



### Integrantes del Proyecto:

- Adell, Nicolas Fabian
- De Blasi, Luca
- Diaz Melion, Danilo Sebastian
- Gil Soria, Ian Lucas
- Montenegro, Luciano Nahuel
- Sojka, Santiago Alejandro

## Índice

<b>1. Introducción</b>	<b>5</b>
1.1. Resumen del proyecto . . . . .	5
1.2. Motivación . . . . .	5
1.3. Objetivos . . . . .	5
<b>2. ¿Quiénes somos?</b>	<b>6</b>
2.1. Contactos . . . . .	6
2.1.1. Adell Nicolás . . . . .	6
2.1.2. De Blasi Luca . . . . .	6
2.1.3. Diaz Melión Danilo . . . . .	7
2.1.4. Gil Soria Ian . . . . .	7
2.1.5. Montenegro Luciano . . . . .	7
2.1.6. Sojka Santiago . . . . .	7
<b>3. Desarrollo del Proyecto</b>	<b>8</b>
3.1. Metodología . . . . .	8
3.2. Historia del Arte . . . . .	8
3.2.1. Neuralink . . . . .	8
3.2.2. Ryan Lopez EEG . . . . .	8
3.2.3. OpenBCI . . . . .	10
3.2.4. Autodesk Instructables . . . . .	13

<b>4. Desarrollo Teórico</b>	<b>13</b>
4.1. Distinción de alimentaciones . . . . .	13
4.2. ¿Que es un electroencefalograma? . . . . .	14
4.3. ¿Que es un EEG? . . . . .	15
4.3.1. Componentes que tiene un EEG: . . . . .	15
4.3.2. Aplicaciones del EEG . . . . .	16
4.4. Cantidad de canales . . . . .	16
4.5. Sistema 10-20 . . . . .	17
4.5.1. ¿Para qué sirve? . . . . .	18
4.5.2. ¿Como se aplica? . . . . .	18
4.5.3. Pasos para aplicar el sistema 10-20 . . . . .	19
4.5.4. Aplicaciones del sistema 10-20 . . . . .	19
4.5.5. Colocacion de Electroodos . . . . .	19
4.6. Raspberry Pi 4 . . . . .	19
4.6.1. Caracteristicas de la RPi4 . . . . .	20
4.6.2. Uso de la Raspberry Pi 4 en la silla . . . . .	20
4.6.3. Programacion con la RPi4 . . . . .	21
4.6.4. Aplicaciones Generales de la RPi4 . . . . .	21
4.7. Comunicación UART . . . . .	21
4.8. Gráfico de Fourier . . . . .	22
4.8.1. ¿Para que sirve? . . . . .	23
4.8.2. ¿En que lo aplicamos? . . . . .	25
4.8.3. ¿Que es una feature? . . . . .	25
4.9. Filtros digitales . . . . .	25
4.9.1. ¿Para que los necesitamos si ya tenemos por Hardware? . . . . .	26
4.10. Gel conductor . . . . .	27
4.10.1. Cómo aplicar el gel conductor . . . . .	27
4.11. Jaula de Faraday . . . . .	27
4.11.1. Principio de Funcionamiento . . . . .	27
4.11.2. ¿Para que sirve? . . . . .	28
4.11.3. Como se utiliza . . . . .	28
4.11.4. Aplicacion en la silla . . . . .	28
4.11.5. Apantallamiento Electrico . . . . .	29
4.12. ¿Qué es un circuito de Offset? . . . . .	29
4.12.1. Utilización de Offsets . . . . .	29
4.13. Sistema de emergencia . . . . .	30
4.14. Tipos de alimentaciones . . . . .	30
4.15. Sistema de control . . . . .	30
4.16. Puentes H . . . . .	31
4.16.1. Puente H de potencia . . . . .	31
4.17. Sistema de recepción de señales por UART . . . . .	32
4.18. Control por PWM . . . . .	33
4.19. Estructura de la silla . . . . .	34
<b>5. Desarrollo Técnico</b>	<b>34</b>
5.1. Descripción del funcionamiento . . . . .	34
5.2. Funcionamiento del EEG . . . . .	35
5.3. Funcionamiento de los filtros HardWare . . . . .	35
5.3.1. Filtro Notch de 50 Hz . . . . .	35
5.3.2. Filtro pasa bajos de 100 Hz . . . . .	36
5.3.3. Filtro pasa altos de 0,5 Hz . . . . .	36
5.4. Funcionamiento de los filtros digitales . . . . .	36
5.5. Funcionamiento del Puente H . . . . .	37
5.5.1. Descripción de los Componentes . . . . .	37

5.6.	Funcionamiento del Sistema de Emergencia . . . . .	37
5.7.	Explicacion de los codigos utilizados . . . . .	38
5.7.1.	Explicacion de bias.py . . . . .	38
5.7.2.	Explicacion bias_ai.py . . . . .	40
5.7.3.	Explicacion bias_dsp.py . . . . .	47
5.7.4.	Explicacion bias_graphing.py . . . . .	48
5.7.5.	Explicacion bias_motors.py . . . . .	49
5.7.6.	Explicacion bias_reception.py . . . . .	51
5.7.7.	Explicacion reception.c . . . . .	53
<b>6.</b>	<b>Circuitos usados</b>	<b>55</b>
6.1.	Esquemático de los filtros . . . . .	55
6.2.	Esquemático del sistema de control . . . . .	55
6.3.	Esquemático del sistema Offset . . . . .	56
6.4.	Esquemáticos de los Puentes H . . . . .	58
6.5.	Esquemático del sistema de emergencia . . . . .	58
6.6.	Lista de materiales . . . . .	58
6.6.1.	EEG . . . . .	59
6.6.2.	Placa OffSet . . . . .	59
6.6.3.	Sistema de Emergencia . . . . .	59
6.6.4.	Puentes H . . . . .	60
6.6.5.	Placa reguladora de tensión . . . . .	60
<b>7.</b>	<b>Conclusiones</b>	<b>60</b>
7.1.	Limitaciones . . . . .	60
7.2.	Trabajo Futuro . . . . .	61
7.3.	Informacion Adicional . . . . .	61
7.3.1.	Programas Utilizados . . . . .	61
7.3.2.	Agradecimientos . . . . .	61
<b>8.</b>	<b>Referencias</b>	<b>62</b>
8.1.	Brain Computer Interface . . . . .	62
8.2.	Artificial Intelligence for Real-Time Decoding of Motor Commands from ECoG of Disabled Subjects for Chronic Brain-Computer Interfacing . . . . .	63
8.3.	Controlled Wheelchair Based on Brain Computer Interface using Neurosky Mindwave Mobile 2 . . . . .	63
8.4.	Brain-computer interface for electric wheelchair based on alpha waves of EEG signal	64
8.5.	A Neural Network Based Brain-Computer Interface for Classification of Movement Related EEG . . . . .	64
<b>9.</b>	<b>Lista de códigos</b>	<b>65</b>
9.1.	Códigos principales . . . . .	65
9.1.1.	bias.py . . . . .	65
9.1.2.	app.py . . . . .	66
9.2.	Inteligencia artificial . . . . .	67
9.2.1.	bias_ai.py . . . . .	67
9.3.	Filtrado y procesamiento de señales . . . . .	73
9.3.1.	bias_dsp.py . . . . .	73
9.3.2.	bias_dsp_task.py . . . . .	73
9.3.3.	bias_graphing.py . . . . .	73
9.4.	Motores . . . . .	74
9.4.1.	bias_motors.py . . . . .	74
9.5.	Recepción de señales . . . . .	77
9.5.1.	bias_reception.py . . . . .	77
9.5.2.	CMakeLists.txt . . . . .	78

9.5.3.	pico_sdk_import.cmake . . . . .	79
9.5.4.	reception.c . . . . .	80
9.6.	Página Web . . . . .	85
9.6.1.	Lenguaje utilizado en la página web y su código . . . . .	85

# 1. Introducción

BIAS es un módulo aplicable a una silla de ruedas que permite a los usuarios dirigirla mediante sus pensamientos, eliminando la necesidad de controles físicos tradicionales y optimizando la experiencia de movilidad. Este proyecto representa un avance significativo en el campo de la movilidad asistida, con el objetivo de mejorar la independencia y la calidad de vida de las personas con movilidad reducida.

Este sistema se basa en la avanzada tecnología de interfaz cerebro-computadora (BCI), la cual captura las señales neuronales del usuario y las traduce en comandos precisos para maniobrar la silla de ruedas sin esfuerzo. Además, el diseño modular y adaptable del sistema permite su integración en diversos modelos de sillas de ruedas y su personalización según las necesidades específicas de cada usuario.

La seguridad es una prioridad fundamental en este desarrollo, por lo que el módulo incorpora múltiples capas de redundancia y verificación de señales, asegurando que la silla de ruedas responda de manera precisa y confiable a las intenciones del usuario.

Aspiramos a revolucionar la forma en que las personas con movilidad reducida interactúan con su entorno, brindándoles una herramienta que amplía su capacidad para controlar su vida de manera más segura e independiente.

## 1.1. Resumen del proyecto

Este proyecto se centra en el desarrollo de una silla de ruedas controlada mediante señales cerebrales. El sistema se compone principalmente de los siguientes componentes: la silla de ruedas, un dispositivo EEG (electroencefalógrafo) y dos motores que permiten al usuario moverse en la dirección deseada.

El funcionamiento del sistema es el siguiente: el usuario se sienta en la silla y se coloca el EEG en la cabeza. A continuación, el usuario piensa en mover la lengua, las manos y los pies (mano izquierda, movimiento a la izquierda, mano derecha, movimiento a la derecha, la lengua es hacia atrás y los pies es hacia adelante). en la que desea moverse, y los motores responden en consecuencia.

El EEG tiene la función de leer las señales cerebrales del usuario. Estas señales se clasifican en las categorías de alpha, beta, gamma, delta y theta, y luego se someten a un proceso de filtrado para eliminar cualquier ruido. Posteriormente, las señales filtradas son analizadas por una Inteligencia Artificial (IA) desarrollada por nuestro equipo, la cual identifica patrones para determinar si el usuario desea moverse o detenerse.

Además, la silla está equipada con un sistema de emergencia que se activa en caso de lecturas erróneas o la detección de obstáculos. Este sistema de emergencia está compuesto por ultrasónicos que detectan la presencia de obstáculos frente a la silla.

## 1.2. Motivación

Como estudiantes de séptimo año, buscamos desarrollar un proyecto con el fin de cumplir con el requerimiento horario de prácticas profesionalizantes. Nuestra intención inicial fue encontrar el proyecto ideal en cuestiones de impacto social, enfocándonos en la mejora de la calidad de vida de aquellas personas con discapacidades. Bajo este marco, y tras una extensa investigación, proponemos crear un módulo aplicable a una silla de ruedas controlado mediante la actividad cerebral del usuario, utilizando tecnología de interfaces cerebro-computadora (BCI), para brindarles mayor autonomía, independencia y participación social. Creemos que este proyecto tiene el potencial de transformar la vida de las personas con movilidad motora reducida debido a enfermedades como ELA, esclerosis múltiple o cuadriplejía, permitiéndoles controlar su propio movimiento y mejorar significativamente su calidad de vida.

## 1.3. Objetivos

El objetivo de este proyecto es desarrollar una silla de ruedas motorizada que pueda ser controlada por la actividad cerebral del usuario. Esto beneficiaría a las personas con discapacidades motoras

severas para lograr independencia y movilidad, mejorando su calidad de vida y participación en la sociedad. De esta manera, estas personas las cuales no pueden caminar o tienen movilidad reducida, pueden gozar de la utilización de una silla de ruedas controlada por señales cerebrales sin la utilización de ambas manos y meramente por sus pensamientos, dándole libertad y autonomía al usuario.

## **2. ¿Quiénes somos?**

Somos el grupo BIAS (Brain Intelligence Artificial System), conformado por seis estudiantes de la especialidad de Aviónica del séptimo año, segunda división, comisión C, de la Escuela Técnica N° 7 IMPA "Taller Regional Quilmes". Nuestro equipo se encuentra comprometido en el desarrollo de un módulo adaptable para sillas de ruedas, para así poder brindarles mas independencia y movilidad.

### **2.1. Contactos**

En este apartado se detallarán los contactos a los integrantes del proyecto, adjuntando enlaces a sus redes:

#### **2.1.1. Adell Nicolás**



Instagram

LinkedIn

Gmail

#### **2.1.2. De Blasi Luca**



Instagram

LinkedIn

Gmail

**2.1.3. Diaz Melión Danilo**



Instagram

LinkedIn

Gmail

**2.1.4. Gil Soria Ian**



Instagram

LinkedIn

Gmail

**2.1.5. Montenegro Luciano**



Instagram

LinkedIn

Gmail

**2.1.6. Sojka Santiago**



Instagram

LinkedIn

Gmail

### 3. Desarrollo del Proyecto

En este capítulo se expondrá en detalle la metodología empleada para avanzar en el desarrollo del proyecto, incluyendo las fuentes de inspiración y los proyectos similares que se tomaron como referencia para el diseño de nuestros circuitos y la programación de los sistemas. Además, se incluirán imágenes ilustrativas de estos proyectos de referencia, con el fin de ofrecer una comprensión visual más completa. Al finalizar el capítulo, se presentarán y explicarán los resultados obtenidos, evaluando su eficacia en relación con los objetivos planteados.

#### 3.1. Metodología

La metodología aplicada en este proyecto se basa en la organización del equipo en dos grupos especializados. El primer grupo se encarga del desarrollo de los programas, siendo ejemplos de esto los filtros virtuales, programas de recepción y procesamiento, mientras que el segundo se dedica a la implementación de los motores, sistemas de emergencia y filtros. Esta división del trabajo permite una utilización más eficiente del tiempo, al abordar simultáneamente diversas áreas del proyecto, lo que resulta en un avance más significativo en periodos de tiempo más reducidos.

#### 3.2. Historia del Arte

En este capítulo se repasa todos los conceptos previamente vistos mediante los cuales pudimos empezar a conceptualizar la idea del proyecto.

##### 3.2.1. Neuralink

Uno de los casos que nos permitió aproximarnos a la conceptualización del proyecto fue el chip cerebral estilo implante de Neuralink, denominado Telepathy. Este dispositivo, desarrollado por la compañía de Elon Musk, tiene como objetivo facilitar la vida de las personas mediante la optimización de herramientas y su adaptación al cuerpo humano a través del implante. El chip crea una conexión entre el ser humano y dispositivos electrónicos, permitiendo, por ejemplo, controlar una puerta con traba eléctrica, desbloquear un teléfono móvil, e incluso monitorear el estado de ánimo de las personas.

Este ejemplo nos ofreció una perspectiva sobre la viabilidad de desarrollar un dispositivo, que, aunque no tan complejo, pudiera ser controlado por el propio cuerpo humano. A partir de esta premisa, iniciamos una investigación para explorar las posibilidades y opciones disponibles basadas en esta idea.

##### 3.2.2. Ryan Lopez EEG

Si bien nuestro motivo y objetivo principal era crear un dispositivo original e innovador, también nos propusimos investigar proyectos con cierta similitud. En este proceso, encontramos el repositorio de un estudiante que desarrolló un sistema capaz de detectar señales alfa y, en función de las reacciones o pensamientos del usuario, permitir el control a través de un test de concentración. Este test consistía en que el usuario debía mantener un estado de concentración mientras estudiaba un documento para un examen, en lo que se podría considerar un simulacro.

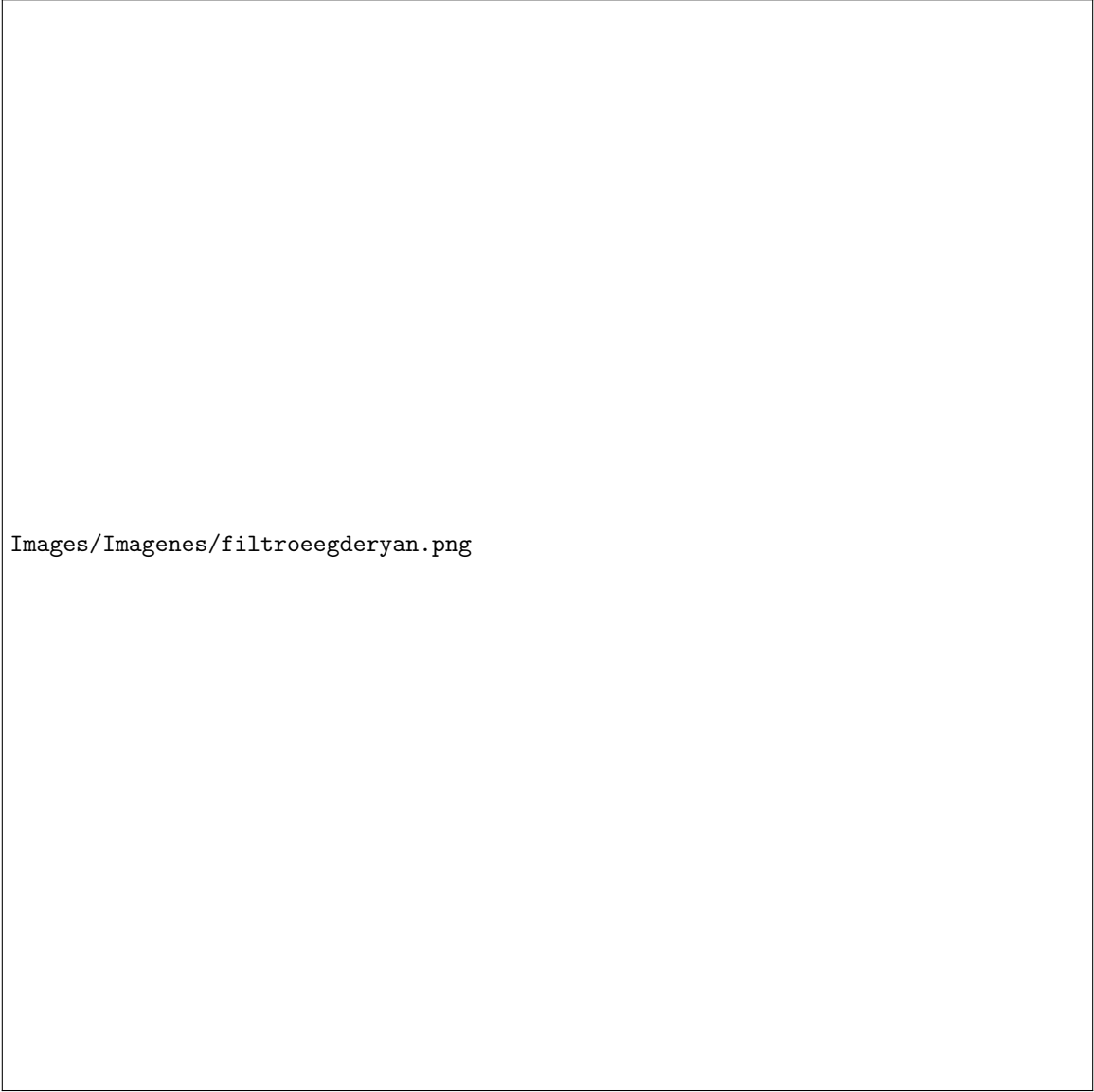
El sistema desarrollado por Ryanlopezz activaba una alarma si los niveles de concentración del usuario comenzaban a disminuir, basándose en un espectro de señales alfa transmitidas por un conjunto de electrodos EEG dorados conectados a la cabeza del estudiante. Además, el sistema realizaba otras dos pruebas: una de ellas consistía en escribir en código Morse mediante pensamientos, enviando "pulsos de concentración", y la otra permitía controlar el movimiento de un personaje en el famoso videojuego "Flappy Bird", ajustando el impulso para subir o bajar.

En el siguiente link se adjunta la página de GitHub de Ryan Lopez en donde se puede observar más a detalle los programas y archivos de su EEG.

Ryan Lopez GitHub

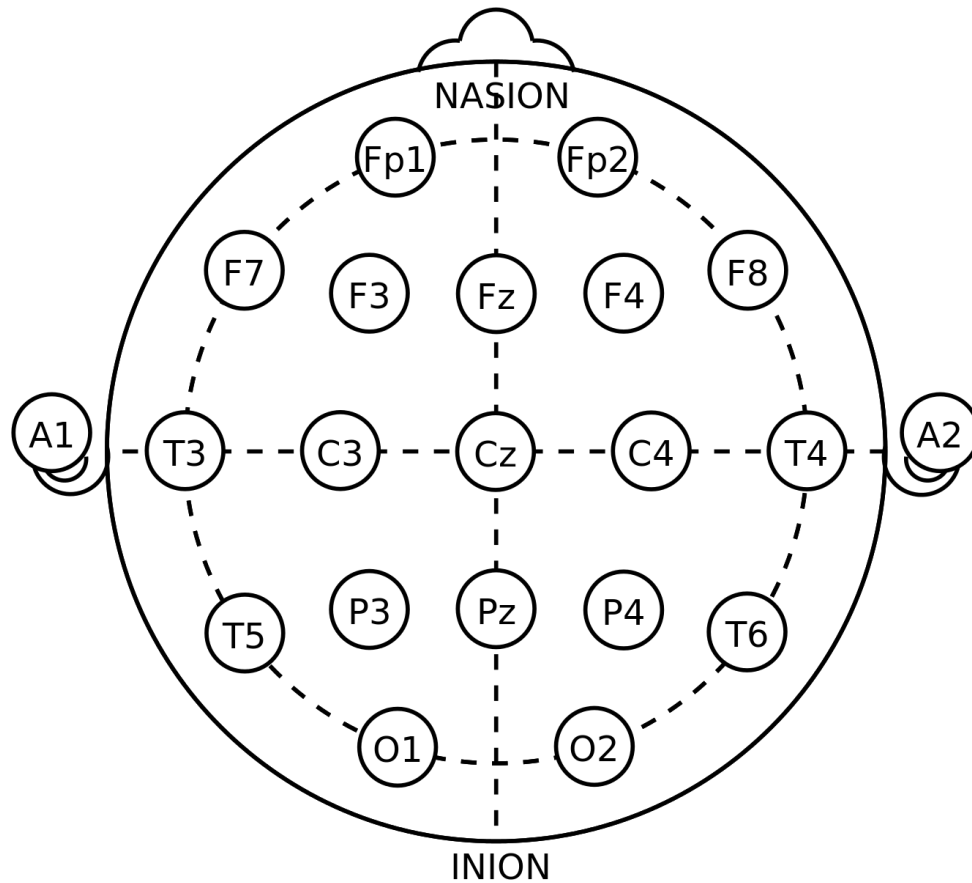


Gracias al mismo proyecto realizado por este usuario, pudimos desarrollar nuestro propio circuito para el sistema de filtrado, ya que íbamos a requerir otras especificaciones con diferentes funciones, tales como la entrada de señal: ya que esta debía ser mayor a la designada, al nosotros trabajar no solo con el espectro radioelectrico de las señales Alpha, sino con las principales señales que el cerebro produce al momento de realizar un pensamiento específico; Por otro lado nosotros íbamos a requerir no solo un sistema de filtros EEG sino 4 placas del mismo circuito, debido a la posibilidad de maniobrar controlando los 3 ejes de los motores de la silla de ruedas



Images/Imagenes/filtroeegderyan.png

Por otro lado nos dio una primera vista de la localización de los botones EEG como irían colocados en la cabeza:



Así mismo para el estudio del área de medicina y saber como el cerebro humano se encarga de transmitir ciertas señales eléctricas gracias a la actividad cerebral tuvimos en cuenta los actuales dispositivos de encefalógrafos, como se componen y actúan gracias a la actividad cerebral, bajo un espectro radioeléctrico determinado adjunto a la clasificación de las ondas transmitidas por el cerebro humano

### 3.2.3. OpenBCI

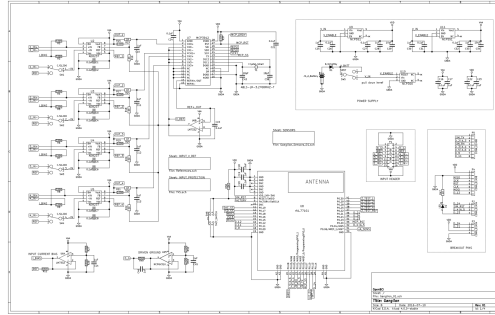
En nuestra búsqueda de conexiones para un microcontrolador y formas de desarrollar nuestro propio proyecto, descubrimos un sitio web llamado OpenBCI, una compañía especializada en la creación de dispositivos que integran tecnología con la interfaz del cuerpo humano. Esta empresa ha desarrollado un encefalógrafo casero, proporcionando dimensiones y parámetros estructurales que nos permitieron diseñar nuestra propia vincha/casco. Este dispositivo incorpora electrodos EEG, adaptados a la cabeza del usuario, como parte fundamental de nuestro proyecto "BIAS".

