');
xlim([0 100])
[~,~] = max(abs(xdft5));
```

Figura 33 Plot en el dominio frecuencial 2

Por último, se muestra las diferentes ondas en la misma grafica. Utilizando el código mostrado en la Figura 34.

```
figure
sgtitle('g = Delta, r = theta , b = alpha , y = beta juntos en frecuencia')
hold on
plot(freq3,abs(xdft5),'g');
plot(freq4,abs(xdft4),'r');
plot(freq3,abs(xdft3),'b');
plot(freq2,abs(xdft2),'y');
hold off
xlim([0 100])
```

Figura 34 Muestra de las ondas en la misma figura

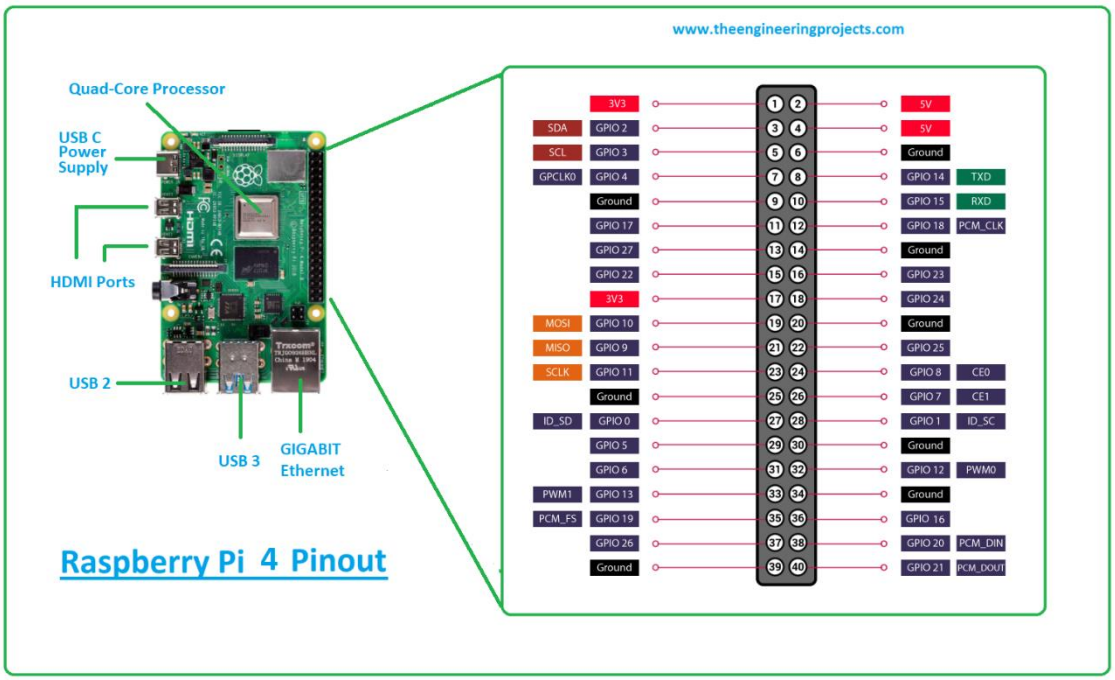
7.7 Conclusión e ideas futuras

Durante este proyecto se ha realizado una conexión entre los diferentes componentes que para poder analizar los registros EEG. En este caso se ha utilizado MATLAB debido a las facilidades que nos proporciona, sin embargo, si se desea realizar una versión sin tanto tiempo de espera será necesario el uso de un lenguaje mas rápido del estilo de C o C++.

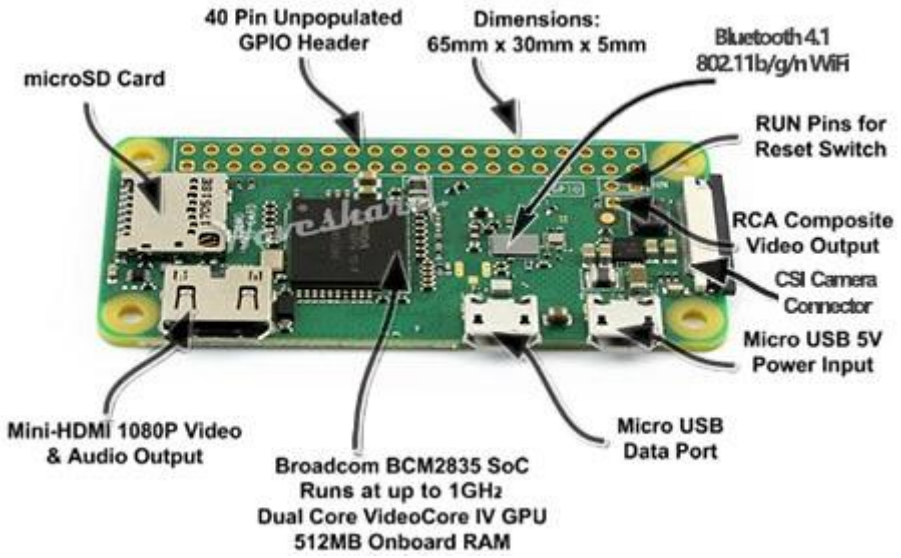
Otro aspecto que ha de tenerse en cuenta es el ruido existente en los registros. Es recomendable utilizar la programación de Unicorn Hybrid primero para observar cuando se estabilizan los electrodos además de usar gel conducto.

8. Planos y diagramas

Esquema Raspberry Pi 4



Esquema Raspberry Pi Zero W



9. Pliego de Condiciones

Requisitos hardware

- Tarjeta Micro SD 64 GB
- Raspberry Pi 4 / Zero W
- Periféricos de salida de audio
- Ordenador con requisitos mínimos
    - Intel o AMD x86-64 procesador
    - 15 GB de almacenamiento HDD disponible
    - 4GB RAM

Requisitos software

- Sistema Operativo Raspbian
- Sistema Operativo Ubuntu 16.04/18.04/20.04 or Debian 10 or Mac Big Sur/Catalina/Mojave or Windows 7/10

10. Presupuesto

Teniendo en cuenta que el coste de un ingeniero junior es de 25€/h, incluyendo gastos por seguridad social y otros impuestos por contratación. Con una jornada laboral de 8h, 5 días a la semana y 44 semanas laborales, son en total 1760 horas.

Concepto	**Coste**
Coste Hardawre	
Unicorn Hybrid Black	990€
Raspberry Pi 4 Model B 4GB RAM / Raspberry Pi Zero W	64€ / 25€
Tarjeta Micro SD 64GB	9€
Accesorios	50€
Coste Software	
Licencia Matlab	800€
Coste de Personal	
Coste de Ingeniero	44.000€
Coste Total	45.874 (+39) €

11. Guía de uso

El primer paso para la puesta en funciona del sistema es el ajuste de la IP dentro del fichero bashrc. Una vez que se hayan añadido las 3 líneas necesarias, las cuales han sido explicadas en la sección de la configuración del entorno Rasbian.

El siguiente paso, será encender el casco.

Una vez realizado esto, se ha abrir una nueva terminal en la que se ejecute el comando “roscore”.

El siguiente paso será ejecutar el archivo generado durante la compilación.

Una vez realizado esto, será necesario esperar a que se encuentre el casco y realizar la conexión por bluetooth.

En este momento, dependiendo de lo que deseamos hacer, es decir, grabar un archivo o realizar una conexión en tiempo real, será necesario realizar diferentes acciones.

11.1 Grabación de un fichero

Una vez realizada la conexión el programa empezara a grabar el fichero leyendo los canales del casco.

11.2 Conexión en tiempo real

En el segundo caso, una vez realizada la conexión, el programa esperara una entrada desde el teclado para continuar. En este momento se debe realizar la conexión desde MATLAB.

Al ejecutar, el script desde este último entorno mencionado, se creará la conexión de ROS conectando el suscriber con el Publisher encargado de enviar los mensajes. El programa pedirá pulsar una tecla y pulsar la tecla “enter”. A continuación, se quedará en espera hasta que reciba un mensaje con los datos provenientes de la Raspberry Pi.

En este momento se continuará operando la Raspberry Pi, en donde enviaremos por teclado cualquier carácter para iniciar la adquisición de los datos. Una vez realizado esto, los datos serán enviados a MATLAB.

Una vez recibidos, el programa continuara filtrando los datos de la manera mencionada anteriormente para mostrar todos los canales de adquisición.

12. Bibliografía

[1] Jonathan Wolpaw, Niels Birbaumer, Dennis J. McFarland, Gert Pfurtscheller, and Theresa M. Vaughan. Brain-computer interfaces for communication and control. Clinical Neurophysiology, 113(6):767{791, June 2002.

[2] Raspberry Pi. https://www.raspberrypi.org

[3] https://www.unicorn-bi.com/product/unicorn-hybrid-black/

[4] https://github.com/unicorn-bi/Unicorn-Suite-Hybrid-Black/tree/master/Unicorn%20Raspberry%20Pi%20Zero%20W%20C%20API

[5] ROS. https://www.ros.org

[6]https://www.researchgate.net/profile/Christoph-Guger/publication/340940556_Unicorn_Brain_Interface/links/5ea69e9b92851c1a90734837/Unicorn-Brain-Interface.pdf

[7] http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv

[8] https://www.mathworks.com/products/ros.html

[9]https://en.wikipedia.org/wiki/10–20_system_(EEG)

[10]https://www.raspberrypi.org/software/operating-systems/

[11]https://www.balena.io/etcher/

[12]http://wiki.ros.org/melodic

[13]http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Melodic%20on%20the%20Raspberry%20Pi

[14] https://www.nano-editor.org/download.php

[15]https://github.com/unicorn-bi/Unicorn-Suite-Hybrid-Black/tree/master/Unicorn%20Raspberry%20Pi%20Zero%20W%20C%20API

[16]https://support.microsoft.com/en-us/windows/use-your-pc-as-a-mobile-hotspot-c89b0fad-72d5-41e8-f7ea-406ad9036b85#:~:text=Select%20the%20Start%20button%2C%20then,Internet%20connection%20with%20other%20devices.

[17]https://www.mathworks.com/matlabcentral/fileexchange/55112-eeg-analysis-and-classification

13. Anexo. Código fuente

Lectura(MATLAB)