Images/Imagenes/vinchamarkiv.png

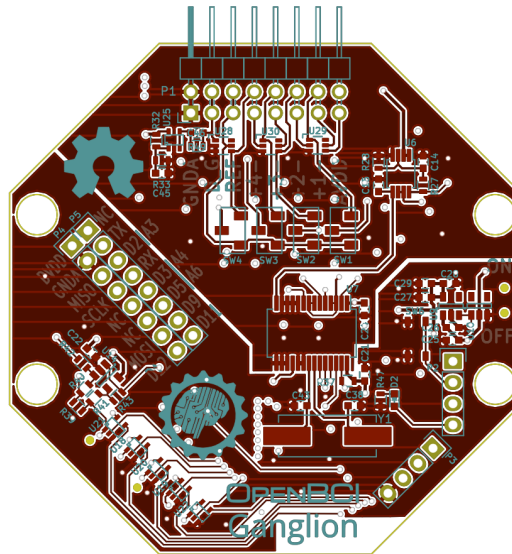
Además, al ser una compañía especializada en la creación de módulos de microcontroladores, nos proporcionaron orientación en cuanto a cómo realizar las conexiones entre la Raspberry Pi 4 que utilizamos y el sistema de filtrado EEG.

En los siguientes textos se adjuntan enlaces para ver la página oficial de OpenBCI y documentación de referencia correspondiente.

Página oficial de OpenBCI  
Documentación

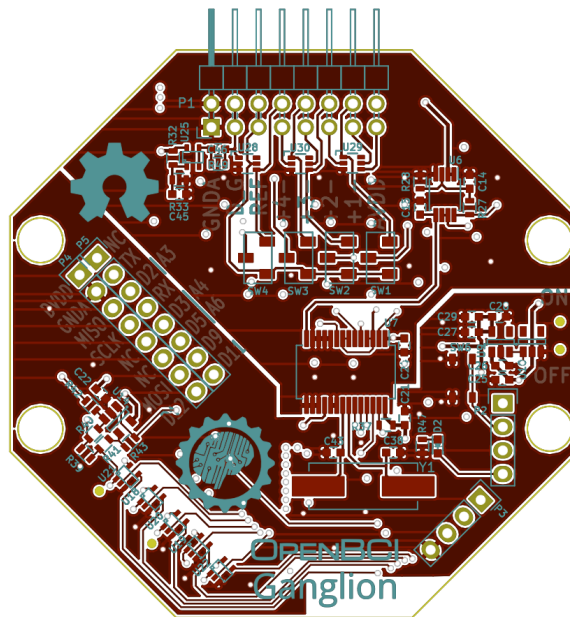


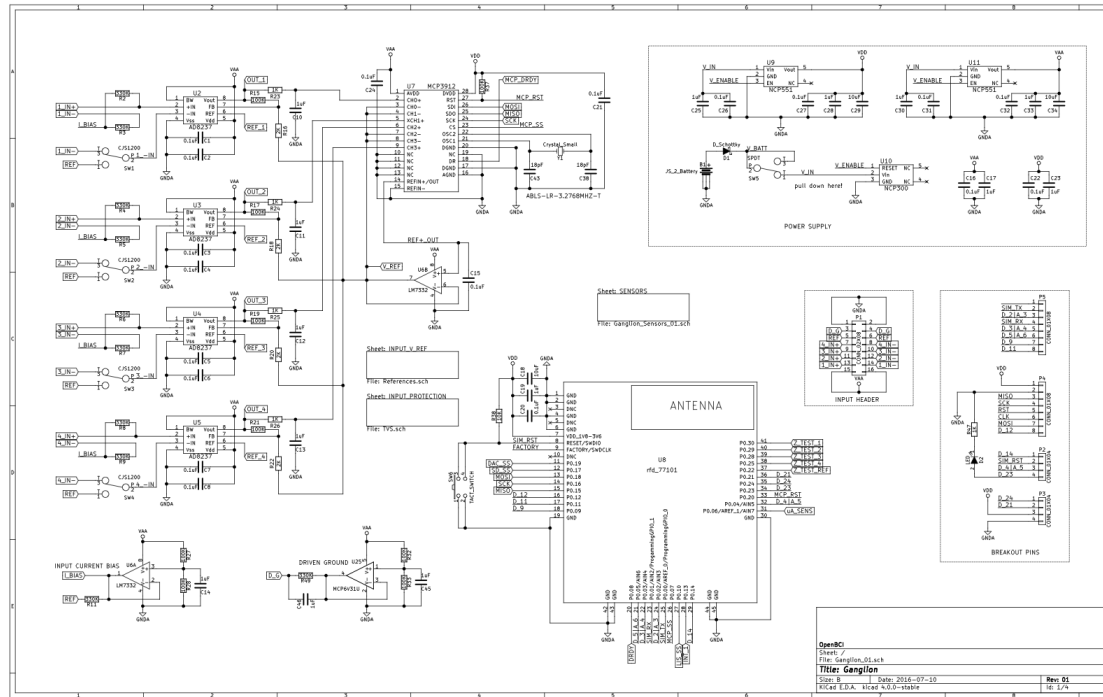
(a) Diseño Esquemático



(b) Diseño PCB

Figura 1: Diseño de microcontrolador de Open BCI





### 3.2.4. Autodesk Instructables

A partir de esta página, obtuvimos los primeros circuitos necesarios para el EEG, lo que nos permitió avanzar de manera significativa en el desarrollo del proyecto. Esta fuente de información fue fundamental para sentar las bases técnicas y nos brindó las herramientas necesarias para continuar con el diseño y la implementación del sistema.

## 4. Desarrollo Teórico

En este capítulo se describirán y desarrollarán todos los aspectos teóricos relacionados, justificando la elección de los componentes utilizados. Se abordarán teorías como la Jaula de Faraday y el sistema 10-20, además de ofrecer una explicación detallada sobre los tipos de señales empleadas, entre otros temas relevantes.

### 4.1. Distinción de alimentaciones

En la etapa de recepción de señales provenientes de los electrodos, se empleará una fuente de alimentación compuesta por baterías de 9 voltios. La elección de este tipo de alimentación responde a la necesidad de limitar la corriente en caso de que ocurra algún problema en el circuito, previniendo así posibles daños a los componentes electrónicos. Esta misma fuente de alimentación será utilizada para suministrar energía a la placa de offset, asegurando que todos los subsistemas relacionados con la recepción de señales operen bajo condiciones controladas y seguras.

Para la alimentación del microcontrolador RP2040 Zero, se utilizará un regulador de 5 voltios que se encargará de convertir el voltaje de 9 voltios a un nivel más adecuado de 5 voltios, garantizando el correcto funcionamiento del dispositivo. La estabilidad en el suministro de voltaje es crucial en

este tipo de sistemas, ya que cualquier fluctuación podría interferir con el procesamiento de las señales cerebrales y comprometer la precisión del sistema de control. Además, las distintas fuentes de alimentación están equipadas con condensadores conectados entre el terminal positivo y GND (tierra), con el objetivo de reducir las fluctuaciones en la tensión y minimizar la introducción de ruido en las señales. El ruido eléctrico, especialmente en sistemas de control y medición sensibles, puede degradar significativamente el rendimiento y la fiabilidad de los datos procesados.

El sistema de control de la silla de ruedas se alimentará mediante una batería de 24 voltios, la cual también suministrará energía al puente H y a los motores responsables del movimiento. Para garantizar la separación adecuada entre las distintas partes del sistema, se utilizará una masa adicional junto con un optoacoplador, que permitirá aislar eléctricamente la etapa de control de la etapa de potencia. Este desacoplamiento es esencial para evitar interferencias entre las señales de control y las corrientes más elevadas que circulan en el circuito de potencia.

Dentro del sistema de control, la Raspberry Pi 4 será alimentada mediante un regulador de 15 voltios, seguido de un segundo regulador que reducirá la tensión a 5 voltios. Este enfoque escalonado para la regulación del voltaje permite una mayor estabilidad en el suministro de energía, mejorando la eficiencia del sistema. De igual manera, se emplearán condensadores conectados en paralelo para reducir las posibles fluctuaciones de tensión y evitar la introducción de ruido, asegurando que la Raspberry Pi 4 funcione de manera óptima. La reducción del ruido es particularmente importante en sistemas basados en señales cerebrales, donde incluso pequeñas perturbaciones pueden alterar los resultados del procesamiento de señales EEG.

Finalmente, el puente H recibirá la alimentación directamente de la batería de 24 voltios. Para mantener una separación adecuada entre la etapa de control y la etapa de alimentación de los motores, se integrará otro optoacoplador, que garantizará que ambas secciones del circuito permanezcan eléctricamente aisladas. Sin embargo, aunque las masas de ambos sistemas no están directamente vinculadas en la placa, comparten el mismo punto de conexión en el borne de la batería de 24 voltios. Este diseño asegura que las distintas partes del sistema mantengan un nivel adecuado de aislamiento, mientras que las conexiones necesarias para la correcta operación del circuito se mantienen intactas.

## 4.2. ¿Que es un electroencefalograma?

Un electroencefalograma (EEG) es una prueba médica que registra la actividad eléctrica del cerebro. Esta actividad se mide a través de electrodos colocados en el cuero cabelludo, que detectan las señales eléctricas generadas por las neuronas mientras se comunican entre sí. Estas señales eléctricas se representan como ondas cerebrales en una gráfica que muestra diferentes frecuencias y patrones.

El EEG se utiliza comúnmente para:

- **Diagnóstico de trastornos neurológicos:** Como la epilepsia, convulsiones, trastornos del sueño, encefalopatías o daño cerebral.
- **Monitoreo del estado cerebral:** En procedimientos quirúrgicos, cuidados intensivos o investigaciones sobre coma y muerte cerebral.
- **Investigación neurológica y cognitiva:** Para estudiar la actividad cerebral en respuesta a estímulos externos, emociones o en procesos de toma de decisiones.
- **Interfaz cerebro-computadora (BCI):** Donde se usa para traducir la actividad cerebral en comandos para controlar dispositivos externos, como en tu proyecto de silla de ruedas controlada por EEG.

Las ondas cerebrales se clasifican en varios tipos según su frecuencia: delta, theta, alpha, beta, y gamma, cada una asociada a diferentes estados de conciencia y actividades cerebrales.

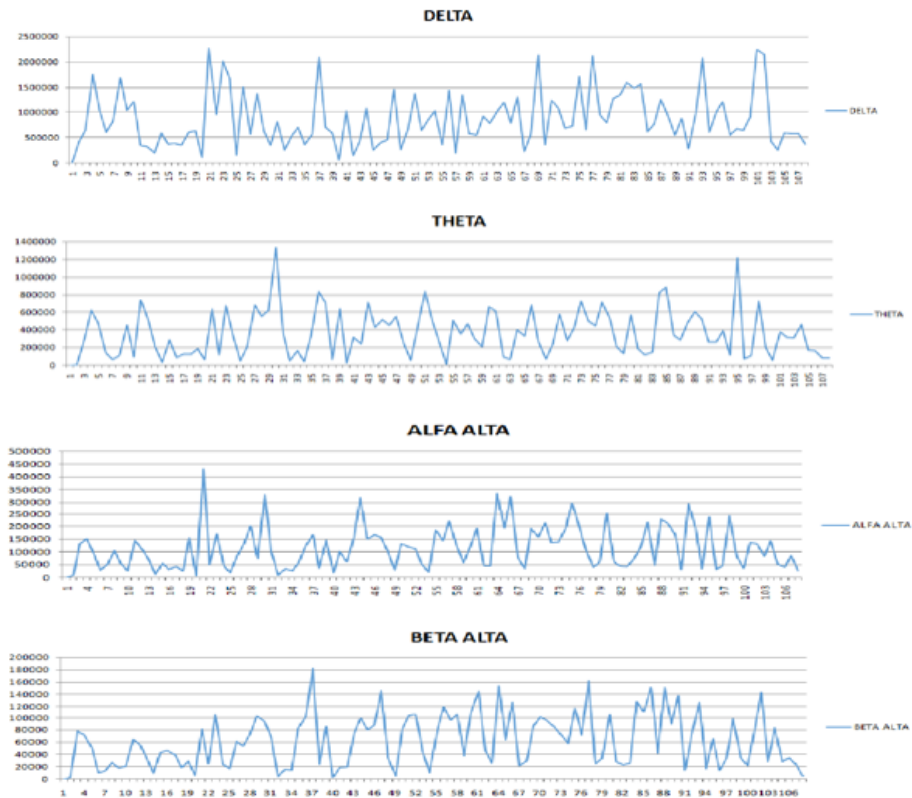


Figura 2: Vision de las Ondas Cerebrales en un electroencefalograma

### 4.3. ¿Que es un EEG?

Un electroencefalógrafo (EEG) es un dispositivo médico que se utiliza para registrar y medir la actividad eléctrica del cerebro. Su función principal es captar las señales eléctricas generadas por las neuronas y transformarlas en gráficos de ondas cerebrales, que se pueden analizar para evaluar la función cerebral, diagnosticar trastornos neurológicos y estudiar patrones de actividad cerebral. Las aplicaciones de los EEG incluyen investigaciones científicas, monitoreo clínico en neurocirugía, diagnóstico de epilepsia y el desarrollo de interfaces cerebro-computadora (BCI), como el control de dispositivos a través de señales cerebrales.

#### 4.3.1. Componentes que tiene un EEG:

- **Electrodos:** Son pequeños sensores colocados en el cuero cabelludo según el sistema internacional 10-20. Estos electrodos miden los cambios de voltaje producidos por la actividad eléctrica de las neuronas. Cada electrodo detecta la actividad en una región específica del cerebro, permitiendo un mapeo detallado de su actividad.
- **Amplificador:** Las señales eléctricas generadas por el cerebro son muy débiles, típicamente entre 0.5 a 100 V. El amplificador del EEG aumenta estas señales, permitiendo que puedan ser procesadas y visualizadas. En algunos dispositivos avanzados, se utilizan técnicas de amplificación diferencial para mejorar la relación señal-ruido.
- **Convertor analógico-digital:** Las señales eléctricas captadas son analógicas, por lo que se requiere un convertor analógico-digital para traducirlas a un formato digital que pueda ser procesado por computadoras. La resolución del ADC es crítica, ya que define la precisión con la que las señales son capturadas.

- **Software de procesamiento:** El software realiza el filtrado y procesamiento de las señales crudas para eliminar el ruido o artefactos no relacionados con la actividad cerebral, como los generados por movimientos o interferencias eléctricas. Además, permite el análisis en tiempo real de las ondas cerebrales (Delta, Theta, Alpha, Beta, Gamma) y su interpretación para diversas aplicaciones, como estudios clínicos o interfaces cerebro-computadora.
- **Pantalla/Gráficos:** El electroencefalógrafo muestra las señales eléctricas como gráficos de ondas cerebrales. Estas ondas reflejan diferentes estados mentales, como relajación, concentración o sueño, y son analizadas por especialistas para determinar la actividad neurológica o la respuesta ante estímulos específicos.

#### 4.3.2. Aplicaciones del EEG

- **Diagnóstico clínico:** Es utilizado para diagnosticar y monitorear trastornos neurológicos como la epilepsia, trastornos del sueño, y diversas encefalopatías.
- **Investigación neurocientífica:** Permite estudiar los mecanismos del cerebro, la plasticidad cerebral, y las respuestas neuronales ante diversos estímulos.
- **Interfaces cerebro-computadora (BCI):** Las señales EEG pueden ser procesadas para controlar dispositivos, como sillas de ruedas eléctricas o prótesis, mediante la detección de patrones específicos de actividad cerebral.
- **Monitoreo intraoperatorio:** Durante cirugías neurológicas, el EEG puede utilizarse para monitorear la actividad cerebral en tiempo real y evitar daños en áreas críticas del cerebro.

#### 4.4. Cantidad de canales

En el diseño de nuestra silla de ruedas controlada mediante señales cerebrales, hemos optado por un sistema de 4 canales con 9 electrodos (4 positivos, 4 negativos y 1 de masa) por varias razones clave, que refuerzan la precisión, la confiabilidad y la calidad de la señal capturada.

##### 1. Optimización de la Captura de Señales EEG

El cerebro humano emite diferentes tipos de ondas cerebrales (alfa, beta, gamma, delta y theta) que se distribuyen de manera no uniforme a lo largo del cuero cabelludo. Al utilizar 4 canales con 4 pares de electrodos (positivos y negativos) podemos capturar señales de distintas áreas cerebrales simultáneamente. Esto nos permite recoger información más completa y representativa de la actividad cerebral relevante para el control de la silla de ruedas.

##### 2. Mayor Resolución en la Separación de Patrones Cerebrales

El uso de múltiples electrodos garantiza una mayor resolución espacial en la detección de ondas cerebrales. Al disponer de 4 canales, podemos asociar cada canal a diferentes áreas de interés cerebral (como las regiones motoras o visuales), lo que aumenta la capacidad del sistema de IA para interpretar correctamente los patrones de ondas que se correlacionan con las intenciones de movimiento. Esto es crítico en una interfaz cerebro-computadora (BCI), donde la precisión es esencial para evitar errores en la dirección del movimiento.

##### 3. Minimización de Ruido y Artefactos

Cada par de electrodos está configurado en un modo diferencial, lo que nos ayuda a minimizar el ruido de fondo y los artefactos externos, como el ruido eléctrico del entorno o movimientos involuntarios. Los 4 electrodos negativos actúan como referencias de los positivos, de modo que la diferencia de potencial entre ellos permite eliminar parte del ruido que pudiera interferir en la calidad de la señal. Esto mejora considerablemente la relación señal/ruido y asegura una detección más precisa de las señales relevantes.



#### 4. Uso del Electrodo de Masa para Estabilizar las Mediciones

El electrodo de masa se utiliza para estabilizar el sistema y reducir el ruido eléctrico residual. Este electrodo está conectado a un punto de referencia neutro, lo que permite estabilizar los potenciales medidos en los otros electrodos, asegurando que las fluctuaciones o interferencias no alteren la calidad de las señales EEG. La masa es esencial para la precisión global del sistema, evitando desequilibrios de voltaje que podrían provocar lecturas erróneas.

#### 5. Equilibrio entre Complejidad y Procesamiento:

Optamos por 4 canales ya que representan un equilibrio adecuado entre complejidad técnica y capacidad de procesamiento. Más canales implicarían una mayor carga computacional para la Raspberry Pi 4, lo que podría requerir un hardware más potente o provocar retrasos en el procesamiento de las señales en tiempo real. Con esta configuración, logramos un sistema eficiente y suficiente para captar las señales necesarias sin comprometer la velocidad de respuesta del sistema de control de la silla.

#### 6. Compatibilidad con el 'Sistema 10-20'

La disposición de los electrodos sigue el estándar del 'Sistema 10-20' para garantizar una colocación uniforme y científicamente validada en el cuero cabelludo. Esto nos permite seleccionar puntos específicos que optimicen la captación de señales relevantes para nuestro sistema BCI, alineándonos con prácticas clínicas y de investigación que han demostrado ser efectivas en la medición de la actividad cerebral.

El uso de estos filtros permite separar de manera efectiva las señales cerebrales del ruido presente en el entorno. Esto garantiza que las señales capturadas sean lo suficientemente limpias y precisas para su procesamiento por la Raspberry Pi 4. El diseño del sistema de filtrado incluye tanto filtros de paso bajo como de paso alto, optimizados para cada banda de frecuencia, lo que mejora la fiabilidad del control basado en las ondas cerebrales.

### 4.5. Sistema 10-20

El sistema 10-20 es un método internacionalmente estandarizado para la colocación de electrodos en el cuero cabelludo durante un electroencefalograma (EEG). Este sistema asegura que los electrodos se coloquen de manera uniforme y precisa en relación con los puntos anatómicos del cráneo, permitiendo una medición consistente y reproducible de la actividad eléctrica cerebral.

El nombre "10-20" proviene de las distancias porcentuales entre los electrodos: Los electrodos se colocan en posiciones que están al 10 % o al 20 % de la distancia total entre puntos anatómicos clave del cráneo. Puntos anatómicos clave de referencia:

- **Nasion:** El punto entre la frente y la nariz.
- **Inion:** El punto más prominente en la parte posterior de la cabeza.
- **Puntos preauriculares:** Justo por encima de los oídos, a cada lado de la cabeza.

Las posiciones de los electrodos se distribuyen sobre la cabeza tomando como referencia estos puntos para cubrir todo el cuero cabelludo de forma equidistante.

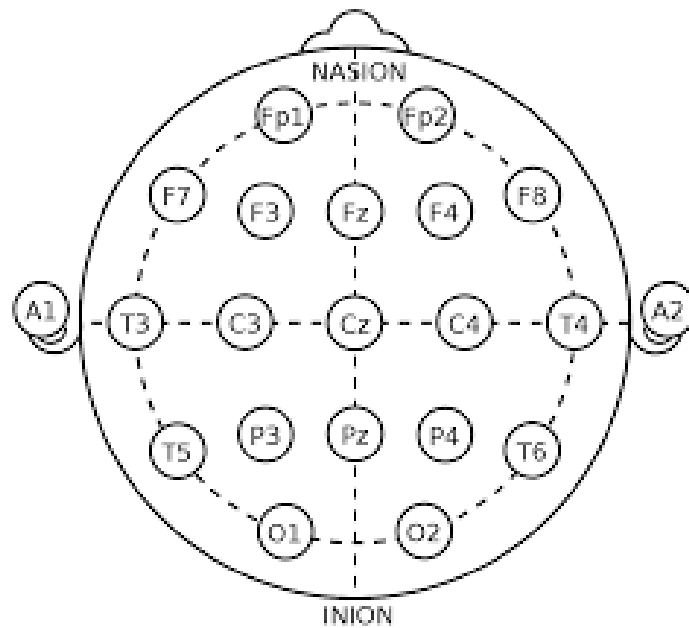


Figura 3: Puntos de medición de Ondas Cerebrales

#### 4.5.1. ¿Para qué sirve?

El sistema 10-20 se utiliza para garantizar una colocación precisa y consistente de los electrodos en los estudios de EEG. Algunos de sus beneficios incluyen:

- **Estandarización global:** Permite que los estudios de EEG sean comparables en diferentes laboratorios y entre diferentes pacientes.
- **Cobertura completa del cerebro:** El sistema asegura que se cubran adecuadamente diferentes áreas del cerebro (frontal, parietal, occipital, temporal).
- **Detección de anomalías:** Facilita la identificación de regiones del cerebro donde pueden estar ocurriendo fenómenos anormales, como descargas epilépticas, actividad lenta anormal o asimetrías en las ondas cerebrales.

#### 4.5.2. ¿Como se aplica?

En un estudio de EEG, se colocan electrodos en el cuero cabelludo siguiendo estas posiciones predeterminadas en el sistema 10-20:

1. Puntos y nomenclatura: Cada posición de electrodo tiene una nomenclatura que se utiliza para identificar las regiones del cerebro de las que se está registrando la actividad.
  - F: Frontal (F3, F4, Fz, etc.)
  - C: Central (C3, C4, Cz, etc.)
  - P: Parietal (P3, P4, Pz, etc.)
  - O: Occipital (O1, O2)
  - T: Temporal (T3, T4, etc.)
  - Los números impares (1, 3, 5, 7) se asignan al hemisferio izquierdo, mientras que los números pares (2, 4, 6, 8) se asignan al hemisferio derecho. La letra "z" indica posiciones centrales (a lo largo de la línea media de la cabeza).

#### 4.5.3. Pasos para aplicar el sistema 10-20

1. **Medición del cráneo:** Se mide la distancia desde el nasion al inion y desde un punto preauricular al otro. Luego, se calculan los puntos intermedios en un 10 o 20 de esas distancias para determinar las ubicaciones exactas de los electrodos.
2. **Colocación de los electrodos:** Se colocan electrodos en los puntos marcados, asegurando una cobertura uniforme de la cabeza. Esto puede hacerse usando una gorra de EEG con posiciones ya marcadas o mediante la medición manual.
3. **Registro de la actividad:** Una vez colocados los electrodos, se inicia el registro de la actividad cerebral. Los electrodos miden las diferencias de potencial eléctrico entre las regiones del cerebro cubiertas por los electrodos.

#### 4.5.4. Aplicaciones del sistema 10-20

- **Diagnóstico clínico:** El sistema se usa comúnmente en estudios de EEG para diagnosticar condiciones neurológicas como la epilepsia, trastornos del sueño, y daño cerebral.
- **Investigación:** En estudios de neurociencia, el sistema 10-20 facilita la comparación de la actividad cerebral entre sujetos, ya que la ubicación de los electrodos está estandarizada.
- **Interfaces cerebro-computadora (BCI):** En proyectos de BCI, se utiliza el sistema 10-20 para colocar electrodos en áreas específicas del cerebro que controlan movimientos o funciones específicas.

#### 4.5.5. Colocacion de Electrodo

Para llevar a cabo el registro de las señales EEG, es necesario disponer adecuadamente los electrodos. En nuestro proyecto, utilizamos un total de 9 electrodos, distribuidos en 4 canales: 4 electrodos positivos, 4 negativos y 1 de masa.

Los puntos de medición seleccionados para la colocación de los electrodos son los siguientes: Fp1, Fp2, F7, F8, T3, T4, O1, O2 y A2. Es importante señalar que los electrodos positivos se colocan en el lado derecho del cuero cabelludo (Fp2, F8, T4 y O2), mientras que los electrodos negativos se ubican en el lado izquierdo (Fp1, F7, T3 y O1). El electrodo de referencia o masa se posiciona en el lóbulo de la oreja, en el punto A2.

### 4.6. Raspberry Pi 4

La Raspberry Pi 4 es una computadora de bajo costo y tamaño compacto que se utiliza en una gran variedad de aplicaciones, especialmente en proyectos de electrónica, IoT (Internet of Things), y sistemas embebidos. Fue desarrollada por la Fundación Raspberry Pi para fomentar la educación en ciencias de la computación y se ha popularizado en proyectos de prototipado debido a su flexibilidad, potencia, y precio accesible.

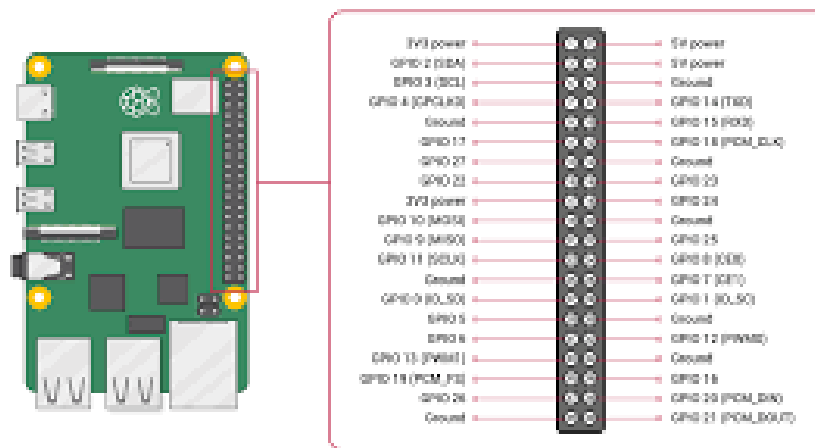


Figura 4: Raspberry Pi 4 PinOut

#### 4.6.1. Características de la RPi4

- **Procesador:** Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) de 64 bits a 1.5GHz.
- **Memoria RAM:** Existen variantes con 2GB, 4GB y 8GB de RAM LPDDR4.
- **Puertos USB:** 2 puertos USB 3.0 y 2 puertos USB 2.0.
- **Conectividad:** Ethernet Gigabit, WiFi 802.11ac, Bluetooth 5.0.
- **Salidas de video:** Dos puertos micro-HDMI, capaces de soportar resoluciones 4K.
- **Almacenamiento:** No tiene disco duro interno, pero se utiliza una tarjeta microSD para almacenar el sistema operativo y los archivos.
- **Puertos GPIO:** 40 pines GPIO (General Purpose Input/Output) para conectar diversos componentes electrónicos, como sensores, actuadores y módulos de comunicación.
- **Alimentación:** Requiere una fuente de alimentación de 5V/3A mediante un puerto USB-C.

#### 4.6.2. Uso de la Raspberry Pi 4 en la silla

Los pines GPIO de la Raspberry Pi 4 permiten interactuar con una variedad de componentes, como sensores, actuadores y módulos de comunicación. En nuestro proyecto, utilizamos estos pines para:

- **Control de Motores:** A través de los pines GPIO, enviamos señales a los controladores de motores que ajustan la velocidad y dirección de los mismos, permitiendo los movimientos precisos de la silla de ruedas.
- **Recepción de Señales EEG:** Las señales EEG provenientes de los electrodos colocados en el cuero cabelludo se procesan con un convertidor ADC (convertidor de señal analógica a digital) y se envían a la Raspberry Pi para su análisis y clasificación en tiempo real.
- **Sistema de Emergencia:** La Raspberry Pi se encarga de controlar los sensores ultrasónicos ubicados en diferentes puntos de la silla de ruedas, activando LEDs y un buzzer cuando se detectan obstáculos en la trayectoria.

#### 4.6.3. Programacion con la RPi4

La programación de la Raspberry Pi 4 se realiza principalmente en Python debido a su versatilidad y al soporte de bibliotecas específicas, como gpiozero, que facilita el control de los pines GPIO. En nuestro proyecto, Python se utiliza para las siguientes funciones:

- **Procesamiento de Señales EEG:** Implementamos algoritmos que procesan las señales EEG en tiempo real utilizando filtros digitales y técnicas de análisis, como CAR (Common Average Reference) y ICA (Independent Component Analysis), para identificar los comandos que controlan la dirección de la silla.
- **Control de Motores y Sensores:** Se desarrolla un programa para controlar los motores y gestionar el sistema de emergencia, combinando datos de los sensores ultrasónicos con los comandos derivados de las señales EEG.

Gracias a su procesador de cuatro núcleos y opciones de memoria de hasta 8 GB, la Raspberry Pi 4 puede ejecutar múltiples tareas simultáneamente. Esto permite que el sistema realice el procesamiento de señales EEG, el control de motores y la gestión del sistema de emergencia sin interrupciones. Para maximizar la eficiencia, se han implementado técnicas de multithreading en el código, lo que permite que diferentes procesos se ejecuten en paralelo.

#### 4.6.4. Aplicaciones Generales de la RPi4

Además de su uso en nuestro proyecto de silla de ruedas, la Raspberry Pi 4 tiene múltiples aplicaciones, entre las que se destacan:

- **Automatización y Robótica:** Control de sistemas de robótica, como vehículos autónomos o robots, con la capacidad de gestionar múltiples sensores y motores simultáneamente.
- **Centros de Medios:** Configuración de servidores multimedia utilizando herramientas como Kodi, para la reproducción de videos y música.
- **IoT (Internet of Things):** Monitorización y control de dispositivos conectados a internet, como sistemas de seguridad, domótica y más.
- **Estaciones de Monitoreo:** Implementación de estaciones meteorológicas o de monitorización ambiental utilizando sensores conectados a los pines GPIO.
- **Proyectos Educativos:** Gracias a su bajo costo y la abundancia de recursos educativos, la Raspberry Pi es ideal para enseñar conceptos de programación, electrónica y robótica.

#### 4.7. Comunicación UART

El UART (Universal Asynchronous Receiver-Transmitter) es un periférico de hardware utilizado para la comunicación serie. Su principal función es permitir la transmisión y recepción de datos de manera asíncrona, lo que implica que no requiere de una señal de clock compartida entre el transmisor y el receptor. A continuación, se destacan las características principales de este tipo de comunicación:

- **Comunicación serie:** Los datos se transmiten de forma secuencial, bit a bit, a través de una línea de transmisión (TX) y se reciben mediante una línea de recepción (RX).
- **Asincronía:** Al no utilizar una señal de clock constante, ambos dispositivos deben estar configurados con la misma velocidad de transmisión (tasa de baudios) para asegurar la correcta recepción de los datos.
- **Formato de datos:** Cada paquete de datos transmitido generalmente incluye un bit de inicio (indicando el comienzo de la transmisión), entre 5 y 9 bits de datos, un bit opcional de paridad (para detección de errores), y uno o dos bits de parada.

- **Velocidad de transmisión:** La velocidad de transmisión de la información, conocida como tasa de baudios, se refiere al número de bits transmitidos por segundo.

La implementación de la comunicación UART en nuestro proyecto se utiliza para la recepción y procesamiento de señales. Específicamente, en la RP2040 Zero, enviamos las lecturas de los cuatro canales del ADC (Convertor Analógico-Digital) en formato JSON durante un período de tiempo definido y con una frecuencia de muestreo específica a la Raspberry Pi 4.

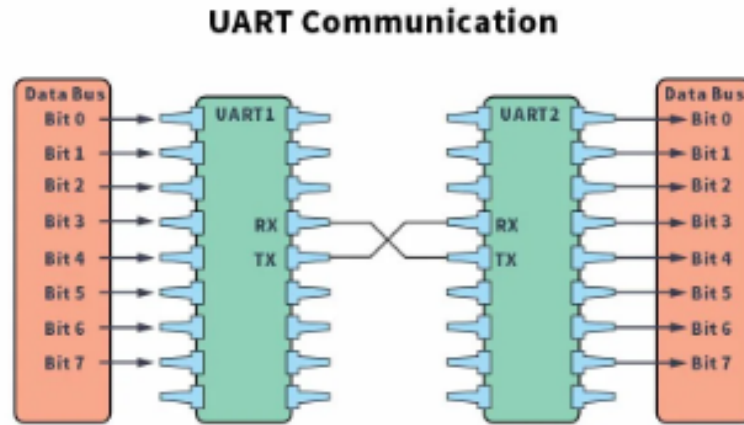


Figura 5: Comunicación UART

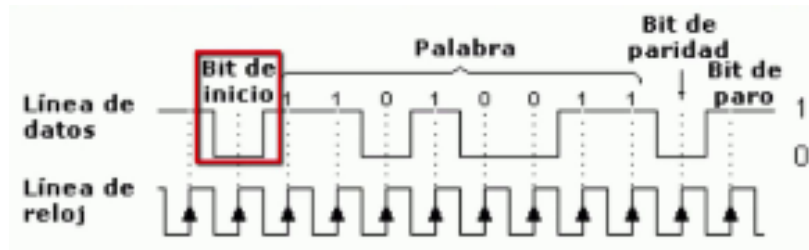


Figura 6: Envío de datos de UART

#### 4.8. Gráfico de Fourier

El gráfico de Fourier está basado en la Transformada de Fourier (TF), que es una operación matemática que descompone una señal en sus componentes sinusoidales de diferentes frecuencias. Este método es especialmente útil para analizar señales periódicas y no periódicas.

La Transformada de Fourier dicta que:

Dada una señal  $x(t)$  en el dominio del tiempo, la transformada de Fourier  $X(f)$  se define como:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} dt$$

Donde:

- $x(t)$  es la señal en el dominio del tiempo.
- $X(f)$  es la señal transformada en el dominio de la frecuencia.
- $f$  es la frecuencia (Hz).
- $j$  es la unidad imaginaria
- $e^{-j2\pi ft}$  es la función compleja que representa la descomposición en sinusoides.

El gráfico de Fourier representa la magnitud o la fase de la transformada de Fourier. Este gráfico tiene dos componentes importantes:

1. **Espectro de Magnitud:** Se representa gráficamente la magnitud de cada frecuencia presente en la señal. En el eje horizontal está la frecuencia  $f$ , y en el eje vertical está la magnitud  $|X(f)|$ , que indica cuánta energía o potencia de la señal está concentrada en esa frecuencia específica.

$$|X(f)| = \sqrt{\text{Re}(X(f))^2 + \text{Im}(X(f))^2} \quad (1)$$

2. **Espectro de Fase:** Indica el desfase de cada frecuencia con respecto a la señal original. En el eje horizontal también está la frecuencia  $f$ , y en el eje vertical está el ángulo de fase  $\theta(f)$ , que puede calcularse como:

$$\theta = \tan^{-1} \left( \frac{\text{Im}(X(f))}{\text{Re}(X(f))} \right) \quad (2)$$

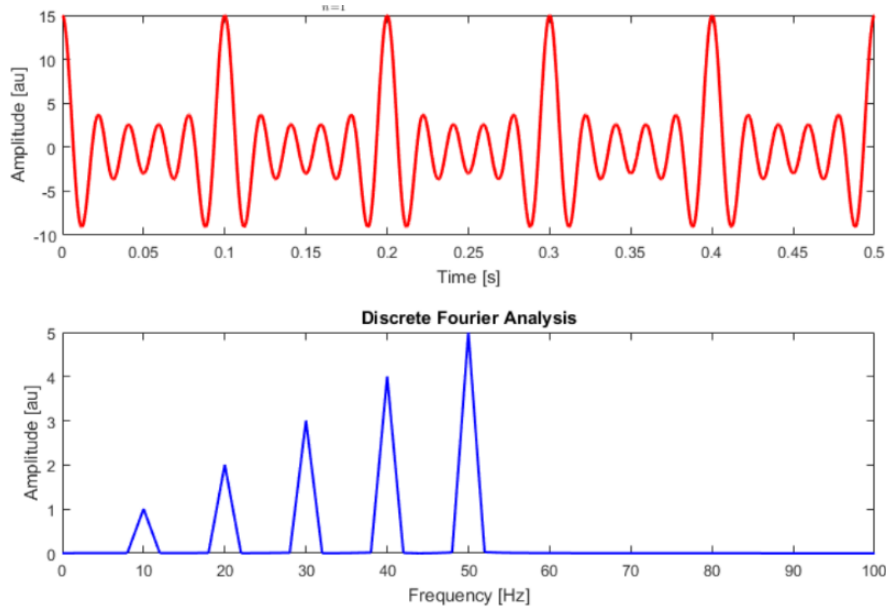


Figura 7: Grafico de Fourier

#### 4.8.1. ¿Para que sirve?

El grafico de Fourier es una herramienta que sirve para analizar señales en el dominio de la frecuencia.

### 1. Análisis de frecuencias:

- **Descomposición de señales complejas:** Permite descomponer una señal en sus componentes de frecuencia individuales. Esto es útil cuando quieres saber qué frecuencias están presentes en una señal y su importancia relativa.
- **Identificación de frecuencias dominantes:** El gráfico te muestra cuáles son las frecuencias que más contribuyen a la señal original, lo que es esencial para entender fenómenos periódicos y patrones.

### 2. Filtrado de señales:

- **Diseño de filtros:** Si necesitas eliminar ruido o componentes no deseados de una señal, el gráfico de Fourier ayuda a identificar las frecuencias del ruido o interferencia. Luego puedes diseñar un filtro que elimine esas frecuencias.
- **Filtrado de bandas:** Puedes aplicar filtros pasa-bajas, pasa-altas o de banda (pasa-banda o elimina-banda) a una señal. El gráfico te muestra exactamente qué frecuencias están siendo afectadas.

### 3. Compresión de datos:

- **Eliminación de información redundante:** En algunas señales (como imágenes o audio), las frecuencias más altas o muy bajas pueden no ser necesarias. Usando el gráfico de Fourier, puedes identificar esas frecuencias y eliminarlas para comprimir los datos sin perder información relevante.
- **Compresión de audio e imágenes:** Muchos algoritmos de compresión (como JPEG o MP3) utilizan la Transformada de Fourier o versiones discretas de ella para eliminar componentes de alta frecuencia que son menos perceptibles para el ser humano.

### 4. Análisis de vibraciones y ondas:

- **Detección de fallos en maquinaria:** En ingeniería mecánica y de control, el gráfico de Fourier es utilizado para analizar vibraciones y detectar problemas como desbalances, resonancias, o desgastes en partes móviles. Las frecuencias anómalas pueden señalar fallos.
- **Análisis de señales acústicas:** Permite estudiar el espectro de frecuencias en señales de audio. Por ejemplo, para identificar las características de sonidos musicales o voces.

### 5. Procesamiento de señales EEG:

- Para el análisis de señales cerebrales (EEG), el gráfico de Fourier es vital para identificar y separar las diferentes bandas de frecuencias asociadas con diferentes actividades cerebrales (como las ondas delta, theta, alpha, beta y gamma).
- **Monitoreo de actividad cerebral:** En neurociencia, el análisis de Fourier se utiliza para analizar patrones de actividad cerebral, detectar anomalías y realizar interfaces cerebro-máquina (como el control de la silla de ruedas en tu proyecto).

### 6. Análisis de señales en telecomunicaciones:

- **Ancho de banda:** El gráfico de Fourier permite determinar el ancho de banda necesario para transmitir una señal sin pérdida de información.
- **Modulación y demodulación:** En sistemas de comunicaciones, la modulación de señales se realiza en el dominio de la frecuencia. El gráfico de Fourier te permite visualizar cómo una señal ha sido modulada para su transmisión y cómo se puede demodular para su interpretación.



## 7. Análisis de imágenes:

- **Reconocimiento de patrones:** En procesamiento de imágenes, la transformada de Fourier se utiliza para identificar patrones repetitivos o estructuras de frecuencia en una imagen. Es útil en tareas como la detección de bordes o compresión de imágenes.
- **Eliminación de ruido:** Permite eliminar patrones específicos de ruido en imágenes filtrando frecuencias indeseadas.

### 4.8.2. ¿En que lo aplicamos?

Se emplea el Análisis de Fourier para identificar el espectro de frecuencias de las señales cerebrales, permitiendo la diferenciación de las ondas alpha, beta, gamma, delta y theta. Posteriormente, se aplica la Transformada Inversa de Fourier para reconstruir las señales en el dominio temporal, recuperando su forma original.

Una vez que las señales han sido diferenciadas por bandas de frecuencia, el algoritmo de machine learning procede a extraer las características (features) de cada tipo de onda a partir de una representación en el dominio de la frecuencia y del tiempo. Estas características son utilizadas para realizar la predicción del movimiento deseado en la silla de ruedas.

### 4.8.3. ¿Que es una feature?

Una feature es una propiedad medible o una característica distintiva de un fenómeno. En el contexto de este proyecto, las features se refieren a las propiedades específicas de cada señal cerebral, clasificadas por tipo de onda (alpha, beta, gamma, delta y theta). Estas propiedades son extraídas para capturar la información relevante que permitirá al sistema de aprendizaje automático identificar patrones en las señales y realizar predicciones relacionadas con el control de la silla de ruedas.

## 4.9. Filtros digitales

Un filtro digital es una herramienta utilizada en el procesamiento de señales para modificar o mejorar ciertos aspectos de una señal digital, eliminando o atenuando componentes no deseados, como ruido, o destacando componentes útiles. Los filtros digitales operan sobre señales discretizadas (muestreadas) y utilizan algoritmos matemáticos para realizar diversas transformaciones.

Existen diferentes tipos de filtros digitales, según su función:

- **Filtro pasa bajas:** Permite el paso de frecuencias bajas y atenúa las frecuencias altas.
- **Filtro pasa altas:** Permite el paso de frecuencias altas y atenúa las bajas.
- **Filtro pasa banda:** Permite el paso de un rango específico de frecuencias.
- **Filtro elimina banda (Notch):** Atenúa un rango específico de frecuencias, como en el caso de eliminar interferencias a 50-60 Hz.

Los filtros digitales pueden ser FIR (Finite Impulse Response) o IIR (Infinite Impulse Response):

- Los filtros FIR tienen una respuesta finita y son más estables, pero pueden necesitar más recursos de procesamiento.
- Los filtros IIR tienen una respuesta infinita y suelen ser más eficientes en términos de recursos, pero pueden ser menos estables si no se diseñan correctamente.

Los filtros digitales son ampliamente usados en aplicaciones como procesamiento de audio, señales biológicas (como EEG), imágenes y telecomunicaciones.

#### 4.9.1. ¿Para que los necesitamos si ya tenemos por Hardware?

La combinación de filtros físicos (implementados en hardware) con filtros digitales (implementados en software) presenta una serie de ventajas significativas, incluso si ya se han instalado filtros físicos en el sistema. Aunque los filtros basados en hardware permiten un filtrado efectivo de las señales EEG, los filtros digitales proporcionan una flexibilidad y capacidades adicionales que no siempre son posibles de lograr mediante hardware exclusivamente. A continuación, se detallan las razones por las cuales puede ser beneficioso incorporar filtros digitales a la salida de los filtros físicos:

##### 1. Ajuste Preciso y Mayor Flexibilidad

Los filtros de hardware están limitados a sus parámetros iniciales, ya que una vez construidos con componentes específicos (como resistencias, condensadores, etc.), sus características clave, tales como la frecuencia de corte y el ancho de banda, son difíciles de modificar. En contraste, los filtros digitales permiten ajustar fácilmente parámetros como la frecuencia de corte o la configuración de filtrado sin necesidad de realizar cambios físicos. Esto resulta especialmente útil cuando, tras el análisis de la señal, es necesario ajustar el rango de paso de un filtro o eliminar una frecuencia no contemplada inicialmente. Además, los filtros digitales permiten realizar modificaciones rápidas y sin intervención física, optimizando la eficiencia del proceso.