```
%%Lectura del archivo
%

fileID = fopen('data.bin');
formatSpec = '%c';
A = fscanf(fileID,formatSpec);
A = split(A,";");

%%Declaracion de variables necesarias
Fs=250; %frecuencia de adquisicion utilizada
BpI=0.5; %frecuencia inferior del filtro bandpass
BpS=30; %frecuencia superior del filtro bandpass
tam=round(length(A)/18);  %preparamos el tamaño del array para optimizar
tiempos
B1=zeros(tam,1,'single');
B2=zeros(tam,1,'single');
B3=zeros(tam,1,'single');
B4=zeros(tam,1,'single');
B5=zeros(tam,1,'single');
B6=zeros(tam,1,'single');
B7=zeros(tam,1,'single');
B8=zeros(tam,1,'single');
B9=zeros(tam,1,'single');
B10=zeros(tam,1,'single');
B11=zeros(tam,1,'single');
B12=zeros(tam,1,'single');
B13=zeros(tam,1,'single');
B14=zeros(tam,1,'single');
B15=zeros(tam,1,'single');
B16=zeros(tam,1,'single');
B17=zeros(tam,1,'single');
B18=zeros(tam,1,'single');
d = designfilt('bandpassiir','FilterOrder',2, ...
    'HalfPowerFrequency1',BpI,'HalfPowerFrequency2',BpS, ...
    'SampleRate',Fs); %% creacion del filtro bandpass
```

```
[b,a] = butter(2, [49 51]./(Fs/2), 'stop'); %%creacion del filtro notch en
50 hz


%se convierte el array inicial leido del archivo a 17 diferentes que se
corresponden con los canales
access=1;
[tamb,~]=size(A);

for i = 1:( tam )
B1(i)= str2double(A(access));
access=access+1;
B2(i)=str2double(A(access));
access=access+1;
B3(i)= str2double(A(access));
access=access+1;
B4(i)= str2double(A(access));
access=access+1;
B5(i)= str2double(A(access));
access=access+1;
B6(i)= str2double(A(access));
access=access+1;
B7(i)= str2double(A(access));
access=access+1;
B8(i)= str2double(A(access));
access=access+1;
B9(i)= str2double(A(access));
access=access+1;
B10(i)= str2double(A(access));
access=access+1;
B11(i)= str2double(A(access));
access=access+1;
B12(i)= str2double(A(access));
access=access+1;
B13(i)= str2double(A(access));
access=access+1;
B14(i)= str2double(A(access));
access=access+1;
B15(i)= str2double(A(access));
access=access+1;
B16(i)= str2double(A(access));
access=access+1;
B17(i)= str2double(A(access));
access=access+1;
B18(i)= str2double(A(access));
access=access+1;
end
%se muestran los 8 canales correspondientes a los EEG's
figure
sgtitle('Raw data')
subplot(4,2,1);
plot(B1);

title('Canal 1')
subplot(4,2,2);
plot(B2);
title('Canal 2')
subplot(4,2,3);
plot(B3);
title('Canal 3')
subplot(4,2,4)
```

```
plot(B4)
title('Canal 4')
subplot(4,2,5);
plot(B5);
title('Canal 5')
subplot(4,2,6);
plot(B6);
title('Canal 6')
subplot(4,2,7);
plot(B7);
title('Canal 7')
subplot(4,2,8)
plot(B8)
title('Canal 8')

%se muestran los canales del 9-18
figure
sgtitle('Raw data 9-18')
subplot(5,2,1);
plot(B9);
title('Canal 9')
subplot(5,2,2);
plot(B10);
title('Canal 10')
subplot(5,2,3);
plot(B11);
title('Canal 11')
subplot(5,2,4);
plot(B12);
title('Canal 12')
subplot(5,2,5);
plot(B13);
title('Canal 13')
subplot(5,2,6)
plot(B14);
title('Canal 14')
subplot(5,2,7);
plot(B15);
title('Canal 15')
subplot(5,2,8);
plot(B16);
title('Canal 16')
subplot(5,2,9)
plot(B17)
title('Canal 17')
subplot(5,2,10)
plot(B18)
title('Canal 18')


% se aplica el filtro bandpass a los diferentes canales

C1= filter(d,B1);
C2= filter(d,B2);
C3= filter(d,B3);
C4= filter(d,B4);
C5= filter(d,B5);
C6= filter(d,B6);
C7= filter(d,B7);
C8= filter(d,B8);

%se muestran los 8 canales correspondientes a los EEG's una vez aplicado
el
```

```
%filtro
figure
sgtitle('Filtrado Bandpass')
subplot(4,2,1);
plot(C1);
title('Canal 1');
subplot(4,2,2);
plot(C2);
title('Canal 2');
subplot(4,2,3);
plot(C3);
title('Canal 3');
subplot(4,2,4)
plot(C4);
title('Canal 4');
subplot(4,2,5);
plot(C5);
title('Canal 5');
subplot(4,2,6);
plot(C6);
title('Canal 6');
subplot(4,2,7);
plot(C7);
title('Canal 7');
subplot(4,2,8)
plot(C8);
title('Canal 8');



% se aplica el filtro notch a los diferentes canales

D1=filter(b,a,C1);
D2=filter(b,a,C2);
D3=filter(b,a,C3);
D4=filter(b,a,C4);
D5=filter(b,a,C5);
D6=filter(b,a,C6);
D7=filter(b,a,C7);
D8=filter(b,a,C8);


%se muestran los 8 canales correspondientes a los EEG's una vez aplicado
el
%filtro
figure
sgtitle('Filtrado Notch')
subplot(4,2,1);
plot(D1);
title('Canal 1')
subplot(4,2,2);
plot(D2);
title('Canal 2')
subplot(4,2,3);
plot(D3);
title('Canal 3')
subplot(4,2,4)
plot(D4);
title('Canal 4')
subplot(4,2,5);
plot(D5);
title('Canal 5')
subplot(4,2,6);
```

```
plot(D6);
title('Canal 6')
subplot(4,2,7);
plot(D7);
title('Canal 7')
subplot(4,2,8)
plot(D8);
title('Canal 8')



UpperLimit=100;
BottomLimit=-100;
% Primero=2500;
% Ultimo=3750;
%
% D1=D1(Primero:Ultimo);
% D2=D2(Primero:Ultimo);
% D3=D3(Primero:Ultimo);
% D4=D4(Primero:Ultimo);
% D5=D5(Primero:Ultimo);
% D6=D6(Primero:Ultimo);
% D7=D7(Primero:Ultimo);
% D8=D8(Primero:Ultimo);


for i = 1:(length(D1)-1)

if ~(all(D1(i) >= BottomLimit & D1(i) <= UpperLimit))
   D1(i)=0;
end
if ~(all(D2(i) >= BottomLimit & D2(i) <= UpperLimit))
   D2(i)=0;
end
if ~(all(D3(i) >= BottomLimit & D3(i) <= UpperLimit))
   D3(i)=0;
end
if ~(all(D4(i) >= BottomLimit & D4(i) <= UpperLimit))
   D4(i)=0;
end
if ~(all(D5(i) >= BottomLimit & D5(i) <= UpperLimit))
   D5(i)=0;
end
if ~(all(D6(i) >= BottomLimit & D6(i) <= UpperLimit))
   D6(i)=0;
end
if ~(all(D7(i) >= BottomLimit & D7(i) <= UpperLimit))
   D7(i)=0;
end
if ~(all(D8(i) >= BottomLimit & D8(i) <= UpperLimit))
   D8(i)=0;
end

end

D1 = nonzeros(D1);
D2 = nonzeros(D2);
D3 = nonzeros(D3);
D4 = nonzeros(D4);
D5 = nonzeros(D5);
D6 = nonzeros(D6);
D7 = nonzeros(D7);
D8 = nonzeros(D8);
```

```
figure
sgtitle('Filtrado Notch con enfoque')
subplot(4,2,1);
plot(D1);
title('Canal 1')
ylabel('microvoltios')
subplot(4,2,2);
plot(D2);
title('Canal 2')
ylabel('microvoltios')
subplot(4,2,3);
plot(D3);
title('Canal 3')
ylabel('microvoltios')
subplot(4,2,4)
plot(D4);
title('Canal 4')
ylabel('microvoltios')
subplot(4,2,5);
plot(D5);
title('Canal 5')
ylabel('microvoltios')
subplot(4,2,6);
plot(D6);
title('Canal 6')
ylabel('microvoltios')
subplot(4,2,7);
plot(D7);
title('Canal 7')
ylabel('microvoltios')
subplot(4,2,8)
plot(D8);
title('Canal 8')
ylabel('microvoltios')

%


 DivisionWaveComplete(D6);
```

TiempoReal(MATLAB)