##### 2. Compensación de las Limitaciones Intrínsecas del Hardware

Los filtros físicos, aunque efectivos, no son perfectos. Su desempeño puede verse afectado por tolerancias en los componentes o por factores ambientales, como la temperatura y el envejecimiento de los mismos. Los filtros digitales, por su parte, pueden corregir o complementar las imperfecciones de los filtros de hardware. Por ejemplo, si un filtro físico no consigue eliminar completamente el ruido de 50 Hz o presenta desviaciones mínimas en las frecuencias de corte, un filtro digital puede ajustarse con precisión para corregir estas deficiencias, mejorando así la calidad general del filtrado.

##### 3. Filtrado Adaptativo en Tiempo Real

Los filtros físicos son estáticos, lo que significa que su comportamiento es constante y no varía según las condiciones de la señal. En cambio, un filtro digital ofrece la posibilidad de implementar técnicas de filtrado adaptativo, es decir, que el filtro ajuste sus características en función de las condiciones cambiantes de la señal. Esto es especialmente beneficioso en entornos donde las condiciones son variables, como cuando la interferencia de 50 Hz no está presente de manera constante. Un filtro digital adaptativo puede activarse solo cuando detecta interferencias, optimizando el procesamiento de la señal en tiempo real. De esta manera, el sistema EEG se ajusta de manera dinámica, lo que puede incrementar la fiabilidad del procesamiento.

##### 4. Procesamiento Avanzado y Filtrado de Alta Precisión

La complejidad de los filtros físicos está limitada por los componentes que los conforman. Sin embargo, en el ámbito digital es posible implementar filtros de mayor complejidad y orden, permitiendo un filtrado más preciso y específico. Los filtros digitales de mayor orden ofrecen transiciones más nítidas entre las bandas de paso y de rechazo. Por ejemplo, con un filtro digital, es posible eliminar una banda estrecha de ruido sin afectar las frecuencias adyacentes de interés, algo difícil de lograr mediante componentes físicos convencionales.

##### 5. Filtrado en Múltiples Etapas

El uso de filtrado digital permite procesar la señal en diferentes etapas, lo que resulta en un análisis más profundo y detallado de las señales EEG. Tras el paso por los filtros físicos, se pueden aplicar diferentes tipos de filtros digitales para aislar aspectos específicos de la señal que no fueron manejados por el filtrado físico. Por ejemplo, se pueden implementar filtros pasa banda específicos para resaltar ciertas bandas de ondas cerebrales, como las ondas alpha o beta, o utilizar técnicas avanzadas como el Análisis de Componentes Independientes (ICA) para separar señales superpuestas o interferencias no deseadas. El uso de filtros adaptativos

también permite ajustar continuamente la respuesta del sistema, optimizando el rendimiento del procesamiento de señales.

## **6. Posprocesamiento y Análisis Selectivo**

El filtrado digital permite realizar un análisis detallado de las señales EEG en una etapa posterior, ya sea sobre los datos almacenados o sobre la señal adquirida en tiempo real. Este posprocesamiento es fundamental para detectar patrones específicos o eliminar artefactos que no fueron suficientemente atenuados en la etapa inicial de filtrado físico. Por ejemplo, los artefactos relacionados con el movimiento o las señales de baja frecuencia, como la respiración, pueden requerir un ajuste adicional después de la adquisición de la señal. Además, un análisis más exhaustivo en tiempo real puede detectar elementos que los filtros de hardware no lograron capturar completamente.

## **7. Redundancia y Seguridad**

Incorporar tanto filtros físicos como digitales en un sistema de procesamiento de señales EEG añade una capa extra de seguridad. Si un filtro físico falla o no cumple completamente su función, el filtro digital actúa como un sistema de respaldo, asegurando que el ruido y las interferencias no afecten el análisis de la señal. Esta redundancia es clave en entornos críticos donde se necesita garantizar la integridad y pureza de los datos EEG para obtener resultados confiables y precisos.

### **4.10. Gel conductor**

El gel conductor neutro empleado en los electroencefalogramas (EEG) es una sustancia formulada para optimizar la conductividad eléctrica entre los electrodos y el cuero cabelludo del paciente. Su principal objetivo es minimizar la resistencia entre los electrodos y la piel, garantizando una transmisión eficiente y sin interferencias de las señales eléctricas cerebrales hacia los electrodos.

#### **4.10.1. Cómo aplicar el gel conductor**

En los electrodos de copa dorada, el procedimiento consiste únicamente en rellenar la cavidad de la copa con el gel conductor, posteriormente colocarlo sobre una zona del cuero cabelludo previamente limpiada con algodón, mantenerlo en posición durante unos segundos y eliminar cualquier exceso de gel, en caso de haberlo, con un pequeño trozo de papel. Utilizamos este gel en el proyecto con el fin de obtener la mejor calidad posible en las señales, además de generar un leve efecto de succión que contribuye a mantener los electrodos firmemente adheridos.

### **4.11. Jaula de Faraday**

Una Jaula de Faraday es una estructura cerrada o parcialmente cerrada hecha de un material conductor (como metal), diseñada para bloquear campos electromagnéticos externos. El principio detrás de la jaula de Faraday fue descubierto por el físico Michael Faraday en 1836, quien observó que una caja hecha de material conductor podía proteger su interior de influencias electromagnéticas.

Cuando un campo electromagnético externo impacta en la superficie de la jaula, las cargas eléctricas libres en el material conductor se redistribuyen, anulando el campo dentro de la estructura. Como resultado, cualquier dispositivo o sistema ubicado dentro de la jaula está protegido de la interferencia electromagnética (EMI) o de campos eléctricos externos.

#### **4.11.1. Principio de Funcionamiento**

El funcionamiento de la jaula de Faraday se basa en la redistribución de cargas eléctricas. Cuando una onda electromagnética incide sobre la superficie de la jaula las cargas en el material conductor de la jaula se reacomodan, generando un campo eléctrico que contrarresta el campo incidente. Este campo interno es tal que cancela el campo electromagnético externo dentro de la jaula. Como

consecuencia, el campo eléctrico dentro de la jaula es nulo, protegiendo su interior de interferencias externas.

Es importante mencionar que, aunque la jaula de Faraday bloquea la mayoría de las frecuencias electromagnéticas, la eficacia del bloqueo depende del tamaño de las aperturas en la jaula y la frecuencia de la señal. Cuanto más pequeña sea la apertura respecto a la longitud de onda de la señal externa, mayor será la efectividad de la protección.

#### 4.11.2. ¿Para que sirve?

El propósito principal de una jaula de Faraday es proteger su contenido contra interferencias electromagnéticas (EMI) y campos eléctricos indeseados. Esto es particularmente importante en ambientes donde la estabilidad de las señales eléctricas o electromagnéticas es crucial. Algunas de sus aplicaciones comunes incluyen:

- **Protección de Dispositivos Electrónicos:** Los dispositivos sensibles pueden verse afectados por interferencias electromagnéticas que podrían alterar su funcionamiento. La jaula de Faraday protege estos dispositivos, asegurando su funcionamiento adecuado.
- **Experimentos Científicos:** En laboratorios, se utilizan jaulas de Faraday para eliminar interferencias en experimentos sensibles a campos eléctricos o electromagnéticos.
- **Sistemas de Comunicaciones:** Para evitar que las señales de radiofrecuencia interfieran en comunicaciones o dispositivos electrónicos, se emplean jaulas de Faraday en recintos y cámaras anecoicas.
- **Protección Contra Ataques Electromagnéticos (EMP):** En algunos casos, las jaulas de Faraday se emplean para proteger equipos contra pulsos electromagnéticos generados de manera natural (por tormentas solares) o artificial (por explosiones nucleares o dispositivos EMP).

#### 4.11.3. Como se utiliza

El diseño y construcción de una jaula de Faraday varía dependiendo del propósito específico, pero los principios generales de uso son los siguientes:

- **Materiales Conductores:** La jaula debe estar hecha de un material conductor, generalmente aluminio, cobre o acero. Cuanto mejor sea el conductor, más efectiva será la jaula para bloquear campos electromagnéticos.
- **Aperturas Pequeñas:** Si la jaula tiene ventanas o aperturas, estas deben ser pequeñas en comparación con la longitud de onda de las señales que se desea bloquear. Por ejemplo, las mallas metálicas con agujeros finos son efectivas para bloquear frecuencias de radio.
- **Aislamiento Completo:** Para garantizar que el campo electromagnético externo no penetre en el interior, la jaula debe estar cerrada por completo o con muy pequeñas aperturas.
- **Conexión a Tierra:** En algunos casos, se recomienda conectar la jaula a tierra para proporcionar un camino seguro para la disipación de cualquier carga acumulada.

#### 4.11.4. Aplicación en la silla

En nuestro proyecto, la jaula de Faraday juega un papel crucial para asegurar que las señales cerebrales recogidas por los electrodos EEG no sean afectadas por interferencias electromagnéticas externas. Las señales EEG son extremadamente débiles, y cualquier interferencia externa podría alterar su interpretación, lo que afectaría directamente el control de la silla.

Usamos la jaula de Faraday para:

- **Protección de Señales EEG:** Las señales EEG tienen una amplitud muy baja, generalmente en el rango de microvoltios ( $\mu V$ ), lo que las hace particularmente vulnerables a interferencias de otros dispositivos electrónicos o radiaciones electromagnéticas del ambiente (como WiFi, Bluetooth, teléfonos móviles, etc.). Sin protección, estas interferencias pueden introducir ruido en las señales, lo que dificultaría o impediría que los algoritmos de procesamiento interpreten correctamente las órdenes cerebrales para controlar el movimiento de la silla.
- **Estabilidad en el Procesamiento de Datos:** El uso de una jaula de Faraday en las proximidades del sistema de adquisición de señales EEG garantizará que las mediciones sean precisas y libres de ruidos, mejorando la eficiencia de los filtros y algoritmos utilizados para procesar las señales. Esto también permitirá que el sistema de IA reconozca patrones cerebrales con mayor fiabilidad.
- **Reducción de Interferencias Ambientales:** En entornos donde hay múltiples dispositivos electrónicos, como laboratorios, hospitales, o entornos industriales, las señales electromagnéticas pueden variar ampliamente. Al utilizar una jaula de Faraday, reducimos el impacto de estas fuentes de ruido externo en las señales EEG, lo que resulta en una mayor precisión en el control de la silla de ruedas.
- **Seguridad de los Datos:** Aunque no es una prioridad principal, la jaula de Faraday también puede actuar como un mecanismo de protección adicional para evitar que las señales de datos sean interceptadas o alteradas por fuentes externas maliciosas o no deseadas.

#### 4.11.5. Apantallamiento Eléctrico

El apantallamiento eléctrico, también conocido como blindaje eléctrico, es un fenómeno que ocurre cuando un campo eléctrico externo es bloqueado o atenuado por un material conductor que lo rodea. Este principio se basa en la distribución de cargas eléctricas en un material conductor, que actúa como una barrera para evitar que el campo eléctrico externo afecte el espacio protegido.

El concepto de apantallamiento eléctrico está estrechamente relacionado con el funcionamiento de la jaula de Faraday. Cuando un conductor rodea un área en presencia de un campo eléctrico, las cargas en el conductor se redistribuyen de tal manera que cancelan el campo eléctrico dentro del área apantallada.

#### 4.12. ¿Qué es un circuito de Offset?

Un circuito de offset se utiliza para agregar un voltaje constante a una señal, desplazando su nivel de referencia sin cambiar la forma de la señal. Este tipo de circuito es común en sistemas de procesamiento de señales, donde se requiere ajustar el punto de referencia de una señal para que coincida con el rango de entrada de otro dispositivo o componente.

**Ejemplo de Funcionamiento:** Una señal de entrada, como una señal de corriente alterna (AC), puede presentar un rango de voltaje que oscila entre valores positivos y negativos, por ejemplo, una onda senoidal entre -5V y 5V.

Si se desea desplazar esta señal para que su rango esté comprendido entre 0V y 10V, se debe aplicar un voltaje de offset de 5V.

El circuito de offset sumará este valor constante de 5V a la señal, desplazando el rango completo de la misma hacia valores positivos, sin alterar su forma original.

##### 4.12.1. Utilización de Offsets

En la sección de procesamiento y recepción, se implementó un circuito de offset para adaptar las señales, ya que la RP2040 Zero no es capaz de procesar señales con componentes negativos. Este ajuste permite una correcta recepción y, en consecuencia, un procesamiento completo de las señales.

Las señales no irán más allá de los 9 Volts y -9 Volts, ya que este es el límite de la alimentación. La amplificación está calculada para que las señales normales cerebrales den un máximo de 3 Volts de amplitud. Por lo tanto, se utilizarán Zeners de 3,3 Volts en la entrada de la RP2040 Zero para evitar que le llegue una sobretensión a un pin de ADC. De esta forma, para cada pin de ADC se le inyecta un offset de 1,65 Volts para que, de esta forma, la lectura sea lo más grande y preciso posible.

#### 4.13. Sistema de emergencia

La silla de ruedas está equipada con un sistema de emergencia basado en tecnología de ultrasonido, que se encuentra instalado en las direcciones principales del dispositivo. Este sistema tiene como propósito prevenir colisiones con objetos o cualquier tipo de obstáculo, protegiendo tanto al usuario como a la integridad de la silla. Cuando los sensores de ultrasonido detectan la presencia de un obstáculo cercano, se activa un indicador visual mediante un LED en la dirección del objeto, acompañado de una señal acústica generada por un buzzer. Simultáneamente, el sistema detiene el movimiento de la silla de ruedas de manera automática y procede a un tiempo de espera de unos segundos para evaluar si es seguro continuar el desplazamiento.

#### 4.14. Tipos de alimentaciones

El proyecto opera con diferentes niveles de tensión de alimentación, siendo las principales fuentes de energía una batería de 24V y unas baterías recargables de 9V. La Raspberry Pi 4 requiere una alimentación de 5V, por lo cual se emplea un circuito regulador de tensión compuesto por un L7815 para reducir la tensión a 15V y, posteriormente, un L7805 para regularla a 5V.

El sistema de filtrado del EEG y la placa del sistema de Offset se alimentan mediante baterías recargables de 9V. En cuanto a la tarjeta RP2040 Zero, utilizada en el sistema Offset, esta recibe energía a través de un regulador de 5V derivado de las baterías de 9V. Los dispositivos del sistema de emergencia, como el buzzer y los sensores ultrasónicos, se alimentan a 5V a través de un pin de la Raspberry Pi 4 y del resto del sistema de control. Finalmente, los LED del sistema de emergencia son alimentados mediante los pines GPIO, los cuales proporcionan una salida de 3,3V.

#### 4.15. Sistema de control

El sistema de control diseñado para este proyecto está compuesto por una Raspberry Pi 4, que tiene la responsabilidad de procesar las señales provenientes de los electrodos del electroencefalograma (EEG). A partir de este procesamiento, la Raspberry Pi 4 será capaz de interpretar las intenciones del usuario y, con base en ellas, determinar los movimientos correspondientes de la silla de ruedas. Este sistema se encuentra altamente integrado con la arquitectura de control, garantizando una respuesta precisa y fluida a las señales cerebrales del usuario.

Para la regulación del movimiento de los motores, se emplea modulación por ancho de pulso (PWM, por sus siglas en inglés), operando a una frecuencia de 50 Hz. La modulación por ancho de pulso es una técnica eficiente que permite controlar la velocidad y dirección de los motores mediante la variación del ciclo de trabajo. En este caso, cuando uno de los canales de PWM presenta un ciclo de trabajo del 25 %, mientras que el otro está en 0 %, se provoca el movimiento del motor en una dirección específica. Por otro lado, cuando ambos canales de PWM se encuentran en un ciclo de trabajo del 0 %, el motor permanece detenido, lo cual corresponde a una condición de reposo.

Es crucial tener en cuenta la existencia de un estado prohibido dentro de este esquema de control. Si ambos canales de PWM se configuran simultáneamente con un ciclo de trabajo del 25 %, se generaría una condición de cortocircuito, que podría dañar el sistema. El cortocircuito ocurre cuando las señales de los canales se anulan entre sí, generando una sobrecarga en los componentes, lo cual puede comprometer la integridad tanto de los motores como de los circuitos de control. Este escenario es considerado una condición crítica, y para evitar cualquier eventualidad peligrosa, se implementa un sistema de emergencias que actúa como una salvaguardia para el correcto funcionamiento del sistema.

El sistema de emergencias tiene la capacidad de detener el movimiento de los motores en el caso de que un obstáculo sea detectado en la dirección hacia la cual el usuario intenta moverse. Cuando se identifica tal situación, los motores se detienen inmediatamente, un buzzer emitirá un sonido de advertencia, y el LED correspondiente a la dirección obstruida se encenderá, proporcionando una señal visual clara de la situación. Este sistema no solo previene accidentes, sino que también mejora la seguridad del usuario al evitar colisiones involuntarias.

Cada una de las salidas del sistema, ya sea para la señalización PWM, los pines dedicados al buzzer, los sensores ultrasónicos o los LEDs, está directamente asociada a un GPIO (General-Purpose Input/Output) de la Raspberry Pi 4. Esta asociación permite un control preciso y eficiente de todos los componentes periféricos, asegurando que el sistema responda de manera coordinada y fiable a las señales de entrada y a las condiciones de emergencia detectadas.

#### 4.16. Puentes H

Un Puente H es un circuito utilizado para invertir el sentido de giro de un motor y para separar su etapa de potencia con la de control. Su nombre viene de la forma gráfica que tiene el circuito. Se construye con 4 interruptores, pueden ser mecánicos o transistores. En la siguiente imagen se puede ver la forma de un puente H.

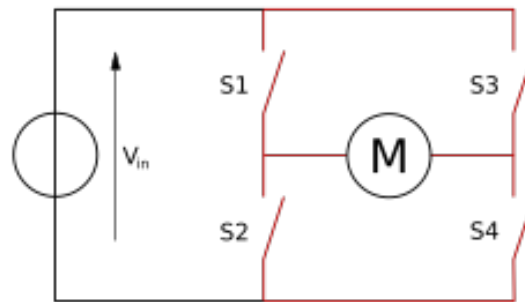


Figura 8: Puente H Genérico por medio de llaves mecánicas

En este ejemplo, cuando la llave S1 y S4 se cierran, correrá corriente por esta rama, haciendo que el motor funcione y tenga un sentido de giro. Cuando la llave S2 y S3 se cierran, correrá corriente por esta rama también, pero con sentido de giro invertido. En este circuito, S1 y S2 nunca pueden estar cerradas al mismo tiempo, porque esto generaría un corto circuito, lo mismo con las llaves S3 y S4.

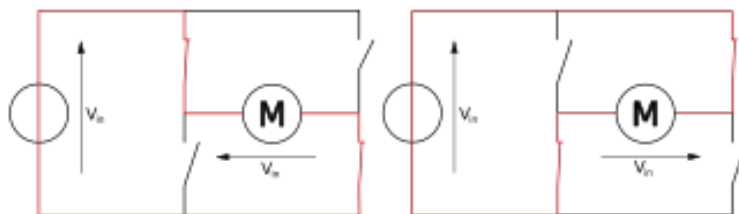


Figura 9: Puente H Operando

##### 4.16.1. Puente H de potencia

Un puente H de potencia es un tipo de circuito en el que la etapa de potencia, que incluye el motor, los transistores y está conectada a altas tensiones y corrientes, se encuentra separada de

la etapa de control, la cual generalmente opera con voltajes más bajos. Esta separación se realiza para proteger la etapa de control de las altas tensiones, que podrían dañar o quemar sus circuitos. Podemos utilizar nuestro puente H como ejemplo para ilustrar este concepto:

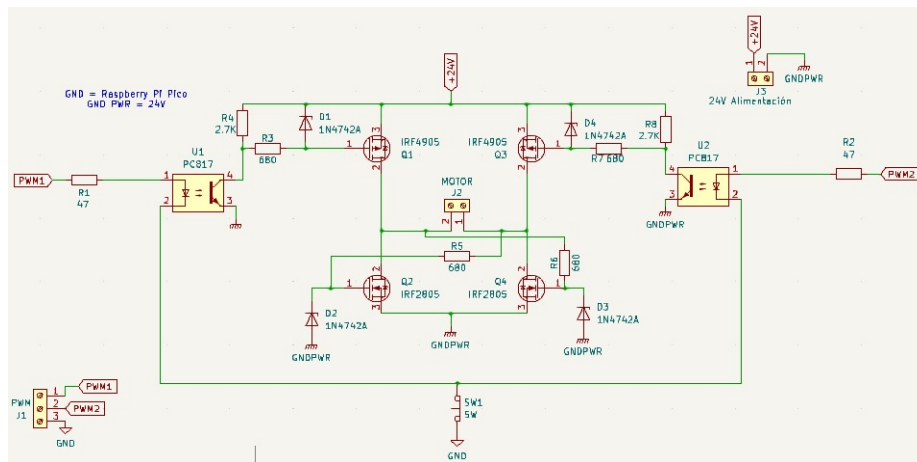


Figura 10: Nuestro Puente H

En nuestro puente H, hemos separado las ramas del motor de las de control, ya que el motor opera con 24V DC y tiene una corriente eficaz de 3,5A. Para controlar el motor de manera eficiente, hemos optado por un diseño que incluye cuatro transistores MOSFET de potencia, los cuales pueden soportar picos de hasta 74A y 82A, garantizando que puedan manejar la corriente del motor de manera continua. Los MOSFET son IRF2805 (canal N) y IRF4905 (canal P). Además, los transistores son controlados mediante señales PWM, las cuales atraviesan un optoacoplador EL817 para minimizar el riesgo de que la corriente del motor regrese a la placa de control y cause daños. De ésta manera, podemos controlar el giro del motor de manera segura y regulada.

La lógica de este circuito se basa en que cuando un MOSFET canal P está saturado, el gate del MOSFET de canal N cruzado se activará. Cuando el MOSFET canal P no está saturado, el MOSFET cruzado canal N no se activará. El circuito cuenta con diodos Zener de 12 Volts para regular la tensión de Gate-Source en cada uno de los transistores, ya que el límite es de 20 Volts.

#### 4.17. Sistema de recepción de señales por UART

El sistema de recepción de señales implementado en este proyecto se fundamenta en la transmisión de datos en formato JSON, el cual es enviado desde el microcontrolador RP2040 Zero hacia la Raspberry Pi 4, utilizando el protocolo de comunicación UART (Universal Asynchronous Receiver-Transmitter). Este esquema de transmisión de datos asíncrono es clave para garantizar la correcta transferencia de la información entre ambos dispositivos, especialmente en sistemas donde la baja latencia y la alta velocidad son factores cruciales.

Cuando la RP2040 Zero actúa como transmisor y envía datos a través del enlace UART, se introduce un optoacoplador de alta velocidad, el modelo 6N137. Este componente tiene como función principal aislar eléctricamente las dos partes del circuito, asegurando que las señales y alimentaciones de los dispositivos conectados permanezcan separadas. Esto no solo mejora la seguridad del sistema, sino que también protege los componentes sensibles de posibles interferencias o fluctuaciones de voltaje que podrían ocasionar fallas operativas o daños permanentes. El uso de optoacopladores es una práctica común en la ingeniería de control y automatización para proteger circuitos en entornos electromagnéticamente ruidosos.

Posteriormente, la señal transmitida es procesada mediante un transistor BJT (Bipolar Junction Transistor) del modelo 2N2222, que invierte la lógica de la señal. Esta operación es fundamental para garantizar que la interpretación de los datos sea precisa en el extremo receptor, adaptando la señal a los requerimientos lógicos de la Raspberry Pi 4.



Es importante destacar la elección de estos componentes electrónicos, especialmente por la alta tasa de transmisión que maneja el sistema UART, la cual opera a una velocidad de 115200 baudios. Esta velocidad de transmisión es adecuada para aplicaciones de tiempo real, donde se requiere una comunicación rápida y eficiente, sin comprometer la integridad de los datos transmitidos. El uso de componentes rápidos y confiables, como el optoacoplador 6N137 y el transistor 2N2222, asegura que el sistema pueda operar de manera estable y eficiente bajo estas condiciones.

#### 4.18. Control por PWM

La modulación por ancho de pulso (PWM, por sus siglas en inglés Pulse Width Modulation) es una técnica empleada para regular la cantidad de energía suministrada a una carga, como un motor o un LED, mediante la variación del ciclo de trabajo de una señal pulsante. En lugar de modificar el voltaje de forma continua, PWM ajusta el tiempo durante el cual la señal permanece en estado alto (encendida) y bajo (apagada) dentro de un período fijo. Esto depende de los siguientes parámetros:

- **Frecuencia:** Se refiere a la cantidad de ciclos de encendido y apagado que ocurren en un segundo, y se mide en Hertz (Hz).
- **Ciclo de trabajo:** Es el porcentaje de tiempo que la señal se mantiene en estado alto durante un ciclo. Por ejemplo: Un ciclo de trabajo del 0 % implica que la señal está siempre apagada. Un ciclo de trabajo del 50 % significa que la señal está encendida la mitad del tiempo y apagada la otra mitad. Un ciclo de trabajo del 100 % indica que la señal permanece siempre encendida. En nuestro proyecto, aplicamos la técnica de PWM en el control de los motores. Dado que utilizamos dos motores, si estos operaran a su máxima potencia, la silla de ruedas se desplazaría a una velocidad excesiva. Por esta razón, empleamos un ciclo de trabajo del 25 % para limitar la velocidad de avance.

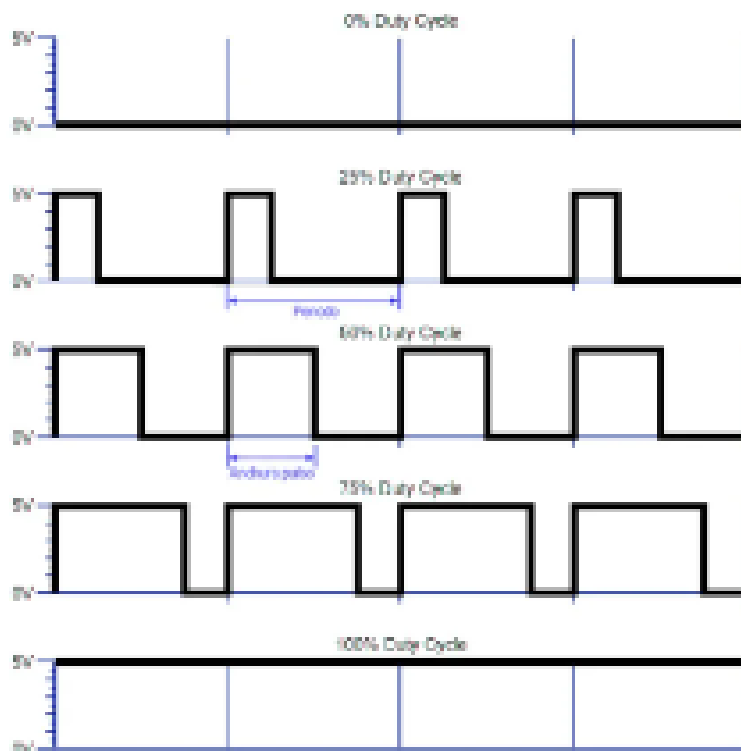


Figura 11: Ejemplo de señal PWM

## 4.19. Estructura de la silla

La estructura de la silla de ruedas está compuesta por varios elementos fundamentales que aseguran su funcionamiento eficiente y seguro. En primer lugar, se encuentra un soporte robusto, cuyo propósito principal es sostener y estabilizar los motores que impulsan la silla. Este soporte ha sido diseñado para garantizar que los motores permanezcan firmemente sujetos, evitando vibraciones o movimientos indeseados que pudieran comprometer la precisión y fiabilidad del sistema.

El sistema de propulsión de la silla está constituido por dos motores de corriente continua (CC), cada uno operando a 24 voltios. Estos motores ofrecen una potencia promedio de 200W y alcanzan una potencia máxima de 500W. Este rango de potencia permite que la silla se desplace con suficiente fuerza para adaptarse a diversas condiciones de uso, ya sea en terrenos planos o en situaciones que requieran mayor esfuerzo.

En cuanto al asiento de la silla, ha sido confeccionado específicamente para este proyecto, integrando un diseño ergonómico que prioriza la comodidad del usuario sin sacrificar la funcionalidad. En la parte posterior del respaldo de la silla se encuentra una caja diseñada para actuar como una "Jaula de Faraday". Esta caja tiene la función de aislar eléctricamente los componentes electrónicos ubicados en su interior, protegiéndolos de interferencias electromagnéticas externas.

## 5. Desarrollo Técnico

En este capítulo se proporcionará una descripción detallada sobre el funcionamiento de los distintos circuitos que conforman el dispositivo, así como su interacción en conjunto para cumplir con los objetivos del proyecto. Se explicará de manera clara y concisa cómo se utiliza el dispositivo, incluyendo instrucciones específicas para su correcto manejo y operatividad.

Además, se incluirán datos técnicos relevantes, tales como especificaciones de los componentes, características del diseño, y cualquier otro detalle que sea esencial para la comprensión del sistema. También se abordarán aspectos importantes acerca de las partes constitutivas del dispositivo, su función dentro del esquema general, y las consideraciones necesarias para su mantenimiento y optimización.

### 5.1. Descripción del funcionamiento

El usuario se sienta en la silla de ruedas y se coloca el dispositivo EEG. El EEG capta las señales cerebrales (alfa, beta, gamma, delta y theta) y las somete a un proceso de filtrado por hardware que incluye un filtro notch de 50 Hz, un filtro pasa-bajo de 100 Hz, un filtro pasa-alto de 0,5 Hz y un segundo filtro notch de 50 Hz. Posteriormente, las señales pasan por un circuito de compensación de offset para poder realizar la lectura por parte de la RP2040 Zero ya que la misma no puede leer valores negativos. Esta unidad se encarga de leer los datos mediante cuatro canales de ADC y enviarlos a la Raspberry Pi 4 por protocolo UART, donde se aloja el programa principal para el funcionamiento de la silla de ruedas.

Entre estos códigos se encuentran el control de motores, los filtros digitales, el procesamiento de señales y la inteligencia artificial (IA). Una vez las señales llegan a la Raspberry Pi 4, se filtran digitalmente para eliminar el ruido y mejorar la calidad de los datos. Tras el filtrado, las señales se envían a la IA, que tiene la función de reconocer los patrones cerebrales y determinar la dirección en la que el usuario desea moverse. Una vez el patrón ha sido identificado, se envía una señal a los motores para dirigir la silla hacia la dirección deseada (adelante, atrás, izquierda o derecha).

Además, el proyecto incluye un sistema de emergencia diseñado para garantizar la seguridad del usuario. Este sistema se activa si se detecta un error en la lectura de las señales o si hay un obstáculo en la trayectoria deseada. Su objetivo es prevenir accidentes y aumentar la seguridad del usuario al utilizar la silla de ruedas.

El sistema de emergencia utiliza sensores de ultrasonido para detectar la presencia de objetos a menos de 20 cm de la silla. En caso de detección, el sistema frena los motores y activa un LED en la dirección del objeto, además de un buzzer que emite una señal sonora durante un breve período para alertar al usuario sobre la activación del sistema.

## 5.2. Funcionamiento del EEG

El sistema de control de nuestra silla de ruedas se basa en el reconocimiento de señales cerebrales, lo que requiere un dispositivo especializado para captarlas: el electroencefalograma (EEG). Hemos desarrollado un EEG en forma de una cofia equipada con electrodos, los cuales se colocan siguiendo el protocolo conocido como “Sistema 10-20”. Este sistema estándar de ubicación define puntos específicos en el cuero cabelludo donde se deben colocar los electrodos para obtener lecturas óptimas de la actividad cerebral.

Los electrodos actúan como sensores pasivos que detectan las ondas cerebrales producidas por la actividad eléctrica del cerebro. Estas ondas se manifiestan como pequeñas fluctuaciones en el voltaje, que son captadas por los electrodos cuando se colocan sobre la superficie del cuero cabelludo del usuario. Es importante destacar que estos electrodos no penetran en el cráneo ni interactúan directamente con las neuronas. En cambio, registran los potenciales eléctricos generados por la actividad cerebral subyacente que se propagan hasta la superficie del cuero cabelludo.

Debido a que las señales captadas por los electrodos son de muy baja amplitud, requieren ser amplificadas antes de su procesamiento. Este paso no solo amplifica la señal cerebral, sino también el ruido presente en el entorno o generado por el propio cuerpo. Para abordar este problema, se realiza un proceso de filtrado que elimina el ruido y permite aislar las frecuencias relevantes de las ondas cerebrales. Para ello, utilizamos los AD620AN que son amplificadores de instrumento muy sensibles.

Las frecuencias de las ondas cerebrales detectadas son las siguientes:

- Ondas alfa: 8-12 Hz
- Ondas beta: 12-30 Hz
- Ondas gamma: 30-100 Hz
- Ondas delta: 0,5-4 Hz
- Ondas theta: 4-8 Hz

Estas ondas cerebrales están asociadas con diferentes estados mentales y actividades cerebrales, por ejemplo, las ondas alfa se observan comúnmente en estados de relajación, mientras que las ondas beta se relacionan con un estado de alerta y concentración. Las ondas gamma, por otro lado, están vinculadas a procesos cognitivos de mayor complejidad, como la percepción y la conciencia. Las ondas delta y theta se asocian con fases profundas del sueño y la meditación.

## 5.3. Funcionamiento de los filtros HardWare

En un sistema de EEG, después de amplificar las señales captadas por los electrodos, se utilizan varios filtros para limpiar y mejorar la calidad de las señales antes de procesarlas o analizarlas. Estos filtros tienen un propósito importante: eliminar el ruido y enfocarse en las señales cerebrales relevantes.

### 5.3.1. Filtro Notch de 50 Hz

- **Propósito:** Atenuación del ruido inducido por la red eléctrica de 50 Hz, común en muchos sistemas de distribución de energía.
- **Funcionamiento:** El filtro Notch es un filtro de rechazo de banda estrecha, diseñado para eliminar una frecuencia específica (en este caso, 50 Hz) sin afectar significativamente las componentes de frecuencia cercanas o las señales EEG de interés. Las señales de EEG, que suelen estar en el rango de 0,5 a 40 Hz, pueden contaminarse por la interferencia de la red, por lo que este filtro resulta esencial para su eliminación sin alterar la información relevante.

### 5.3.2. Filtro pasa bajos de 100 Hz

- **Propósito:** Reducción de las componentes de alta frecuencia que no son relevantes para el análisis EEG.
- **Funcionamiento:** Este filtro pasa bajos permite el paso de frecuencias inferiores a 100 Hz, atenuando aquellas que exceden este umbral. Las señales cerebrales de interés generalmente se sitúan por debajo de los 40 Hz, aunque algunas pueden extenderse hasta 100 Hz. Este filtro elimina frecuencias más altas, que usualmente corresponden a interferencias electromagnéticas, ruido de dispositivos electrónicos o artefactos musculares (EMG).

### 5.3.3. Filtro pasa altos de 0,5 Hz

- **Propósito:** Eliminación de las componentes de muy baja frecuencia, incluyendo artefactos de corriente continua (DC) y ruido de baja frecuencia no relacionado con la actividad cerebral.
- **Funcionamiento:** El filtro pasa altos atenúa las señales con frecuencias por debajo de 0,5 Hz, permitiendo únicamente el paso de frecuencias superiores. La señal EEG relevante generalmente comienza en torno a los 0,5 Hz, mientras que frecuencias menores suelen estar asociadas a artefactos como el desplazamiento de los electrodos, fluctuaciones en la impedancia de los mismos, o interferencias lentas que no contienen información útil sobre la actividad cerebral.

## 5.4. Funcionamiento de los filtros digitales

Una vez que las ondas cerebrales son captadas y sometidas a un proceso inicial de filtrado, son transferidas a una Raspberry Pi 4. En este dispositivo se ejecuta un programa, desarrollado íntegramente por nuestro equipo, que continúa con el procesamiento y refinamiento de la señal. El motivo por el cual se aplica un filtrado adicional es que, luego de la etapa de filtrado inicial, la señal pasa por un segundo amplificador. Este amplificador, además de amplificar las señales cerebrales de interés, también amplifica el ruido residual presente en las mismas. Para contrarrestar este efecto, empleamos filtros digitales implementados por software, los cuales ofrecen una mayor precisión y control en la eliminación de interferencias no deseadas.

Dentro del programa, se han implementado los siguientes filtros:

- **Filtro Notch de 50 Hz:** Este filtro está diseñado para eliminar las interferencias generadas por la red eléctrica, las cuales operan a una frecuencia de 50 Hz. Dado que estas interferencias pueden superponerse con la señal cerebral y distorsionar su interpretación, su eliminación es esencial para obtener un registro más fiel de la actividad cerebral.
- **Filtro Pasa-Bajo de 100 Hz:** El objetivo de este filtro es retener las frecuencias cerebrales más relevantes, es decir, aquellas que se encuentran por debajo de los 100 Hz, mientras que elimina el ruido de alta frecuencia, que suele ser resultado de interferencias externas o artefactos no relacionados con la actividad cerebral. Este rango incluye frecuencias relacionadas con ritmos cerebrales importantes como las ondas alfa, beta, y gamma.
- **Filtro Pasa-Alto de 0,5 Hz:** Este filtro se encarga de remover las señales de baja frecuencia que no aportan información útil sobre la actividad cerebral, como los artefactos causados por movimientos lentos o variaciones de corriente. Solo se permite el paso de las frecuencias EEG relevantes, garantizando que las señales reflejen con mayor precisión la actividad neural de interés.

Es importante señalar que el uso de estos filtros garantiza que la señal procesada esté lo más limpia posible de interferencias y artefactos, mejorando la precisión del sistema para detectar y procesar patrones cerebrales. La implementación de filtros mediante software, además, permite ajustar dinámicamente las configuraciones según las necesidades específicas del sistema y del entorno operativo, optimizando así el rendimiento de la silla de ruedas controlada por señales cerebrales.

## 5.5. Funcionamiento del Puente H

El circuito utiliza dos señales PWM (PWM1 y PWM2) para controlar los transistores MOSFET en configuración de puente H, lo que permite invertir la polaridad del voltaje aplicado al motor, controlando así la dirección de rotación.

La velocidad del motor puede controlarse ajustando el ciclo de trabajo (duty cycle) de las señales PWM, que modulan la cantidad de energía entregada al motor.

Este circuito es ideal para aplicaciones donde se requiere un control preciso de dirección y velocidad de motores DC.

### 5.5.1. Descripción de los Componentes

1. **U1 y U2 (PC817 - Optoacopladores):** Aíslan eléctricamente el circuito de control (de baja tensión) del circuito de potencia (de alta tensión). Esto protege al microcontrolador de posibles picos de voltaje y ruidos eléctricos provenientes del circuito de potencia. Cada optoacoplador tiene un LED interno que, al ser activado por una señal PWM, enciende un fototransistor interno, permitiendo que la señal de control pase al lado de potencia.
2. **Q1 y Q3 (IRF4905 - MOSFET canal P):** Actúan como interruptores en la parte superior del puente H. Controlan la alimentación positiva hacia el motor. Estos MOSFET se activan cuando la señal PWM1 o PWM2 está en bajo (0V) y se desactivan cuando está en alto (señal de 5V), debido a la naturaleza de los MOSFET de canal P.
3. **Q2 y Q4 (IRF2805 - MOSFET canal N):** Actúan como interruptores en la parte inferior del puente H. Conectan el motor a tierra cuando están activados. Estos MOSFET se activan cuando la señal PWM1 o PWM2 está en alto (5V) y se desactivan cuando está en bajo (0V).
4. **R1 y R2 (Resistencias de 180 ohms):** Limitan la corriente de entrada hacia los LEDs internos de los optoacopladores (U1 y U2) para protegerlos de daños.
5. **R3 y R4 (Resistencias de 1k ohms):** Están conectadas en serie con los MOSFET de canal P (Q1 y Q3). Limitan la corriente de la compuerta y aseguran un apagado rápido del MOSFET al detenerse la señal.
6. **J1 y J2 (Conectores de entrada PWM):** Reciben las señales PWM de control provenientes de un microcontrolador o una fuente de señal PWM.
7. **J3 y J4 (Conectores de alimentación):** Proveen la alimentación al circuito, uno con 24V para el motor y otro con 15V para los circuitos de control.
8. **SW1 (Interruptor):** utilizado para habilitar o deshabilitar el circuito completo, conectando o desconectando GND.

## 5.6. Funcionamiento del Sistema de Emergencia

El sistema de emergencia de nuestra silla de ruedas controlada mediante señales cerebrales es un componente crítico para la seguridad del usuario, diseñado específicamente para evitar colisiones y accidentes en caso de que el usuario intente desplazarse hacia una dirección donde exista un obstáculo. Este sistema está integrado con los sensores ultrasónicos, que actúan como "ojos" del sistema, escaneando constantemente el entorno alrededor de la silla en cuatro direcciones: adelante, atrás, izquierda y derecha.

El funcionamiento es sencillo:

1. **Detección de Obstáculos:** Cada uno de los sensores ultrasónicos está ubicado estratégicamente en una de las direcciones mencionadas. Estos sensores emiten pulsos ultrasónicos que se reflejan al chocar con un objeto, lo que permite calcular la distancia al obstáculo con alta precisión. Si el sistema detecta un objeto en la dirección hacia la cual el usuario ha decidido moverse, el sensor ultrasónico correspondiente enviará una señal de alerta al sistema de control.

2. **Intervención Automática:** Una vez que se detecta un obstáculo en la trayectoria, el sistema de emergencia interviene automáticamente para detener el movimiento de los motores. Esto se logra mediante la interrupción de las señales de control enviadas por la Raspberry Pi 4 a los motores, utilizando la modulación PWM. Al cortar las señales de control de los motores, se asegura que la silla no avance hacia la dirección donde se ha identificado el peligro.
3. **Señales Visuales y Auditivas:** Al mismo tiempo que se detienen los motores, el sistema activa una señal auditiva mediante un buzzer, que emite un sonido de advertencia. Además, se enciende un LED específico que indica la dirección donde se encuentra el obstáculo, proporcionando una indicación visual clara tanto para el usuario como para quienes estén cerca. Estos elementos visuales y auditivos aseguran que la persona a cargo de la silla, o cualquier acompañante, esté consciente del riesgo potencial en la dirección de movimiento.
4. **Restablecimiento del Movimiento:** Una vez que el obstáculo es removido, o el usuario decide cambiar la dirección de movimiento hacia un área despejada, el sistema de emergencia desactiva automáticamente las señales de advertencia (buzzer y LED), y el control de los motores se restablece, permitiendo que la silla continúe su desplazamiento normal. Este proceso es completamente autónomo, sin necesidad de intervención manual.
5. **Integración con el Sistema de Control Principal:** El sistema de emergencia está completamente integrado con el sistema de control principal basado en la Raspberry Pi 4. Todos los sensores ultrasónicos, buzzer y LEDs están conectados a los GPIOs de la Raspberry Pi, permitiendo una comunicación eficiente entre los distintos componentes. Además, el sistema de emergencia está diseñado para operar en tiempo real, lo que garantiza una respuesta inmediata ante cualquier detección de peligro.

## 5.7. Explicacion de los codigos utilizados

En este apartado se explicaran los codigos utilizados en el proyecto y porqué los hemos utilizados.

### 5.7.1. Explicacion de bias.py

Este codigo es la clase organizadora de las demas clases que contienen: recepcion, filtrado, IA y movimiento de motor.

En este codigo se crea la clase principal: BiasClass.

```

1  class BiasClass:
2  def __init__(self, n, fs, channels, port, baudrate, timeout):
3      # Define properties for the class
4      self._n = n
5      self._fs = fs
6      self._number_of_channels = channels
7      self._duration = self._n / self._fs
8      self._port = port
9      self._baudrate = baudrate
10     self._timeout = timeout
11     self._commands = ["forward", "backwards", "left", "right", "stop", "rest"]
12     self._samples_trainig_command = 100

```

- **Propiedades:** Se inicializan varias propiedades para manejar parámetros de la clase, como el número de muestras `n`, la frecuencia de muestreo `fs`, el número de canales EEG, y las configuraciones del puerto serie para la recepción de datos.
- **Comandos:** Se define una lista de comandos posibles que la IA debe predecir para controlar el movimiento de la silla de ruedas.

```

1  self._biasReception = ReceptionBias(self._port, self._baudrate, self._timeout)
2  self._biasFilter = FilterBias(n=self._n, fs=self._fs, notch=True, bandpass=True, fir=False, iir=False)

```

```

3 self._biasProcessing = ProcessingBias(n=self._n, fs=self._fs)
4 self._biasGraphing = GraphingBias(graph_in_terminal=True)
5 self._biasMotor = MotorBias(...)
6 self._biasAI = AIBias(self._n, self._fs, self._number_of_channels, self._commands)

```

Aquí se crean objetos de distintas clases que se utilizarán en los métodos de la clase:

- **ReceptionBias:** Para recibir datos desde los electrodos EEG.
- **FilterBias:** Para aplicar filtros digitales (notch, bandpass).
- **ProcessingBias:** Para procesar las señales EEG.
- **GraphingBias:** Para graficar las señales en tiempo real.
- **MotorBias:** Para controlar los motores en respuesta a los comandos predichos por la IA.
- **AIBias:** Para entrenar el modelo de IA y predecir comandos basados en los datos EEG.

```

1 def train_ai_model(self, save_path, saved_dataset_path):
2     self._biasAI.collect_and_train(reception_instance=self._biasReception,
3                                   filter_instance=self._biasFilter,
4                                   processing_instance=self._biasProcessing,
5                                   samples_per_command=self._samples_trainig_command,
6                                   save_path=save_path,
7                                   saved_dataset_path=saved_dataset_path, real_data=
8                                   True)

```

Este método permite entrenar un modelo de IA con datos de señales EEG en tiempo real:

- **collect\_and\_train:** Este método recolecta datos, los filtra, procesa y luego entrena la IA para reconocer comandos a partir de señales EEG.
- **Parámetros:** save\_path y saved\_dataset\_path indican los caminos donde se guardan los datos entrenados o el conjunto de datos usado.

```

1 def app_run(self):
2     while True:
3         # Receive eeg data
4         signals = self._biasReception.get_real_data(channels=self._number_of_channels,
5             n=self._n)
6
7         # Graph signals
8         for ch, signal in signals.items():
9             t = np.arange(len(signals[ch])) / self._fs
10            self._biasGraphing.graph_signal_voltage_time(t=t, signal=np.array(signal),
11                title="Signal {}".format(ch))
12
13            # Apply digital filtering
14            filtered_data = self._biasFilter.filter_signals(signals)
15
16            # Graph filtered signals
17            t = np.linspace(0, self._duration, self._n, endpoint=False)
18            for ch, signal in filtered_data.items():
19                self._biasGraphing.graph_signal_voltage_time(t=t, signal=signal, title="
20                Filtered Signal {}".format(ch))
21
22            # Process data
23            times, eeg_signals = self._biasProcessing.process_signals(filtered_data)
24
25            # Graph processed EEG signals by band
26            for ch, signals in eeg_signals.items():
27                for band_name, sig in signals.items():
28                    self._biasGraphing.graph_signal_voltage_time(t=times[ch], signal=sig,
29                        title=f"{band_name.capitalize()} interpolated. {ch}")

```

```

27         # Plot now
28         self._biasGraphing.plot_now()
29
30         # Predict and move motors
31         #command = self._biasAI.predict_command(eeg_data=eeg_signals)
32         #self._biasMotor.move_if_possible(command)

```

- **Recepción de datos EEG:** Se utiliza `get_real_data` para obtener señales EEG en tiempo real.
- **Graficación de señales sin filtrar:** Se grafican las señales sin filtrar usando `GraphingBias`.
- **Filtrado digital:** Se aplican filtros a las señales mediante `FilterBias`.
- **Procesamiento:** Las señales filtradas son procesadas usando `ProcessingBias`.
- **Predicción de comandos (comentada):** El código está preparado para usar IA para predecir un comando basado en las señales EEG procesadas, pero esta parte está comentada.
- **Control de motores:** Según el comando predicho, se puede mover la silla de ruedas.