```
setenv('ROS_MASTER_URI','http://192.168.1.125:11311')
setenv('ROS_IP','192.168.137.188')
rosinit('192.168.137.188')
```

```
Fs=250;
T = 1/Fs;
BpI=0.5; %frecuencia inferior de bandpass
BpS=30; %frecuencia superior de bandpass
datosRecibidos=[];
B1=[];
B2=[];
B3=[];
B4=[];
B5=[];
B6=[];
B7=[];
B8=[];
B9=[];
B10=[];
B11=[];
B12=[];
B13=[];
B14=[];
B15=[];
B16=[];
B17=[];
B18=[];
d = designfilt('bandpassiir','FilterOrder',2, ...
    'HalfPowerFrequency1',BpI,'HalfPowerFrequency2',BpS, ...
    'SampleRate',Fs); %% creacion del filtro bandpass

[b,a] = butter(2, [49 51]./(Fs/2), 'stop'); %%creacion del filtro notch en
50 hz


sub=rossubscriber('/chatter_pub');


disp(" Suscriber establecido" )



input('Pulse una tecla para empezar la transmision de los datos ','s');


j=0;
nRep=1;
while (j<nRep)

j=j+1;

msg = receive(sub);
disp("mensaje recibido")

A = strsplit(msg.Data,';');
access=1;
[~,tamb]=size(A);
tamb=tamb/18;


for i = 1:( tamb -1 )
B1(i)= str2double(A(access));
access=access+1;
B2(i)=str2double(A(access));
access=access+1;
B3(i)= str2double(A(access));
```

```
access=access+1;
B4(i)= str2double(A(access));
access=access+1;
B5(i)= str2double(A(access));
access=access+1;
B6(i)= str2double(A(access));
access=access+1;
B7(i)= str2double(A(access));
access=access+1;
B8(i)= str2double(A(access));
access=access+1;
B9(i)= str2double(A(access));
access=access+1;
B10(i)= str2double(A(access));
access=access+1;
B11(i)= str2double(A(access));
access=access+1;
B12(i)= str2double(A(access));
access=access+1;
B13(i)= str2double(A(access));
access=access+1;
B14(i)= str2double(A(access));
access=access+1;
B15(i)= str2double(A(access));
access=access+1;
B16(i)= str2double(A(access));
access=access+1;
B17(i)= str2double(A(access));
access=access+1;
B18(i)= str2double(A(access));
access=access+1;
end


% se aplica el filtro bandpass a los diferentes canales

C1= filter(d,B1);
C2= filter(d,B2);
C3= filter(d,B3);
C4= filter(d,B4);
C5= filter(d,B5);
C6= filter(d,B6);
C7= filter(d,B7);
C8= filter(d,B8);


% se aplica el filtro notch a los diferentes canales

D1=filter(b,a,C1);
D2=filter(b,a,C2);
D3=filter(b,a,C3);
D4=filter(b,a,C4);
D5=filter(b,a,C5);
D6=filter(b,a,C6);
D7=filter(b,a,C7);
D8=filter(b,a,C8);



UpperLimit=100;
BottomLimit=-100;
% Primero=2500;
```

```
% Ultimo=3750;
%
% D1=D1(Primero:Ultimo);
% D2=D2(Primero:Ultimo);
% D3=D3(Primero:Ultimo);
% D4=D4(Primero:Ultimo);
% D5=D5(Primero:Ultimo);
% D6=D6(Primero:Ultimo);
% D7=D7(Primero:Ultimo);
% D8=D8(Primero:Ultimo);
for i = 1:(length(D1)-1)

if ~(all(D1(i) >= BottomLimit & D1(i) <= UpperLimit))
   D1(i)=0;
end
if ~(all(D2(i) >= BottomLimit & D2(i) <= UpperLimit))
   D2(i)=0;
end
if ~(all(D3(i) >= BottomLimit & D3(i) <= UpperLimit))
   D3(i)=0;
end
if ~(all(D4(i) >= BottomLimit & D4(i) <= UpperLimit))
   D4(i)=0;
end
if ~(all(D5(i) >= BottomLimit & D5(i) <= UpperLimit))
   D5(i)=0;
end
if ~(all(D6(i) >= BottomLimit & D6(i) <= UpperLimit))
   D6(i)=0;
end
if ~(all(D7(i) >= BottomLimit & D7(i) <= UpperLimit))
   D7(i)=0;
end
if ~(all(D8(i) >= BottomLimit & D8(i) <= UpperLimit))
   D8(i)=0;
end

end

D1 = nonzeros(D1);
D2 = nonzeros(D2);
D3 = nonzeros(D3);
D4 = nonzeros(D4);
D5 = nonzeros(D5);
D6 = nonzeros(D6);
D7 = nonzeros(D7);
D8 = nonzeros(D8);


figure
sgtitle('Filtrado Notch con enfoque')
subplot(4,2,1);
plot(D1);
title('Canal 1')
ylabel('microvoltios')
subplot(4,2,2);
plot(D2);
title('Canal 2')
ylabel('microvoltios')
subplot(4,2,3);
plot(D3);
title('Canal 3')
ylabel('microvoltios')
```

```
subplot(4,2,4)
plot(D4);
title('Canal 4')
ylabel('microvoltios')
subplot(4,2,5);
plot(D5);
title('Canal 5')
ylabel('microvoltios')
subplot(4,2,6);
plot(D6);
title('Canal 6')
ylabel('microvoltios')
subplot(4,2,7);
plot(D7);
title('Canal 7')
ylabel('microvoltios')
subplot(4,2,8)
plot(D8);
title('Canal 8')
ylabel('microvoltios')

DivisionWaveComplete(D6);
end
    rosshutdown
```

DivisionWave(MATLAB)

```
function DivisionWaveComplete(EEG)
close all;

s=EEG;
figure;
p=plot(s);

title('EEG Signal')
warning('off','all')

fs = 250;

% Sampling frequency

N=length(s);


waveletFunction = 'db8';
                [C,L] = wavedec(s,8,waveletFunction);

                cD1 = detcoef(C,L,1);
                cD2 = detcoef(C,L,2);
                cD3 = detcoef(C,L,3);
                cD4 = detcoef(C,L,4);
                cD5 = detcoef(C,L,5); %GAMMA
                cD6 = detcoef(C,L,6); %BETA
                cD7 = detcoef(C,L,7); %ALPHA
                cD8 = detcoef(C,L,8); %THETA
                cA8 = appcoef(C,L,waveletFunction,8); %DELTA
```