### 5.7.2. Explicacion bias\_ai.py

En este codigo se programa la IA. Es donde la IA esta la IA y se entrena con datasets.

```

1  def main():
2      n = 1000
3      fs = 500
4      online = True
5      number_of_channels = 4
6      port = '/dev/serial0'
7      baudrate = 115200
8      timeout = 1
9      biasReception = ReceptionBias(port, baudrate, timeout)
10     biasFilter = FilterBias(n=n, fs=fs, notch=True, bandpass=True, fir=False, iir=False)
11     biasProcessing = ProcessingBias(n=n, fs=fs)
12     commands = ["forward", "backwards", "left", "right", "stop", "rest"]
13     biasAI = AIBias(n=n, fs=fs, channels=number_of_channels, commands=commands)
14     train = input("Do you want to train model? (y/n): ")
15     if train.lower() == "y":
16         saved_dataset_path = None
17         save_path = None
18         loading_dataset = input("Do you want to load a existent dataset? (y/n): ")
19         if loading_dataset.lower() == "y":
20             saved_dataset_path = input("Write the name of the file where dataset was saved: ")
21         else:
22             save_new_dataset = input("Do you want to save the new dataset? (y/n): ")
23             if save_new_dataset == "y":
24                 save_path = input("Write the path where you want to save the dataset: ")
25
26         biasAI.collect_and_train(reception_instance=biasReception, filter_instance=biasFilter, processing_instance=biasProcessing,
27                                trials_per_command=1, save_path=save_path,
28                                saved_dataset_path=saved_dataset_path, real_data=False)
29         # Generate synthetic data
30         signals = generate_synthetic_eeg(n_samples=n, n_channels=number_of_channels, fs=fs, command="left")
31         #signals = biasReception.get_real_data(channels=number_of_channels, n=n)
32
33         filtered_data = biasFilter.filter_signals(signals)
34         # Process data
35         times, eeg_signals = biasProcessing.process_signals(filtered_data)
36         predicted_command = biasAI.predict_command(eeg_data=eeg_signals)
37         print(f"Predicted Command: {predicted_command}")

```



La función `main()` es el punto de entrada del programa y orquesta la inicialización y ejecución del sistema.

#### Flujo de la Función Principal:

1. **Configuración Inicial:** Define parámetros esenciales como el número de muestras, frecuencia de muestreo, número de canales, puerto serial y velocidad de comunicación.
2. **Inicialización de Objetos:**
  - **ReceptionBias:** Maneja la recepción de datos EEG desde el puerto serial.
  - **FilterBias:** Aplica filtros Notch y Bandpass para limpiar las señales EEG.
  - **ProcessingBias:** Procesa las señales filtradas para extraer información relevante.
  - **AIBias:** Gestiona la IA encargada de interpretar las señales y predecir comandos.
3. **Entrenamiento del Modelo de IA:** Ofrece al usuario la opción de entrenar el modelo, ya sea cargando un dataset existente o creando uno nuevo con datos reales o sintéticos.
4. **Generación de Datos Sintéticos:** Crea señales EEG simuladas para pruebas sin necesidad de hardware físico.
5. **Filtrado y Procesamiento de Señales:** Limpia y prepara las señales para la extracción de características.
6. **Predicción de Comandos:** Utiliza el modelo entrenado para interpretar las señales EEG y determinar el comando de movimiento correspondiente.

La clase `AIBias` es responsable de manejar todo el ciclo de vida del modelo de IA, desde la recopilación de datos hasta la predicción de comandos basados en las señales EEG.

```
1 class AIBias:
2     def __init__(self, n, fs, channels, commands):
3         # Inicialización de parámetros y estructuras de datos
4         self._n = n
5         self._fs = fs
6         self._number_of_channels = channels
7         self._features_length = len(["mean", "variance", "skewness", "kurt", "energy",
8                                     "band_power", "wavelet_energy", "entropy"])
9         self._number_of_waves_per_channel = len(["alpha", "beta", "gamma", "delta", "theta"])
10        self._num_features_per_channel = self._features_length * self._number_of_waves_per_channel
11        self._commands = commands
12        self._model = self.build_model(output_dimension=len(self._commands))
13        self._is_trained = False
14        self._pca = PCA(n_components=0.95) # Retiene el 95% de la varianza
15        self._scaler = StandardScaler()
16
17        # Mapeo dinámico de etiquetas basado en los comandos proporcionados
18        self._label_map = {command: idx for idx, command in enumerate(self._commands)}
19        self._reverse_label_map = {idx: command for command, idx in self._label_map.items()}
20
21        # Getter para verificar si el modelo ha sido entrenado
22        def ai_is_trained(self):
23            return self._is_trained
24
25        def collect_and_train(self, reception_instance, filter_instance,
26                             processing_instance, trials_per_command,
27                             save_path=None, saved_dataset_path=None, real_data=True):
28            """
29            Recopila datos EEG, extrae características y entrena el modelo.
```

```

30     X = []
31     y = []
32
33     if saved_dataset_path is None:
34         for trial in range(trials_per_command):
35             for command in self._commands:
36                 # Obtener datos reales o generar datos sintéticos
37                 if real_data:
38                     print(f"Think about {command}. Trial: {trial}")
39                     signals = reception_instance.get_real_data(channels=self.
_number_of_channels, n=self._n)
40                 else:
41                     print(f"Trial: {trial}")
42                     signals = generate_synthetic_eeg(n_samples=self._n,
n_channels=self._number_of_channels, fs=self._fs, command=command)
43
44                 # Filtrar y procesar las señales
45                 filtered_data = filter_instance.filter_signals(signals)
46                 _, eeg_signals = processing_instance.process_signals(
filtered_data)
47
48                 # Extraer características y añadir a X
49                 features = self.extract_features(eeg_signals)
50
51                 X.append(features)
52                 y.append(self._label_map[command])
53
54                 if real_data:
55                     time.sleep(1)
56
57                 if real_data:
58                     print("Changing command. Be ready")
59                     time.sleep(20)
60
61                 # Convertir X y y a arrays de Numpy
62                 X = np.array(X)
63                 y = np.array(y)
64
65                 if save_path:
66                     # Guardar el dataset como un archivo comprimido de Numpy
67                     np.savez_compressed(f"{save_path}.npz", X=X, y=y)
68                     print(f"Dataset saved to {save_path}.npz")
69
70                 else:
71                     # Cargar dataset existente
72                     data = np.load(f"{saved_dataset_path}.npz")
73                     X, y = data['X'], data['y']
74
75                 # Convertir y a codificación one-hot
76                 lb = LabelBinarizer()
77                 y = lb.fit_transform(y)
78
79                 # Entrenar el modelo con los datos recopilados
80                 self.train_model(X, y)
81
82     def build_model(self, output_dimension):
83         """
84         Construye y compila el modelo de red neuronal.
85         """
86         model = Sequential([
87             InputLayer(shape=(self._number_of_channels, self.
_num_features_per_channel)),
88             Conv1D(filters=64, kernel_size=3, activation='relu'),
89             MaxPooling1D(pool_size=1),
90             Dropout(0.5),
91             Flatten(),
92             Dense(100, activation='relu'),
93             Dropout(0.5),

```

```

104         Dense(50, activation='relu'),
105         Dropout(0.5),
106         Dense(output_dimension, activation='softmax') # 6 clases de salida
107     ])
108     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
109 accuracy'])
110     return model
111
112 def extract_features(self, eeg_data):
113     """
114     Extrae caracter sticas estad sticas y en el dominio de la frecuencia de las
115     se ales EEG.
116     """
117     features = []
118     # Iterar sobre cada canal en eeg_data
119     for ch, signals_per_channel in eeg_data.items():
120         channel_features = []
121         assert(len(signals_per_channel) == self._number_of_waves_per_channel)
122         # Iterar sobre las se ales de cada canal
123         for band_name, signal_wave in signals_per_channel.items():
124             signal_wave = np.array(signal_wave)
125
126             # Caracter sticas estad sticas
127             mean = np.mean(signal_wave)
128             variance = np.var(signal_wave)
129             skewness = skew(signal_wave)
130             kurt = kurtosis(signal_wave)
131             energy = np.sum(signal_wave ** 2)
132
133             # Caracter sticas en el dominio de la frecuencia (Densidad Espectral
134             de Potencia)
135             freqs, psd = welch(signal_wave, fs=self._fs) # fs = 500 Hz
136
137             # Potencia de banda
138             band_power = np.sum(psd) # Potencia total dentro de esta banda
139
140             # Energ a de wavelet utilizando CWT con la funci n morlet
141             scales = np.arange(1, 31)
142             coeffs = cwt(signal_wave, morlet, scales)
143             wavelet_energy = np.sum(coeffs ** 2)
144
145             # Entrop a
146             signal_entropy = entropy(np.histogram(signal_wave, bins=10)[0])
147
148             list_of_features = [mean, variance, skewness, kurt, energy,
149 band_power, wavelet_energy, signal_entropy]
150
151             # Aadir todas las caracter sticas juntas
152             channel_features.extend(list_of_features)
153
154             assert(len(list_of_features) == self._features_length)
155
156             features.append(channel_features)
157
158     features = np.abs(np.array(features))
159     features = self._scaler.fit_transform(features) # Normalizaci n
160     # features = self._pca.fit_transform(features) # Reducci n de
161     dimensionalidad (actualmente comentada)
162
163     # Ajustar la forma basada en el tama o real
164     num_features_per_channel = features.shape[1]
165     assert(self._num_features_per_channel == num_features_per_channel)
166     expected_shape = (self._number_of_channels, self._num_features_per_channel)
167     features = features.reshape(expected_shape)
168     return features
169
170 def train_model(self, X, y):
171     """

```

```

157     Entrena el modelo de red neuronal con los datos proporcionados.
158     """
159     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
160 random_state=42)
161     self._model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(
162 X_test, y_test))
163     self._is_trained = True
164
165 def predict_command(self, eeg_data):
166     """
167     Predice el comando de movimiento basado en las se a les EEG proporcionadas.
168     """
169     if not self._is_trained:
170         raise Exception("Model has not been trained yet.")
171
172     # Extraer caracter sticas de los datos EEG
173     features = self.extract_features(eeg_data)
174
175     # Asegurar que las caracter sticas tienen la forma correcta
176     features = features.reshape(1, self._number_of_channels, self.
177 _num_features_per_channel)
178
179     # Realizar la predicci n
180     prediction = self._model.predict(features)
181
182     # Obtener el ndice de la etiqueta predicha
183     predicted_label_index = np.argmax(prediction, axis=1)[0]
184
185     # Convertir el ndice num rico a la etiqueta de texto correspondiente
186     predicted_command = self._reverse_label_map[predicted_label_index]
187
188     return predicted_command

```

## Funciones Clave de la Clase AIBias:

### 1. Constructor (\_\_init\_\_):

- Parámetros Iniciales:
  - n: Número de muestras por canal.
  - fs: Frecuencia de muestreo.
  - channels: Número de canales EEG.
  - commands: Lista de comandos que el modelo debe reconocer.
- Configuración de features:
  - Define la longitud de las características extraídas por banda de frecuencia.
  - Calcula el número total de características por canal.
- Modelo de IA: Construye y compila un modelo de red neuronal utilizando la función build\_model.
- Preprocesamiento: Inicializa escaladores (StandardScaler) y reducción de dimensionalidad (PCA).
- Mapeo de Etiquetas: Crea mapas para convertir comandos en índices numéricos y vice-versa.

### 2. Método collect\_and\_train:

- Propósito: Recopila datos EEG (reales o sintéticos), extrae características y entrena el modelo de IA.
- Proceso:
  - Si no se proporciona una ruta de dataset existente, recopila datos para cada comando, filtra y procesa las señales, extrae características y las almacena en X y y.

- Opcionalmente guarda el dataset recopilado.
- Si se proporciona una ruta de dataset existente, carga los datos directamente.
- Convierte las etiquetas y a una codificación one-hot para el entrenamiento.
- Llama al método `train_model` para entrenar el modelo con los datos recopilados.

### 3. Método `build_model`:

- Descripción: Construye una red neuronal secuencial con las siguientes capas:
  - `InputLayer`: Define la forma de entrada basada en el número de canales y características.
  - `Conv1D`: Capa convolucional para detectar patrones en las señales.
  - `MaxPooling1D`: Reducción de dimensionalidad.
  - `Dropout`: Prevención de sobreajuste.
  - `Flatten`: Aplana las salidas para las capas densas.
  - `Dense`: Capas completamente conectadas para clasificación.
  - `Softmax`: Capa de salida para clasificación multiclase.
- Compilación del Modelo: Utiliza el optimizador `'adam'`, la función de pérdida `'categorical_crossentropy'` y métrica de precisión.

### 4. Método `extract_features`:

- Propósito: Extrae Features estadísticas y de dominio de frecuencia de las señales EEG.
- Features Extraídas:
  - Estadísticas: Media, varianza, asimetría, curtosis, energía.
  - Frecuencia: Potencia espectral de banda, energía de wavelet.
  - Entropía: Medida de la incertidumbre o aleatoriedad de la señal.
- Proceso:
  - Itera sobre cada canal y cada banda de frecuencia para calcular las Features.
  - Normaliza las Features utilizando `StandardScaler`.
  - Opcionalmente aplica PCA para reducir la dimensionalidad (actualmente comentado).
  - Reorganiza las Features en la forma esperada para el modelo.

### 5. Método `train_model`:

- Descripción: Divide los datos en conjuntos de entrenamiento y prueba, entrena el modelo y actualiza el estado de entrenamiento.
- Proceso:
  - Utiliza `train_test_split` para dividir los datos.
  - Entrena el modelo con las características y etiquetas de entrenamiento.
  - Valida el modelo con el conjunto de prueba.
  - Marca el modelo como entrenado.

### 6. Método `predict_command`:

- Descripción: Predice el comando de movimiento basado en nuevas señales EEG.

■ Proceso:

- Verifica si el modelo ha sido entrenado.
- Extrae características de las señales EEG proporcionadas.
- Ajusta la forma de las características para la predicción.
- Utiliza el modelo para predecir el comando.
- Convierte la predicción numérica a la etiqueta de comando correspondiente.

```

1  def generate_synthetic_eeg(n_samples, n_channels, fs, command=None):
2      """
3      Genera datos EEG sint ticos para m ltiples canales.
4      La salida es un diccionario donde cada canal tiene 1000 muestras crudas.
5      Simula diferentes tareas alterando los patrones de la se al.
6      """
7      t = np.linspace(0, n_samples/fs, n_samples, endpoint=False)
8      data = {}
9
10     for ch in range(n_channels):
11         # Simular diferentes bandas de frecuencia con algunas correlaciones b sicas
12         base_alpha = np.sin(2 * np.pi * 10 * t) # Alpha (10 Hz)
13         base_beta = np.sin(2 * np.pi * 20 * t) # Beta (20 Hz)
14         base_theta = np.sin(2 * np.pi * 6 * t) # Theta (6 Hz)
15         base_delta = np.sin(2 * np.pi * 2 * t) # Delta (2 Hz)
16         base_gamma = np.sin(2 * np.pi * 40 * t) # Gamma (40 Hz)
17
18         alpha_power = 1.0
19         beta_power = 1.0
20         theta_power = 1.0
21         delta_power = 1.0
22         gamma_power = 1.0 # Ajustar se al seg n el comando
23
24         if command == "forward":
25             alpha_power = 1.5
26             beta_power = 0.5
27         elif command == "backward":
28             alpha_power = 0.5
29             beta_power = 1.5
30         elif command == "left":
31             theta_power = 1.5
32             delta_power = 0.5
33         elif command == "right":
34             theta_power = 0.5
35             delta_power = 1.5
36         elif command == "stop":
37             alpha_power = 0.2
38             beta_power = 0.2
39             gamma_power = 0.2
40         else: # rest
41             alpha_power = 1.0
42             beta_power = 1.0
43             theta_power = 1.0
44             delta_power = 1.0
45             gamma_power = 1.0
46
47         # Generar se al con algo de aleatoriedad y correlaciones
48         signal = (
49             alpha_power * base_alpha +
50             beta_power * base_beta +
51             theta_power * base_theta +
52             delta_power * base_delta +
53             gamma_power * base_gamma
54         )
55
56         # A adir correlaci n entre canales (e.g., 10% de la se al del canal
57         anterior)
58         if ch > 0:

```

```

58         signal += 0.1 * data[ch-1]
59
60         # A adir ruido aleatorio para simular se ales EEG realistas
61         noise = np.random.normal(0, 0.1, size=t.shape)
62         signal += noise
63
64         # Almacenar la se al cruda en el diccionario
65         data[ch] = signal
66
67     return data

```

Características de la Función:

- **Simulación de Bandas de Frecuencia:** Crea señales para diferentes bandas (Alpha, Beta, Theta, Delta, Gamma) ajustando sus amplitudes según el comando simulado.
- **Correlación entre Canales:** Introduce una correlación simple entre canales adyacentes para mayor realismo.
- **Ruido Aleatorio:** Añade ruido gaussiano para emular las imperfecciones de las señales EEG reales.
- **Flexibilidad de Comandos:** Permite ajustar las características de las señales según el comando que se está simulando (e.g., "forward", "left").

### 5.7.3. Explicacion bias\_dsp.py

Este código implementa un flujo completo de recepción, filtrado y procesamiento de señales EEG utilizando diversas técnicas de procesamiento digital de señales (DSP). A continuación, explicaré cada parte clave del código:

```

1     class DSPBias:
2     def __init__(self, n, fs):
3         self._n = n
4         self._fs = fs
5         self._duration = self._n / self._fs

```

Esta es una clase base que inicializa las propiedades comunes para las operaciones de DSP, como la cantidad de muestras (n), la frecuencia de muestreo (fs), y la duración de la señal. Es utilizada como punto de partida para clases que heredan estas características.

```

1     class ProcessingBias(DSPBias):
2     def __init__(self, n, fs):
3         super().__init__(n, fs)
4         self._biasGraphing = GraphingBias(graph_in_terminal=False)

```

Esta clase hereda de DSPBias y añade la funcionalidad de procesamiento de señales EEG. En el método process\_signals, se procesa cada señal en los canales EEG y se grafican en el dominio del tiempo y frecuencia.

```

1     def do_fft(self, signal):
2         signal_fft = np.fft.fft(signal)
3         frequencies = np.fft.fftfreq(self._n, d=1/self._fs)
4         signal_fft_magnitude = np.abs(signal_fft) / self._n
5         return signal_fft, frequencies, signal_fft_magnitude

```

Este método realiza la Transformada de Fourier para convertir la señal del dominio del tiempo al dominio de la frecuencia, permitiendo analizar los componentes de frecuencia de la señal EEG.

```

1     class FilterBias(DSPBias):
2     def __init__(self, n, fs, notch, bandpass, fir, iir):
3         self._notch = notch
4         self._bandpass = bandpass
5         self._fir = fir
6         self._iir = iir
7         super().__init__(n=n, fs=fs)

```

Esta clase implementa varios tipos de filtros digitales, como:

- **Notch Filter:** Para eliminar interferencias de frecuencia específicas (por ejemplo, ruido de la línea de energía a 50 Hz).
- **Bandpass Filter:** Para filtrar un rango de frecuencias.
- **FIR e IIR Filters:** Filtros de paso bajo que permiten suavizar las señales.

```
1 def butter_bandpass_filter(self, data, lowcut, highcut, order=5):
2     nyquist = 0.5 * self._fs
3     low = lowcut / nyquist
4     high = highcut / nyquist
5     b, a = butter(order, [low, high], btype='band')
6     y = filtfilt(b, a, data, axis=1)
7     return y
```

Este método aplica un filtro de paso de banda usando un filtro Butterworth. Este tipo de filtro permite el paso de un rango específico de frecuencias, bloqueando las demás.

Dentro de main, después de recibir las señales del hardware (ReceptionBias), las señales son graficadas antes y después del filtrado. Luego, las señales filtradas son procesadas usando Fourier y graficadas en diferentes bandas de EEG.

```
1 for ch, signal in filtered_data.items():
2     biasGraphing.graph_signal_voltage_time(t=t, signal=signal, title="Filtered Signal
3     {}".format(ch))
```

Aquí se grafican las señales filtradas para observar cómo se ven en el dominio del tiempo, permitiendo una inspección visual de los efectos del filtrado.

```
1 def interpolate_signal(self, t, signal, new_t):
2     interpolated_signal = scipy.interpolate.interp1d(t, signal, kind='cubic')(new_t)
3     return interpolated_signal
```

Este método interpola las señales para mejorar la resolución temporal, lo cual es útil cuando se desea obtener más puntos de datos entre las muestras originales.

#### 5.7.4. Explicacion bias\_graphing.py

En este código se realiza los gráficos de las señales en función del tiempo y la frecuencia. En el contexto de nuestro proyecto, la clase "GraphingBias" se utiliza para visualizar las señales cerebrales captadas por el sistema EEG de nuestra silla de ruedas. Estas señales se pueden representar tanto en el dominio del tiempo como en el de la frecuencia, lo que es crucial para entender la actividad cerebral y ajustar los patrones de control de la silla.

```
1 class GraphingBias:
2     # Constructor
3     def __init__(self, graph_in_terminal):
4         self._graph_in_terminal = graph_in_terminal
```

El constructor inicializa la clase GraphingBias, recibiendo como parámetro graph\_in\_terminal, que es un valor booleano. Este parámetro determina si los gráficos se deben visualizar en la terminal (True) o en una ventana gráfica externa usando matplotlib (False).

```
1 def graph_signal_voltage_time(self, t, signal, title):
2     if not self._graph_in_terminal:
3         plt.figure(figsize=(12, 6))
4         if signal.ndim == 1:
5             plt.plot(t, signal)
6         else:
7             for i in range(signal.shape[0]):
8                 plt.plot(t, signal[i])
9         plt.title(title)
10        plt.xlabel("Time [s]")
11        plt.ylabel("Magnitude")
```



```

12     plt.grid()
13     else:
14         plotext.clear_data()
15         plotext.plot(t, signal)
16         plotext.title(title)
17         plotext.xlabel("Time [s]")
18         plotext.ylabel("Magnitude")
19         plotext.show()

```

Esta función grafica la señal en el dominio del tiempo, donde el eje x representa el tiempo (en segundos) y el eje y representa la magnitud de la señal. Dependiendo de si se selecciona graficar en la terminal o en una interfaz gráfica, se utilizan diferentes librerías (matplotlib o plotext).

- Si `self._graph_in_terminal` es `False`, se usa matplotlib para graficar en una ventana gráfica.
- Si es `True`, se usa plotext para graficar directamente en la terminal.

```

1  def graph_signal_voltage_frequency(self, frequencies, magnitudes, title):
2  if not self._graph_in_terminal:
3      plt.figure(figsize=(12, 6))
4      plt.plot(frequencies, magnitudes)
5      plt.title(title)
6      plt.xlabel("Frequency [Hz]")
7      plt.ylabel("Magnitude")
8      plt.grid()
9  else:
10     plotext.clear_data()
11     plotext.plot(frequencies, magnitudes)
12     plotext.title(title)
13     plotext.xlabel("Frequency [Hz]")
14     plotext.ylabel("Magnitude")
15     plotext.show()

```

Esta función grafica la señal en el dominio de la frecuencia. En el eje x, se representan las frecuencias (en Hz), y en el eje y, las magnitudes de la señal para cada frecuencia. Nuevamente, dependiendo del valor de `self._graph_in_terminal`, se selecciona si se grafica en una interfaz gráfica o directamente en la terminal.

```

1  def plot_now(self):
2  if not self._graph_in_terminal:
3      plt.tight_layout()
4      plt.show()
5  else:
6      pass

```

Esta función se encarga de mostrar el gráfico después de haberlo configurado. Si estamos usando matplotlib, se asegura de que el diseño del gráfico sea el adecuado (`tight_layout`) y luego lo muestra en pantalla. Si estamos graficando en la terminal, no hace nada, ya que la función `plotext.show()` ya gestiona la visualización.

```

1  if not self._graph_in_terminal:
2      # Uso de matplotlib
3  else:
4      # Uso de plotext

```

Estas estructuras determinan si los gráficos se crean usando matplotlib o plotext, dependiendo de si estamos trabajando en un entorno gráfico o en la terminal.

### 5.7.5. Explicacion bias\_motors.py

Este código controla el movimiento de la silla de ruedas utilizando un conjunto de sensores ultrasónicos, LEDs, un buzzer, y motores. Dependiendo del comando ingresado por el usuario (`forward`, `left`, `backwards`, `right` o `stop`), la silla de ruedas se mueve en la dirección indicada o se detiene. Los sensores ultrasónicos verifican si hay obstáculos a menos de 20 cm, y si se detecta uno, los LEDs y el buzzer se activan para alertar al usuario y bloquear el movimiento.

```

1 def main():
2     biasMotor = MotorBias(echo_forward=18, trigger_forward=17, echo_backwards=23,
3                           trigger_backwards=22,
4                           echo_right=5, trigger_right=6, echo_left=25, trigger_left
5                           =24,
6                           led_forward=16, led_backwards=20, led_left=21, led_right
7                           =26,
8                           buzzer=12, motor1_in1=13, motor1_in2=19, motor2_in1=7,
9                           motor2_in2=8)
10    while True:
11        command = input("Enter command (forward/left/backwards/right/stop): ").strip
12        ()
13        biasMotor.move_if_possible(command)

```

Se crea un objeto MotorBias que controla los sensores, LEDs, buzzer y motores. Cada componente se asocia a un pin GPIO. También se pide al usuario que ingrese un comando de dirección (forward, backwards, left, right, stop). El comando se pasa al método move\_if\_possible.

```

1 class MotorBias:
2     def __init__(self, echo_forward, trigger_forward, echo_backwards,
3                 trigger_backwards, echo_right,
4                 trigger_right, echo_left, trigger_left, led_forward, led_backwards,
5                 led_left,
6                 led_right, buzzer, motor1_in1, motor1_in2, motor2_in1, motor2_in2):

```

El constructor de la clase inicializa todos los sensores, LEDs, buzzer y motores utilizando los pines especificados. También configura los pines para controlar los motores con PWM.

```

1     self._ultrasonic_forward = DistanceSensor(echo=echo_forward, trigger=
2     trigger_forward, pin_factory=factory)
3 self._ultrasonic_backwards = DistanceSensor(echo=echo_backwards, trigger=
4     trigger_backwards, pin_factory=factory)

```

Sensores ultrasónicos: Detectan obstáculos en las direcciones hacia adelante, atrás, izquierda y derecha.

```

1     self._led_forward = PWMLED(led_forward, pin_factory=factory)
2 self._buzzer = Buzzer(buzzer)

```

LEDs y Buzzer: Se usan para alertar si se detecta un obstáculo.

```

1     self._motor1_in1 = PWMOutputDevice(motor1_in1, frequency=50, pin_factory=factory)
2 self._motor2_in1 = PWMOutputDevice(motor2_in1, frequency=50, pin_factory=factory)

```

Motores: Controlados mediante PWMOutputDevice, permiten ajustar la velocidad de la silla de ruedas.

```

1     def move_if_possible(self, command):
2     try:
3         if command == "forward":
4             distance = self._ultrasonic_forward.distance * 100
5             if distance < 20:
6                 self._led_forward.on()
7                 self._buzzer.on()
8                 print(f"Obstacle forward: {distance:.1f} cm. Blocked movement.")
9             else:
10                self._led_forward.off()
11                self._buzzer.off()
12                self.move_forward(25)
13            ...

```

- Dependiendo del comando (forward, backwards, etc.), el código revisa si hay obstáculos a través de los sensores ultrasónicos.
- Si se detecta un obstáculo dentro de los 20 cm, el movimiento se bloquea, y se encienden el LED y el buzzer.
- Si no hay obstáculos, se apagan los LEDs y el buzzer, y se ejecuta el movimiento correspondiente a través de los métodos de la clase (move\_forward, move\_backward, etc.).

```

1 def set_motor_speed(self, motor_in1, motor_in2, speed, invert=False):
2     if speed > 0:
3         motor_in1.value = ((100.0 - speed) / 100.0) if invert else speed / 100.0
4         motor_in2.value = 0

```

- Controla la velocidad de los motores utilizando PWM.
- Dependiendo del valor de speed, el método ajusta el ciclo de trabajo para definir la velocidad de los motores. Si speed es negativo, invierte el sentido del motor.

```

1 def move_forward(self, speed):
2     self.set_motor_speed(self._motor1_in1, self._motor1_in2, speed, invert=True)
3     self.set_motor_speed(self._motor2_in1, self._motor2_in2, speed, invert=True)

```

Estos métodos (move\_forward, move\_backward, turn\_left, turn\_right) controlan los motores para mover la silla de ruedas en la dirección deseada. Ajustan la velocidad de ambos motores para mover la silla hacia adelante, atrás o realizar giros.

```

1 def brake(self):
2     self.set_motor_speed(self._motor1_in1, self._motor1_in2, 0, invert=True)
3     self.set_motor_speed(self._motor2_in1, self._motor2_in2, 0, invert=True)

```

El método brake detiene ambos motores al reducir la velocidad a 0.

### 5.7.6. Explicación bias\_reception.py

Este código se enfoca en la adquisición de datos de señales EEG a través de la comunicación serie entre un microcontrolador (RP2040 Zero) y la Raspberry Pi 4. Utiliza serial para establecer la conexión y recibir los datos en formato JSON, luego los procesa y grafica las señales de EEG utilizando la clase GraphingBias. El código también implementa funcionalidades para capturar y manejar los datos en tiempo real y mostrarlos en el terminal de la computadora, probablemente para monitoreo y análisis en vivo.

El flujo general del programa incluye:

- Inicialización de la comunicación serie con el microcontrolador.
- Recepción de datos en tiempo real, procesándolos para convertirlos a un formato adecuado.
- Graficación de las señales EEG obtenidas, representando el voltaje en función del tiempo para cada canal.

```

1 def main():
2     n = 1000
3     fs = 500
4     number_of_channels = 4
5     biasReception = ReceptionBias()
6     signals = biasReception.get_real_data(channels=number_of_channels, n=n)
7     biasGraphing = GraphingBias(graph_in_terminal=True)
8     for ch, signal in signals.items():
9         t = np.arange(len(signals[ch])) / fs
10        biasGraphing.graph_signal_voltage_time(t=t, signal=np.array(signal), title="
Signal {}".format(ch))

```

La función principal configura los parámetros clave:

- n: El número de muestras por canal (1000).
- fs: La frecuencia de muestreo (500 Hz).
- number\_of\_channels: El número de canales a recibir (4 canales).

Después de esto, se instancia la clase ReceptionBias, que se encarga de la recepción de datos. Luego, los datos de cada canal se pasan a la clase GraphingBias para ser graficados.

```

1 def __init__(self, port='/dev/serial0', baudrate=115200, timeout=1):
2     self._port = port
3     self._baudrate = baudrate
4     self._timeout = timeout

```

Inicializa la clase con los parámetros del puerto serial, la tasa de baudios y el tiempo de espera.

```

1 def get_real_data(self, channels, n):
2     self._ser = self.init_serial(self._port, self._baudrate, self._timeout)
3     try:
4         real_eeg_signals = self.capture_signals(channels=channels, n=n)
5         return real_eeg_signals
6     finally:
7         self._ser.close()

```

Este método:

- Establece la comunicación serie inicializando el puerto.
- Llama a `capture_signals()` para recibir los datos de EEG de los canales especificados.
- Cierra la conexión serial después de terminar la captura.

```

1 def capture_signals(self, channels, n):
2     signals = {f'ch{ch}': [] for ch in range(channels)}
3     start_time = time.time()
4     while len(signals['ch3']) < n:
5         if self._ser.in_waiting > 0:
6             try:
7                 data = self._ser.readline().decode('utf-8').strip()
8                 eeg_data = self.process_data(data)
9                 if eeg_data:
10                     for ch in range(channels):
11                         signals[f'ch{ch}'].extend(eeg_data[f'ch{ch}'])
12             except Exception as e:
13                 print("Can't be decoded")
14     elapsed_time = time.time() - start_time
15     print(f"elapsed time: {elapsed_time}")
16     for ch in range(channels):
17         signals[f'ch{ch}'] = signals[f'ch{ch}'][:n]
18     return signals

```

Este método:

- Inicializa un diccionario `signals` para almacenar las señales de EEG de cada canal.
- Entra en un bucle que continúa hasta que se hayan recibido `n` muestras para el último canal (`ch3`).
- Si hay datos disponibles en el buffer de la conexión serial, los lee y procesa, añadiendo los datos al canal correspondiente.
- Una vez recibidos suficientes datos, ajusta la longitud de las señales y las devuelve.

```

1 def init_serial(self, port, baudrate, timeout):
2     return serial.Serial(port, baudrate, timeout=timeout)

```

Este método inicializa la comunicación UART usando la biblioteca serial.

```

1 def process_data(self, data):
2     try:
3         json_data = json.loads(data)
4         print(json_data)
5         return json_data
6     except json.JSONDecodeError as e:
7         print(f"Error decoding JSON: {e}")
8         return None

```

Este método intenta convertir la línea de datos recibida a un formato JSON. Si falla, muestra un mensaje de error y retorna `None`.

### 5.7.7. Explicacion reception.c

Este código está diseñado para funcionar en un sistema embebido, como la Raspberry Pi Pico, y tiene como objetivo leer valores de varios canales de un conversor analógico-digital (ADC), convertir estos valores a milivoltios (mV), almacenarlos, generar una cadena JSON con los datos y enviarla a través de UART. Se emplean cuatro canales de ADC para tomar un total de 1000 muestras, lo que permite leer señales analógicas de diferentes fuentes simultáneamente. Además, el código tiene un temporizador para controlar la frecuencia de muestreo de los datos.

```
1  #define UART_ID uart0
2  #define BAUDRATE 115200
3  #define NUMBER_OF_CHANNELS 4
4  #define NUMBER_OF_TOTAL_SAMPLES 1000
5  #define ADC_DELAY_US -2000
```

- UART\_ID define el identificador del puerto UART a utilizar.
- BAUDRATE establece la velocidad de transmisión de datos (115200 bits por segundo).
- NUMBER\_OF\_CHANNELS es la cantidad de canales ADC que se van a leer (4).
- NUMBER\_OF\_TOTAL\_SAMPLES es el número total de muestras que se tomarán (1000).
- ADC\_DELAY\_US especifica el tiempo de retardo entre lecturas del ADC (2000 microsegundos, lo que equivale a una frecuencia de 500 Hz).

```
1  const float CONVERSION_FACTOR = 3.3f * 1000 / (1 << 12);
```

Esta constante convierte los valores digitales del ADC a milivoltios. El valor del ADC es de 12 bits, y se multiplica por 3.3 (la referencia de voltaje del ADC) y por 1000 para convertir a milivoltios.

```
1  uint16_t values_mv[NUMBER_OF_CHANNELS][NUMBER_OF_TOTAL_SAMPLES];
2  static bool sampling_done = false;
```

Se declara una matriz para almacenar las muestras convertidas a milivoltios. sampling\_done es una bandera que indica cuándo se ha completado el proceso de muestreo.

```
1  void init_uart(uint8_t tx_pin, uint8_t rx_pin);
2  void init_adc(uint8_t adc_channel_0, uint8_t adc_channel_1, uint8_t adc_channel_2,
3  uint8_t adc_channel_3);
4  bool read_adc(struct repeating_timer *t);
5  void start_sampling(void);
6  void build_json(char *data, uint total_bytes);
7  void send_data(char *data);
```

Aquí se definen los prototipos de las funciones principales que configuran y controlan la UART, ADC, la lectura de datos, el muestreo, la generación del JSON y el envío de datos.

```
1  void init_uart(uint8_t tx_pin, uint8_t rx_pin) {
2  uart_init(UART_ID, BAUDRATE);
3  gpio_set_function(tx_pin, GPIO_FUNC_UART);
4  gpio_set_function(rx_pin, GPIO_FUNC_UART);
5  uart_set_format(UART_ID, 8, 1, UART_PARITY_NONE);
6  uart_set_fifo_enabled(UART_ID, true);
7  }
```

Esta función configura el puerto UART con los parámetros establecidos (velocidad, formato, pines) para permitir la transmisión de datos.

```
1  void init_adc(uint8_t adc_channel_0, uint8_t adc_channel_1, uint8_t adc_channel_2,
2  uint8_t adc_channel_3) {
3  adc_init();
4  adc_gpio_init(adc_channel_0);
5  adc_gpio_init(adc_channel_1);
6  adc_gpio_init(adc_channel_2);
7  adc_gpio_init(adc_channel_3);
8  }
```

Se inicializan los canales ADC y se configuran los pines correspondientes para permitir la lectura de señales analógicas.

```

1  bool read_adc(struct repeating_timer *t) {
2      static uint sampling_count = 0;
3      if (sampling_count >= NUMBER_OF_TOTAL_SAMPLES) {
4          sampling_done = true;
5          cancel_repeating_timer(t);
6          sampling_count = 0;
7      } else {
8          for (int channel = 0; channel < NUMBER_OF_CHANNELS; channel++) {
9              adc_select_input(channel);
10             uint16_t adc_value = adc_read();
11             values_mv[channel][sampling_count] = round(adc_value * CONVERSION_FACTOR)
12             ;
13             sampling_count++;
14         }
15         return true;
16     }

```

Cada vez que se ejecuta esta función, se lee el valor de cada canal ADC, se convierte a milivoltios y se almacena en la matriz `values_mv`. Cuando se alcanzan las 1000 muestras, el temporizador se detiene y la bandera `sampling_done` se activa.

```

1  void start_sampling(void) {
2      static struct repeating_timer timer;
3      sampling_done = false;
4      add_repeating_timer_us(ADC_DELAY_US, read_adc, NULL, &timer);
5  }

```

Esta función inicia el temporizador, que repetidamente llama a `read_adc` con un intervalo de 2000 microsegundos.

```

1  void build_json(char *data, uint total_bytes) {
2      char *str = (char*) malloc(total_bytes);
3      if (str == NULL) {
4          printf("Memory allocation failed\n");
5          return;
6      }
7      strcpy(str, "{");
8      for (int channel = 0; channel < NUMBER_OF_CHANNELS; channel++) {
9          char label[8];
10         sprintf(label, "\"ch%d\":[" , channel);
11         strcat(str, label);
12         for (int sampling_number = 0; sampling_number < NUMBER_OF_TOTAL_SAMPLES;
13             sampling_number++) {
14             char aux[sizeof("0000,")];
15             sprintf(aux, "%d,", values_mv[channel][sampling_number]);
16             strcat(str, aux);
17         }
18         strcat(str, "],");
19     }
20     strcat(str, "}\n");
21     strcpy(data, str);
22     free(str);
23 }

```

Esta función genera una cadena JSON con los valores de cada canal de ADC, lo que permite estructurar los datos en un formato adecuado para su envío.

```

1  void send_data(char *data) {
2      uart_puts(UART_ID, data);
3  }

```

Los datos JSON generados se envían a través de UART utilizando la función `uart_puts`.

## 6. Circuitos usados

En este capítulo se incluirán imágenes de los esquemas de los circuitos, junto con su representación gráfica en formato virtual de los circuitos impresos, así como fotografías de los circuitos completos una vez finalizados.