```
D1 = wrcoef('d',C,L,waveletFunction,1);
D2 = wrcoef('d',C,L,waveletFunction,2);
D3 = wrcoef('d',C,L,waveletFunction,3);
D4 = wrcoef('d',C,L,waveletFunction,4);
D5 = wrcoef('d',C,L,waveletFunction,5); %GAMMA
D6 = wrcoef('d',C,L,waveletFunction,6); %BETA
D7 = wrcoef('d',C,L,waveletFunction,7); %ALPHA
D8 = wrcoef('d',C,L,waveletFunction,8); %THETA
A8 = wrcoef('a',C,L,waveletFunction,8); %DELTA


Gamma = D5;
figure; subplot(5,1,1); plot(1:1:length(Gamma),Gamma);title('GAMMA');

Beta = D6;
subplot(5,1,2); plot(1:1:length(Beta), Beta); title('BETA');


Alpha = D7;
subplot(5,1,3); plot(1:1:length(Alpha),Alpha); title('ALPHA');

Theta = D8;
subplot(5,1,4); plot(1:1:length(Theta),Theta);title('THETA');
D8 = detrend(D8,0);

Delta = A8;

subplot(5,1,5);plot(1:1:length(Delta),Delta);title('DELTA');



D5 = detrend(D5,0);
xdft = fft(D5);
freq = 0:N/length(D5):N/2;
xdft = xdft(1:length(D5)/2+1);
figure;subplot(511);plot(freq,abs(xdft));title('GAMMA-FREQUENCY');
xlim([0 100])
[~,~] = max(abs(xdft));

D6 = detrend(D6,0);
xdft2 = fft(D6);
freq2 = 0:N/length(D6):N/2;
xdft2 = xdft2(1:length(D6)/2+1);

subplot(512);plot(freq2,abs(xdft2));title('BETA');
xlim([0 100])
[~,~] = max(abs(xdft2));

D7 = detrend(D7,0);
xdft3 = fft(D7);
freq3 = 0:N/length(D7):N/2;
xdft3 = xdft3(1:length(D7)/2+1);

subplot(513);plot(freq3,abs(xdft3));title('ALPHA');
xlim([0 100])


[~,~] = max(abs(xdft3));


 xdft4 = fft(D8);
```

```
freq4 = 0:N/length(D8):N/2;
xdft4 = xdft4(1:length(D8)/2+1);

subplot(514);plot(freq4,abs(xdft4));title('THETA');
xlim([0 100])
[~,~] = max(abs(xdft4));


A8 = detrend(A8,0);
xdft5 = fft(A8);
freq5 = 0:N/length(A8):N/2;
xdft5 = xdft5(1:length(A8)/2+1);

subplot(515);plot(freq5,abs(xdft5));title('DELTA');
xlim([0 100])
[~,~] = max(abs(xdft5));


figure
sgtitle('g = Delta, r = theta , b = alpha , y = beta juntos en
frecuencia')
hold on
plot(freq3,abs(xdft5),'g');
plot(freq4,abs(xdft4),'r');
plot(freq3,abs(xdft3),'b');
plot(freq2,abs(xdft2),'y');
hold off
xlim([0 100])

end
```

Trama Completa (Raspberry)

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <chrono>
#include <thread>
#include <string.h>
// Include unicorn header-file.
#include "unicorn.h"
using namespace std;
// Specifications for the data acquisition.
//------------------------------------------------------------------------
------
```

```
#define DATA_FILE                       "data.bin"          // The name of the
file which is storing acquired data.
#define ACQUISITION_DURATION_S    10.0f               // 10 segundos por cada
repeticion de trama eeg
#define FRAME_LENGTH              1                         // The number of
samples acquired per get data call.
#define TESTSIGNAL_ENABLED        false               // Flag to enable or
disable testsignal.

// Function declarations.
//------------------------------------------------------------------------------
------
void HandleError(int errorCode);
void PrintErrorMessage(int errorCode);


// Metodo de lectura de los datos del casco usando el ejemplo proporcionado por
el fabricante
//------------------------------------------------------------------------------
------
// Main. Program entry point.
//------------------------------------------------------------------------------
------
int main()
{
	std::cout << "Unicorn Acquisition Example" << std::endl;
	std::cout << "---------------------------" << std::endl << std::endl;

	// Variable to store error codes.
	int errorCode = UNICORN_ERROR_SUCCESS;

	// Structure that holds the handle for the currecnt session.
	UNICORN_HANDLE deviceHandle = 0;

	try
	{
		// Get available devices.
		//------------------------------------------------------------------
--------------------

		// Get number of available devices.
		unsigned int availableDevicesCount = 0;
		errorCode = UNICORN_GetAvailableDevices(NULL,
&availableDevicesCount, TRUE);
		HandleError(errorCode);

		if (availableDevicesCount < 1)
		{
			std::cout << "No device available. Please pair with a Unicorn
device first.";
			errorCode = UNICORN_ERROR_GENERAL_ERROR;
		}

		//Get available device serials.
		UNICORN_DEVICE_SERIAL *availableDevices = new
UNICORN_DEVICE_SERIAL[availableDevicesCount];
		errorCode = UNICORN_GetAvailableDevices(availableDevices,
&availableDevicesCount, true);
		HandleError(errorCode);

		//Print available device serials.
```

```
		std::cout << "Available devices:" << std::endl;
		for (unsigned int i = 0; i<availableDevicesCount; i++)
		{
			std::cout << "#" << i << ": " << availableDevices[i] <<
std::endl;
		}

		// Request device selection.
		std::cout << "\nSelect device by ID #";
		unsigned int deviceSelection;
		float estimulo;
		std::cin >> deviceSelection;
		if (deviceSelection >= availableDevicesCount || deviceSelection <
0)
			errorCode = UNICORN_ERROR_GENERAL_ERROR;

		HandleError(errorCode);

		// Open selected device.
		//---------------------------------------------------------------
--------------------
		std::cout << "Trying to connect to '" <<
availableDevices[deviceSelection] << "'." << std::endl;
		errorCode =
UNICORN_OpenDevice(availableDevices[deviceSelection],&deviceHandle);
		HandleError(errorCode);

		std::cout << "Connected to '" << availableDevices[deviceSelection]
<< "'." << std::endl;
		std::cout << "Device Handle: " << deviceHandle << std::endl;

		// Create a file to store data.
		std::ofstream file(DATA_FILE, std::ios_base::binary);
		float* acquisitionBuffer = NULL;
		try
		{
			// Initialize acquisition members.
			//---------------------------------------------------------
-------------------------
			unsigned int numberOfChannelsToAcquire;
			UNICORN_GetNumberOfAcquiredChannels(deviceHandle,
&numberOfChannelsToAcquire);

			UNICORN_AMPLIFIER_CONFIGURATION configuration;
			errorCode = UNICORN_GetConfiguration(deviceHandle,
&configuration);
			HandleError(errorCode);
			char efectoSonido[100];
			char popUpCierre[100];
			char popUpAbrir[100];
			strcpy(efectoSonido, "omxplayer -o local --no-keys Ding-
sound-effect3.mp3");
			strcpy(popUpAbrir, "xmessage  -timeout 2 -center ABIERTOS");
			strcpy(popUpCierre, "xmessage -timeout 2 -center CERRADOS");
			cout << configuration.Channels;
			// Print acquisition configuration
			std::cout << std::endl << "Acquisition Configuration:" <<
std::endl;
			std::cout << "Frame Length: " << FRAME_LENGTH << std::endl;
			std::cout << "Number Of Acquired Channels: " <<
numberOfChannelsToAcquire << std::endl;
```

```
                    std::cout << "Data Acquisition Length: " <<
ACQUISITION_DURATION_S << "s" << std::endl;

                    // Allocate memory for the acquisition buffer.
                    int acquisitionBufferLength = (numberOfChannelsToAcquire + 1)
* FRAME_LENGTH;
                    acquisitionBuffer = new float[acquisitionBufferLength];

                    // Start data acquisition.
                    //-------------------------------------------------------------
-------------------------
                    errorCode = UNICORN_StartAcquisition(deviceHandle,
TESTSIGNAL_ENABLED);
                    HandleError(errorCode);
                    std::cout << std::endl << "Data acquisition started." <<
std::endl;

                    // Calculate number of get data calls.
                    int numberOfGetDataCalls = (int) (ACQUISITION_DURATION_S *
(UNICORN_SAMPLING_RATE / FRAME_LENGTH));

                    // Limit console update rate to max. 25Hz or slower.
                    int consoleUpdateRate = (int) ((UNICORN_SAMPLING_RATE /
FRAME_LENGTH) / 25.0f);
                    if (consoleUpdateRate == 0)
                           consoleUpdateRate = 1;

////////////////////////////////////////////////////////////////////////////

                    sleep(1); //nos esperamos un segundo antes de hacer el
estimulo
                    system(efectoSonido); // ejecutamos el estimulo auditivo
                    system(popUpCierre);// ejecutamos el estimulo visual
                    estimulo=1;

////////////////////////////////////////////////////////////////////////////


                    // Acquisition loop.
                    //-------------------------------------------------------------
-------------------------
                    for (int i = 0; i < numberOfGetDataCalls; i++)
                    {
                           // Receives the configured number of samples from the
Unicorn device and writes it to the acquisition buffer.
                           errorCode = UNICORN_GetData(deviceHandle,
FRAME_LENGTH, acquisitionBuffer, acquisitionBufferLength * sizeof(float));
                          HandleError(errorCode);
                        file <<  acquisitionBuffer[0] << ";" <<
acquisitionBuffer[1] << ";" << acquisitionBuffer[2] << ";" <<
acquisitionBuffer[3] << ";" << acquisitionBuffer[4] << ";" <<
acquisitionBuffer[5] << ";" << acquisitionBuffer[6] << ";" <<
acquisitionBuffer[7] <<";" <<  acquisitionBuffer[8] << ";" <<
acquisitionBuffer[9] << ";" << acquisitionBuffer[10] << ";" <<
acquisitionBuffer[11] << ";" << acquisitionBuffer[12] << ";" <<
acquisitionBuffer[13] << ";" << acquisitionBuffer[14] << ";" <<
acquisitionBuffer[15] <<";" << acquisitionBuffer[16] << ";" << estimulo << ";";

                          estimulo=0;
                          // Write data to file.
```

```
                        // Update console to indicate that the data
acquisition is running.
                        if (i%consoleUpdateRate == 0)
                                std::cout << ".";
                }

                // Stop data acquisition.
                //-------------------------------------------------------------
-------------------------
                errorCode = UNICORN_StopAcquisition(deviceHandle);
                HandleError(errorCode);
                std::cout << std::endl << "Data acquisition stopped." <<
std::endl;
        }
        catch (int errorCode)
        {
                // Write error code to console if something goes wrong.
                PrintErrorMessage(errorCode);
        }
        catch (...)
        {
                // Write error code to console if something goes wrong.
                std::cout << std::endl << "An unknown error occurred." <<
std::endl;
        }

        // Free memory of the acquisition buffer if necessary.
        if (acquisitionBuffer != NULL)
        {
                delete[] acquisitionBuffer;
                acquisitionBuffer = NULL;
        }

        // Free memory of the device buffer if necessary.
        if (availableDevices != NULL)
        {
                delete[] availableDevices;
                availableDevices = NULL;
        }

        // Close file.
        file.close();

        // Close device.
        //-------------------------------------------------------------------
-------------------
        errorCode = UNICORN_CloseDevice(&deviceHandle);
        HandleError(errorCode);
        std::cout << "Disconnected from Unicorn." << std::endl;
        //    varReproduccionSonora.join();
    }
    catch (int errorCode)
    {
        // Write error code to console if something goes wrong.
        PrintErrorMessage(errorCode);
    }
    catch (...)
    {
        // Write error code to console if something goes wrong.
```

```
			std::cout << std::endl << "An unknown error occurred." <<
std::endl;
		}

		std::cout << std::endl << "Press ENTER to terminate the application.";
		std::cin.clear();
		std::cin.ignore();
		getchar();
		return 0;
}

// The method throws an exception and forwards the error code if something goes
wrong.
//-------------------------------------------------------------------------------
------
void HandleError(int errorCode)
{
		if (errorCode != UNICORN_ERROR_SUCCESS)
		{
			throw errorCode;
		}
}

// The method prints an error messag to the console according to the error code.
//-------------------------------------------------------------------------------
------
void PrintErrorMessage(int errorCode)
{
		std::cout << std::endl << "An error occurred. Error Code: " << errorCode
<< " - ";
		switch (errorCode)
		{
		case UNICORN_ERROR_INVALID_PARAMETER:
			std::cout << "One of the specified parameters does not contain a
valid value.";
			break;
		case UNICORN_ERROR_BLUETOOTH_INIT_FAILED:
			std::cout << "The initialization of the Bluetooth adapter failed.";
			break;
		case UNICORN_ERROR_BLUETOOTH_SOCKET_FAILED:
			std::cout << "The operation could not be performed because the
Bluetooth socket failed.";
			break;
		case UNICORN_ERROR_OPEN_DEVICE_FAILED:
			std::cout << "The device could not be opened.";
			break;
		case UNICORN_ERROR_INVALID_CONFIGURATION:
			std::cout << "The configuration is invalid.";
			break;
		case UNICORN_ERROR_BUFFER_OVERFLOW:
			std::cout << "The acquisition buffer is full.";
			break;
		case UNICORN_ERROR_BUFFER_UNDERFLOW:
			std::cout << "The acquisition buffer is empty.";
			break;
		case UNICORN_ERROR_OPERATION_NOT_ALLOWED:
			std::cout << "The operation is not allowed.";
			break;
		case UNICORN_ERROR_INVALID_HANDLE:
			std::cout << "The specified connection handle is invalid.";
			break;
```

```
	case UNICORN_ERROR_GENERAL_ERROR:
		std::cout << "An unspecified error occurred.";
		break;
	default:
		break;
	}
	std::cout << std::endl;
}
```

Tiempo Real (Raspberry)

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <chrono>
#include <thread>
#include <string.h>
// Include unicorn header-file.
#include "unicorn.h"
using namespace std;
// Specifications for the data acquisition.
//------------------------------------------------------------------------------
------
#define DATA_FILE                       "data.bin"          // The name of the
file which is storing acquired data.
#define ACQUISITION_DURATION_S    10.0f               // 10 segundos por cada
repeticion de trama eeg
#define FRAME_LENGTH              1                       // The number of
samples acquired per get data call.
#define TESTSIGNAL_ENABLED        false               // Flag to enable or
disable testsignal.

// Function declarations.
//------------------------------------------------------------------------------
------
void HandleError(int errorCode);
void PrintErrorMessage(int errorCode);


// Metodo de lectura de los datos del casco usando el ejemplo proporcionado por
el fabricante
//------------------------------------------------------------------------------
------
// Main. Program entry point.
//------------------------------------------------------------------------------
------
int main()
```

```
{
	std::cout << "Unicorn Acquisition Example" << std::endl;
	std::cout << "---------------------------" << std::endl << std::endl;

	// Variable to store error codes.
	int errorCode = UNICORN_ERROR_SUCCESS;

	// Structure that holds the handle for the currecnt session.
	UNICORN_HANDLE deviceHandle = 0;

	try
	{
		// Get available devices.
		//-----------------------------------------------------------------
--------------------

		// Get number of available devices.
		unsigned int availableDevicesCount = 0;
		errorCode = UNICORN_GetAvailableDevices(NULL,
&availableDevicesCount, TRUE);
		HandleError(errorCode);

		if (availableDevicesCount < 1)
		{
			std::cout << "No device available. Please pair with a Unicorn
device first.";
			errorCode = UNICORN_ERROR_GENERAL_ERROR;
		}

		//Get available device serials.
		UNICORN_DEVICE_SERIAL *availableDevices = new
UNICORN_DEVICE_SERIAL[availableDevicesCount];
		errorCode = UNICORN_GetAvailableDevices(availableDevices,
&availableDevicesCount, true);
		HandleError(errorCode);

		//Print available device serials.
		std::cout << "Available devices:" << std::endl;
		for (unsigned int i = 0; i<availableDevicesCount; i++)
		{
			std::cout << "#" << i << ": " << availableDevices[i] <<
std::endl;
		}

		// Request device selection.
		std::cout << "\nSelect device by ID #";
		unsigned int deviceSelection;
		float estimulo;
		std::cin >> deviceSelection;
		if (deviceSelection >= availableDevicesCount || deviceSelection <
0)
			errorCode = UNICORN_ERROR_GENERAL_ERROR;

		HandleError(errorCode);

		// Open selected device.
		//-----------------------------------------------------------------
--------------------
		std::cout << "Trying to connect to '" <<
availableDevices[deviceSelection] << "'." << std::endl;
```

```
            errorCode =
UNICORN_OpenDevice(availableDevices[deviceSelection],&deviceHandle);
            HandleError(errorCode);

            std::cout << "Connected to '" << availableDevices[deviceSelection]
<< "'." << std::endl;
            std::cout << "Device Handle: " << deviceHandle << std::endl;

            // Create a file to store data.
            std::ofstream file(DATA_FILE, std::ios_base::binary);
            float* acquisitionBuffer = NULL;
            try
            {
                  // Initialize acquisition members.
                  //-------------------------------------------------------------
-------------------------
                  unsigned int numberOfChannelsToAcquire;
                  UNICORN_GetNumberOfAcquiredChannels(deviceHandle,
&numberOfChannelsToAcquire);

                  UNICORN_AMPLIFIER_CONFIGURATION configuration;
                  errorCode = UNICORN_GetConfiguration(deviceHandle,
&configuration);
                  HandleError(errorCode);
                  char efectoSonido[100];
                  char popUpCierre[100];
                  char popUpAbrir[100];
                  strcpy(efectoSonido, "omxplayer -o local --no-keys Ding-
sound-effect3.mp3");
                  strcpy(popUpAbrir, "xmessage  -timeout 2 -center ABIERTOS");
                  strcpy(popUpCierre, "xmessage -timeout 2 -center CERRADOS");
                  cout << configuration.Channels;
                  // Print acquisition configuration
                  std::cout << std::endl << "Acquisition Configuration:" <<
std::endl;
                  std::cout << "Frame Length: " << FRAME_LENGTH << std::endl;
                  std::cout << "Number Of Acquired Channels: " <<
numberOfChannelsToAcquire << std::endl;
                  std::cout << "Data Acquisition Length: " <<
ACQUISITION_DURATION_S << "s" << std::endl;

                  // Allocate memory for the acquisition buffer.
                  int acquisitionBufferLength = (numberOfChannelsToAcquire + 1)
* FRAME_LENGTH;
                  acquisitionBuffer = new float[acquisitionBufferLength];

                  // Start data acquisition.
                  //-------------------------------------------------------------
-------------------------
                  errorCode = UNICORN_StartAcquisition(deviceHandle,
TESTSIGNAL_ENABLED);
                  HandleError(errorCode);
                  std::cout << std::endl << "Data acquisition started." <<
std::endl;

                  // Calculate number of get data calls.
                  int numberOfGetDataCalls = (int) (ACQUISITION_DURATION_S *
(UNICORN_SAMPLING_RATE / FRAME_LENGTH));

                  // Limit console update rate to max. 25Hz or slower.
```

```
                    int consoleUpdateRate = (int) ((UNICORN_SAMPLING_RATE /
FRAME_LENGTH) / 25.0f);
                    if (consoleUpdateRate == 0)
                           consoleUpdateRate = 1;

//////////////////////////////////////////////////////////////////////////

                    sleep(1); //nos esperamos un segundo antes de hacer el
estimulo
                    system(efectoSonido); // ejecutamos el estimulo auditivo
                    system(popUpCierre);// ejecutamos el estimulo visual
                    estimulo=1;

//////////////////////////////////////////////////////////////////////////


                    // Acquisition loop.
                    //-------------------------------------------------------
-------------------------
                    for (int i = 0; i < numberOfGetDataCalls; i++)
                    {
                           // Receives the configured number of samples from the
Unicorn device and writes it to the acquisition buffer.
                           errorCode = UNICORN_GetData(deviceHandle,
FRAME_LENGTH, acquisitionBuffer, acquisitionBufferLength * sizeof(float));
                           HandleError(errorCode);
                        file <<  acquisitionBuffer[0] << ";" <<
acquisitionBuffer[1] << ";" << acquisitionBuffer[2] << ";" <<
acquisitionBuffer[3] << ";" << acquisitionBuffer[4] << ";" <<
acquisitionBuffer[5] << ";" << acquisitionBuffer[6] << ";" <<
acquisitionBuffer[7] <<";" <<  acquisitionBuffer[8] << ";" <<
acquisitionBuffer[9] << ";" << acquisitionBuffer[10] << ";" <<
acquisitionBuffer[11] << ";" << acquisitionBuffer[12] << ";" <<
acquisitionBuffer[13] << ";" << acquisitionBuffer[14] << ";" <<
acquisitionBuffer[15] <<";" << acquisitionBuffer[16] << ";" << estimulo << ";";

                           estimulo=0;
                           // Write data to file.


                           // Update console to indicate that the data
acquisition is running.
                           if (i%consoleUpdateRate == 0)
                                  std::cout << ".";
                    }

                    // Stop data acquisition.
                    //-------------------------------------------------------
-------------------------
                    errorCode = UNICORN_StopAcquisition(deviceHandle);
                    HandleError(errorCode);
                    std::cout << std::endl << "Data acquisition stopped." <<
std::endl;
             }
             catch (int errorCode)
             {
                    // Write error code to console if something goes wrong.
                    PrintErrorMessage(errorCode);
             }
             catch (...)
             {
```

```
                        // Write error code to console if something goes wrong.
                        std::cout << std::endl << "An unknown error occurred." <<
std::endl;
                }

                // Free memory of the acquisition buffer if necessary.
                if (acquisitionBuffer != NULL)
                {
                        delete[] acquisitionBuffer;
                        acquisitionBuffer = NULL;
                }

                // Free memory of the device buffer if necessary.
                if (availableDevices != NULL)
                {
                        delete[] availableDevices;
                        availableDevices = NULL;
                }

                // Close file.
                file.close();

                // Close device.
                //--------------------------------------------------------------------
--------------------
                errorCode = UNICORN_CloseDevice(&deviceHandle);
                HandleError(errorCode);
                std::cout << "Disconnected from Unicorn." << std::endl;
                //    varReproduccionSonora.join();
        }
        catch (int errorCode)
        {
                // Write error code to console if something goes wrong.
                PrintErrorMessage(errorCode);
        }
        catch (...)
        {
                // Write error code to console if something goes wrong.
                std::cout << std::endl << "An unknown error occurred." <<
std::endl;
        }

        std::cout << std::endl << "Press ENTER to terminate the application.";
        std::cin.clear();
        std::cin.ignore();
        getchar();
        return 0;
}

// The method throws an exception and forwards the error code if something goes
wrong.
//------------------------------------------------------------------------------
------
void HandleError(int errorCode)
{
        if (errorCode != UNICORN_ERROR_SUCCESS)
        {
                throw errorCode;
        }
}
```

```
// The method prints an error messag to the console according to the error code.
//------------------------------------------------------------------------------
------
void PrintErrorMessage(int errorCode)
{
	std::cout << std::endl << "An error occurred. Error Code: " << errorCode
<< " - ";
	switch (errorCode)
	{
	case UNICORN_ERROR_INVALID_PARAMETER:
		std::cout << "One of the specified parameters does not contain a
valid value.";
		break;
	case UNICORN_ERROR_BLUETOOTH_INIT_FAILED:
		std::cout << "The initialization of the Bluetooth adapter failed.";
		break;
	case UNICORN_ERROR_BLUETOOTH_SOCKET_FAILED:
		std::cout << "The operation could not be performed because the
Bluetooth socket failed.";
		break;
	case UNICORN_ERROR_OPEN_DEVICE_FAILED:
		std::cout << "The device could not be opened.";
		break;
	case UNICORN_ERROR_INVALID_CONFIGURATION:
		std::cout << "The configuration is invalid.";
		break;
	case UNICORN_ERROR_BUFFER_OVERFLOW:
		std::cout << "The acquisition buffer is full.";
		break;
	case UNICORN_ERROR_BUFFER_UNDERFLOW:
		std::cout << "The acquisition buffer is empty.";
		break;
	case UNICORN_ERROR_OPERATION_NOT_ALLOWED:
		std::cout << "The operation is not allowed.";
		break;
	case UNICORN_ERROR_INVALID_HANDLE:
		std::cout << "The specified connection handle is invalid.";
		break;
	case UNICORN_ERROR_GENERAL_ERROR:
		std::cout << "An unspecified error occurred.";
		break;
	default:
		break;
	}
	std::cout << std::endl;
}
```