### 6.1. Esquemático de los filtros del EEG

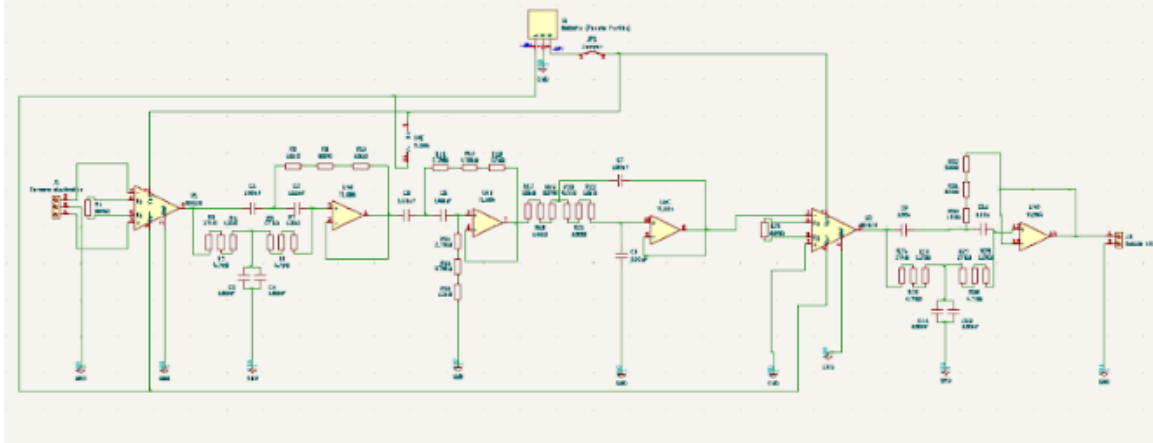


Figura 12: Esquemático de los filtros del EEG

### 6.2. Esquemático del sistema de control

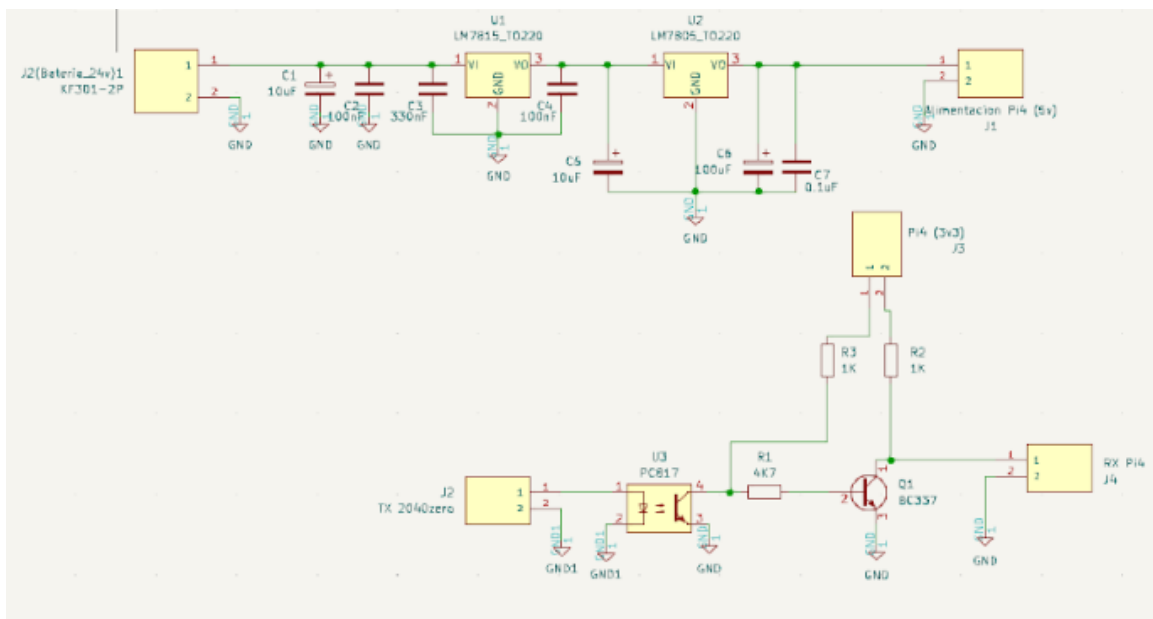


Figura 13: Esquemáticos del sistema de control

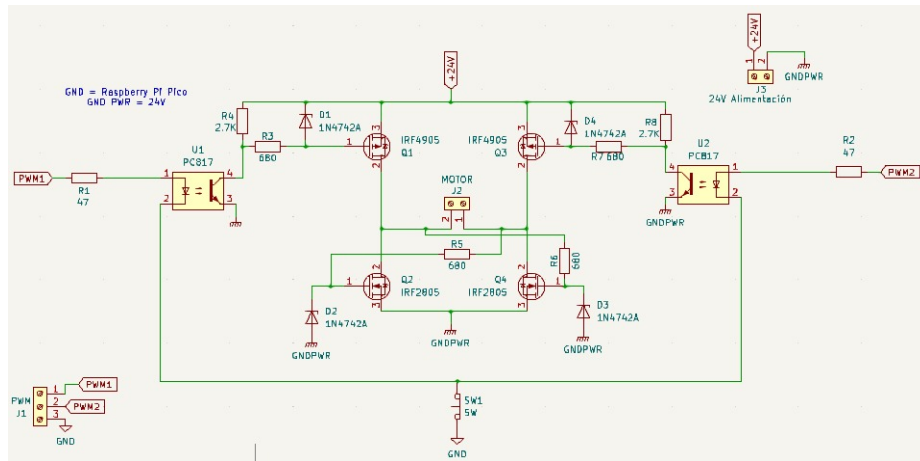


Figura 15: Puente H

### 6.3. Esquemático del sistema Offset

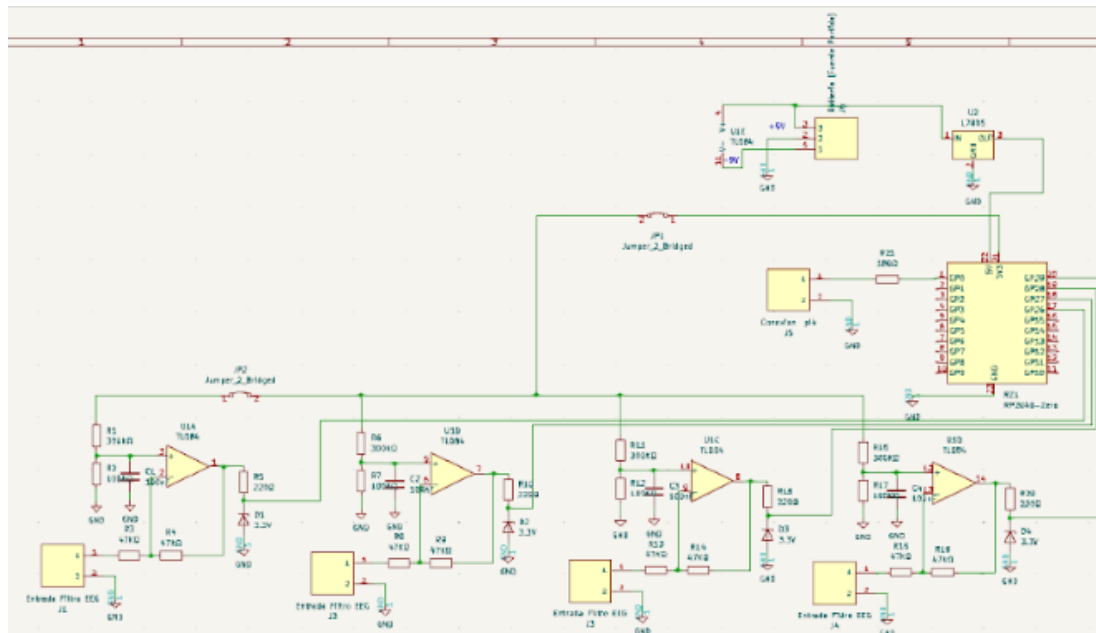


Figura 14: Esquemático del sistema de Offset



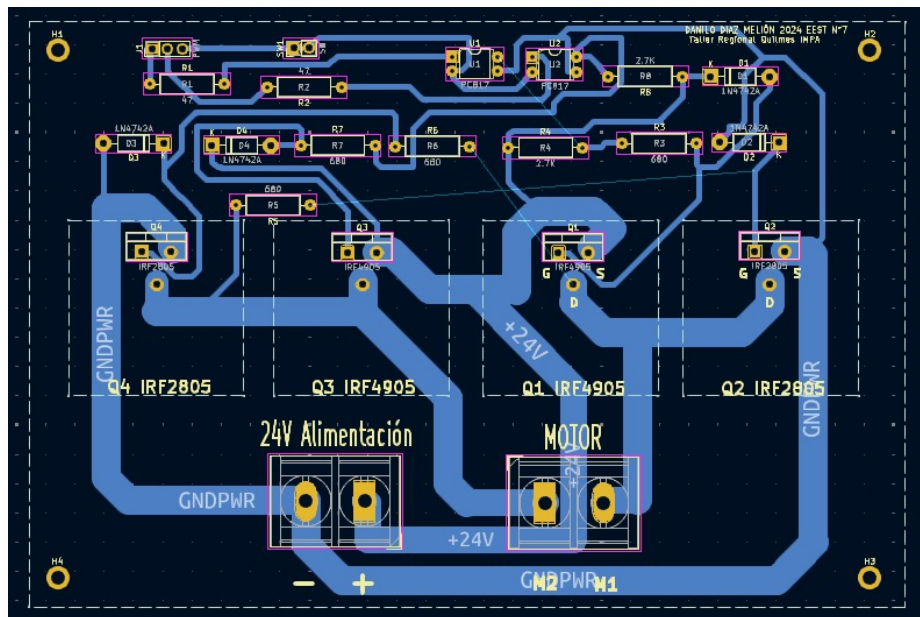


Figura 16: PCB del Puente H

#### 6.4. Esquemáticos de los Puentes H

#### 6.5. Esquemático del sistema de emergencia

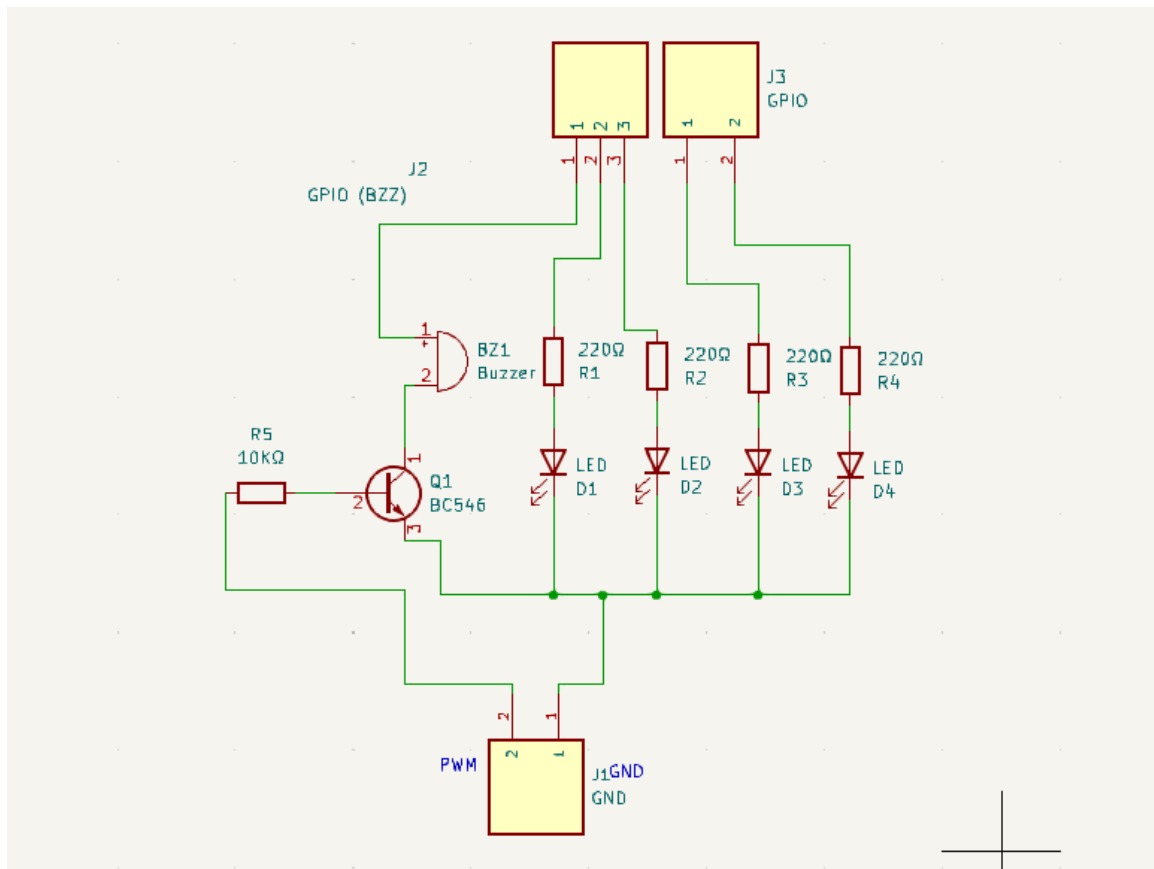
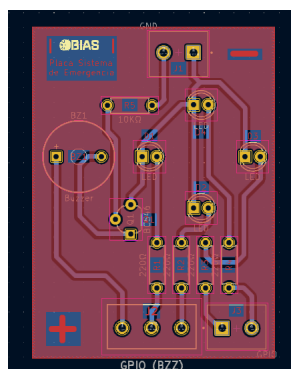
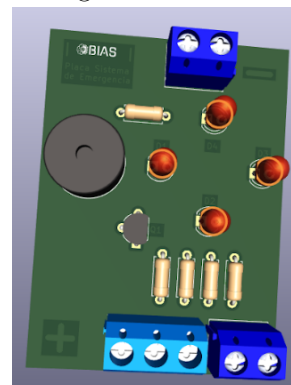


Figura 17: Esquemático del sistema de emergencia



(a) PCB Del sistema de emergencia



(b) Modelo 3D Del sistema de emergencia

#### 6.6. Lista de materiales

Los programas desarrollados para el proyecto, tales como los filtros digitales, sistemas de inteligencia artificial y otros códigos relacionados, se almacenan en los siguientes dispositivos:

- Raspberry Pi 4
- RP2040-zero

#### **6.6.1. EEG**

1. TL084 (x4)
2. AD620 (x8)
3. Capacitores 100nF (x48)
4. Resistencias 100 (x4)
5. Resistencias 120 (x16)
6. Resistencias 560 (x8)
7. Resistencias 820 (x16)
8. Resistencias 1,5k (x8)
9. Resistencias 4,7k (x16)
10. Resistencias 12k (x8)
11. Resistencias 15k (x8)
12. Resistencias 27k (x16)
13. Resistencias 470k (x8)
14. Resistencias 2,7M (x8)
15. Electrodo (x9)

#### **6.6.2. Placa OffSet**

1. L7805 (x1)
2. TL084 (x4)
3. Jumpers (x2)
4. Diodos Zenner 3,3V (x4)
5. Capacitores 100nF (x4)
6. Resistencias 220 (x4)
7. Resistencias 50k (x8)
8. Resistencias 100k (x8)

#### **6.6.3. Sistema de Emergencia**

1. Transistor BC546 (x1)
2. Buzzer 5V (x1)
3. LEDs Rojos (x4)
4. Resistencias 220 (x4)
5. Resistencia 4,7k (x1)

#### **6.6.4. Puentes H**

1. Transistores IRF4905 (x4)
2. Transistores IRF2805 (x4)
3. Optoacopladores PC817 (x4)
4. Motores 24V 575W (x2)
5. Resistencias 100 (x4)
6. Resistencias 180 (x4)
7. Resistencias 1k (x4)
8. Resistencias 4,7k (x4)

#### **6.6.5. Placa reguladora de tensión**

1. LM7815 (x1)
2. Capacitor 100F (x1)
3. Capacitor 330F (x1)
4. Capacitor 470F 35V (x1)
5. Capacitor 470F 25V (x1)

## **7. Conclusiones**

En esta sección se presentará un análisis detallado de los conocimientos adquiridos a lo largo del desarrollo del proyecto, así como las conclusiones a las que hemos llegado durante este proceso. Se incluirán las conclusiones finales obtenidas, así como una evaluación de las principales limitaciones que afectaron el progreso del trabajo. Además, se ofrecerá un resumen de los resultados alcanzados, destacando los aspectos más relevantes y significativos. Finalmente, se abordarán las perspectivas futuras del proyecto, delineando los pasos a seguir para continuar su desarrollo y mejorarlo en el futuro.

### **7.1. Limitaciones**

Durante la ejecución de este proyecto, nos enfrentamos a diversas limitaciones, algunas de las cuales fueron ocasionadas por circunstancias personales de cada integrante del equipo. Como es común en todo proyecto, al principio experimentamos dificultades en la organización y carecíamos de ciertos conocimientos técnicos, los cuales fuimos adquiriendo a lo largo del año.

El tiempo dedicado al proyecto quizá no fue suficiente, considerando la magnitud y la ambición del mismo. Además de estos desafíos iniciales, una de las principales limitaciones fue el prolongado tiempo de espera para obtener los materiales necesarios, los cuales provenían principalmente de la cooperadora. Dado que esta era nuestra principal fuente de herramientas y componentes, nuestro progreso estuvo estrictamente limitado por los tiempos de adquisición manejados por la institución.

Asimismo, al estar creando algo “nuevo”, nos vimos obligados a adoptar un enfoque basado en la prueba y error, ya que la información disponible sobre los temas específicos que requeríamos era escasa. Esta situación también representó un obstáculo significativo, ya que corregir nuestros errores consumió una parte considerable de nuestro tiempo.

Finalmente, al tratarse de un proyecto desarrollado en el ámbito escolar, también nos vimos limitados por el tiempo disponible en la escuela. No pudimos dedicar todo nuestro tiempo al proyecto, ya que debíamos atender otras asignaturas y compromisos académicos.

## 7.2. Trabajo Futuro

El proyecto presenta un alto potencial de rentabilidad en el futuro, sustentado en su carácter innovador y las posibilidades que ofrece su desarrollo continuo. Al tratarse de un invento único en su categoría, se prevé que, con el tiempo y el trabajo adicional necesario, este dispositivo pueda alcanzar un nivel de madurez suficiente para ser comercializado a gran escala.

La inversión en tiempo y recursos adicionales permitirá perfeccionar sus características técnicas, optimizar su funcionalidad, y adaptar el producto a las necesidades de un mercado en expansión. Una vez finalizado y refinado, el proyecto podrá captar la atención de un público más amplio, lo que incrementará su viabilidad comercial y su posicionamiento en el mercado.

En caso de que el desarrollo del proyecto no se complete en su totalidad, este podrá ser aprovechado por la institución educativa, permitiendo que otros grupos de estudiantes continúen su avance. De esta manera, el proyecto no solo servirá como un recurso valioso para futuras generaciones, sino que también fomentará la colaboración y el aprendizaje continuo dentro de la comunidad académica, garantizando su evolución y finalización.

## 7.3. Informacion Adicional

La silla desarrollada en este proyecto es un prototipo, es decir, puede contener fallas de todo tipo, y es sujeto a modificaciones a futuro. Se recomienda su uso siempre acompañado, en zonas amplias libres de obstáculos. No se debe usar en lugares húmedos o si el equipo está mojado, y la inteligencia artificial está adaptada a las muestras de una persona del equipo del proyecto, por lo que para ser usada por alguien mas se deberán muestrear los patrones de movimiento de esta misma.

### 7.3.1. Programas Utilizados

Para el desarrollo del proyecto, hemos requerido hacer uso de los siguientes programas, con sus enlaces principales:

- Github
- Visual Studio Code
- Thonny
- Overleaf
- Pycharm
- Termius
- Kicad
- Livewire
- LTSpice
- GitHub Desktop

### 7.3.2. Agradecimientos

Queremos expresar nuestro más profundo agradecimiento a los profesores que nos han brindado su invaluable apoyo en la realización de este proyecto:

- Fabrizio Carlassara: Su asistencia fue crucial en la programación de la mayor parte del proyecto, así como en la creación de circuitos y esquemáticos, la búsqueda de materiales y el desarrollo de los filtros necesarios.
- Sergio Medina: Nos brindó su experiencia en la presentación y comercialización del proyecto, además de colaborar en la programación, la búsqueda de materiales y la planificación de los puentes H.
- Carlos Bianco: Su apoyo fue esencial en las pruebas y acondicionamiento de los motores, proporcionándonos herramientas y baterías para dichas pruebas, además de colaborar en la búsqueda de materiales.
- Daniel Espósito: Nos proporcionó motores y materiales, además de enseñarnos a construir los puentes H para los motores y nos ayudó a solucionar problemas con las pistas de los circuitos para las altas corrientes.
- Federico Solomiewicz: Colaboró principalmente en el desarrollo de los filtros, proporcionándonos herramientas para el proyecto y estando siempre disponible para resolver cualquier duda que surgiera.
- Juan Carlos Ruiz: Su colaboración fue esencial en las pruebas de los motores para la silla de ruedas, además de ayudarnos principalmente en la creación de los puentes H para los motores y sus circuitos impresos.
- Ana Hartvig: Su ayuda fue imprescindible durante la creación del EEG. Gracias a ella, pudimos diseñar el EEG, aprendimos como colocar los electrodos y pudimos acceder al hospital El Cruce, Dr. Nestor Kirchner y aprender mas sobre EEGs.

## 8. Referencias

En este apartado se presentarán de manera detallada las referencias consultadas al inicio del proyecto, las cuales fueron fundamentales para la conceptualización y desarrollo de la idea final. Estas fuentes proporcionaron el marco teórico y práctico necesario para orientar la dirección del trabajo, influyendo en las decisiones tomadas durante el proceso de diseño y ejecución.

### 8.1. Brain Computer Interface

La primera referencia utilizada en este proyecto es un informe técnico titulado "Brain Computer Interface", escrito por María Madrid Sobrino, el cual aborda el desarrollo de una interfaz cerebro-computadora (BCI) basada en señales EEG, específicamente empleando potenciales evocados visuales de estado estacionario (SSVEP).

Este documento resultó fundamental para nuestra investigación, ya que abarca los siguientes aspectos relevantes:

- Introducción a las interfaces cerebro-computadora (BCI): Una interfaz cerebro-computadora (BCI) es un sistema que permite a los usuarios comunicar sus intenciones al entorno externo sin necesidad de utilizar mecanismos fisiológicos naturales, como los nervios periféricos o los músculos. Estas interfaces están diseñadas principalmente para mejorar la comunicación de personas con discapacidades severas, aunque también tienen aplicaciones en la rehabilitación neurológica, en la industria de los videojuegos, entre otros. El informe distingue entre dos tipos de BCI: los invasivos, en los cuales los electrodos se implantan directamente en el cerebro, y los no invasivos, donde los electrodos se colocan en la superficie del cráneo. Los BCIs no invasivos, basados en la electroencefalografía (EEG), son los más ampliamente utilizados debido a su practicidad y menor riesgo.

- BCI basado en SSVEP: El informe describe un sistema BCI basado en SSVEP (Potenciales Evocados Visuales de Estado Estacionario), que utiliza la respuesta cerebral a estímulos visuales repetitivos. Cuando un estímulo visual parpadea a una frecuencia constante, el cerebro genera señales EEG a la misma frecuencia del estímulo. Se destaca la alta efectividad de estos sistemas, que presentan una elevada relación señal-ruido y requieren poco entrenamiento por parte del usuario. Sin embargo, se señala que algunos estímulos visuales pueden provocar fatiga o, en ciertos casos, ataques epilépticos.

La referencia detalla el uso de la plataforma BCI2000 para la adquisición de datos, procesamiento de señales y presentación de estímulos. El sistema fue desarrollado empleando C++ y Matlab, lo que permitió la integración de un módulo de procesamiento de señales en Matlab. El sistema desarrollado incluye una aplicación que presenta 16 opciones visuales organizadas en cuatro grupos, permitiendo al usuario seleccionar una imagen específica enfocando su atención en ella. El método de clasificación empleado se basa en la frecuencia de los estímulos visuales, y la integración con BCI2000 facilita la ejecución de un sistema de lazo cerrado, donde las señales cerebrales controlan directamente el proceso de selección de imágenes.

Esta referencia fue clave para establecer las bases teóricas y técnicas necesarias para la implementación de un sistema de control basado en señales cerebrales, aplicable a nuestro proyecto de silla de ruedas.

## **8.2. Artificial Intelligence for Real-Time Decoding of Motor Commands from ECoG of Disabled Subjects for Chronic Brain-Computer Interfacing**

La segunda referencia utilizada corresponde a la tesis doctoral titulada "Artificial Intelligence for Real-Time Decoding of Motor Commands from ECoG of Disabled Subjects for Chronic Brain-Computer Interfacing", presentada por Maciej Sliwowski en la Université Grenoble Alpes en 2022.

Esta tesis aborda el uso de inteligencia artificial (IA) para la decodificación en tiempo real de comandos motores a partir de señales de electrocorticografía (ECoG) en sujetos con discapacidades motoras. El objetivo principal es mejorar la comunicación y el control mediante interfaces cerebro-computadora (BCI). Los sistemas BCI ofrecen una herramienta de asistencia crucial para pacientes con tetraplejia, proporcionando un medio de interacción directa entre el cerebro y el entorno, compensando la pérdida de funciones motoras.

El enfoque principal del trabajo se centra en los BCI basados en ECoG, que presentan un equilibrio óptimo entre los sistemas intracorticales, más invasivos, y los métodos no invasivos, como el EEG, que suelen ser menos precisos. La tesis estudia específicamente la decodificación de movimientos continuos de traslación de la mano en un paciente tetrapléjico, logrando avances importantes en la interpretación de las señales cerebrales para aplicaciones prácticas en la rehabilitación motora.

Este trabajo es relevante para nuestro proyecto ya que aborda la integración de IA en la interpretación de señales neuronales, lo que resulta fundamental para desarrollar sistemas de control eficientes en tiempo real, como es el caso de nuestra silla de ruedas controlada por la mente. Además, su enfoque en la ECoG aporta perspectivas valiosas para mejorar la precisión y la respuesta del sistema BCI en sujetos con discapacidades motoras severas.

## **8.3. Controlled Wheelchair Based on Brain Computer Interface using Neurosky Mindwave Mobile 2**

El artículo presenta el desarrollo de una silla de ruedas controlada mediante una interfaz cerebro-computadora (BCI) utilizando el dispositivo Neurosky Mindwave Mobile 2 para la adquisición de señales EEG. El propósito de este sistema es mejorar la movilidad y la calidad de vida de pacientes post-ictus, permitiéndoles controlar la silla de ruedas a través de sus ondas cerebrales.

Este artículo resultó valioso para nuestras primeras pruebas, dado que inicialmente contábamos con un dispositivo Neurosky Mindwave Mobile 2. Sin embargo, debido a limitaciones en la progra-

mación y procesamiento de señales, decidimos descartar esta opción y desarrollar nuestro propio sistema EEG.

El sistema BCI permite a los usuarios controlar dispositivos mediante señales cerebrales, las cuales son registradas utilizando electroencefalografía (EEG). El Neurosky Mindwave Mobile 2 es un dispositivo portátil de bajo costo diseñado para registrar dichas señales EEG. En el contexto de este proyecto, las señales cerebrales se utilizan para controlar una silla de ruedas eléctrica, que originalmente funcionaba a través de un joystick.

El módulo de control del joystick fue reemplazado por un controlador basado en microcontroladores, que procesa las señales cerebrales capturadas y las clasifica utilizando un software desarrollado en Matlab. Este proceso permite que las señales EEG controlen el motor de la silla de ruedas en tiempo real.

En la metodología utilizada se emplea el método de motor imagery, el cual consiste en registrar y analizar las ondas cerebrales asociadas a la imaginación de movimientos motores, tales como caminar o mover las manos. Las señales capturadas por el dispositivo Neurosky se procesan en Matlab y se clasifican en cinco clases de movimiento: avanzar, retroceder, girar a la derecha, girar a la izquierda y una posición neutral.

El sistema incorpora la funcionalidad eSense, una característica del dispositivo que mide los niveles de atención y meditación del usuario, contribuyendo a la precisión del control basado en señales cerebrales.

#### **8.4. Brain-computer interface for electric wheelchair based on alpha waves of EEG signal**

El presente estudio aborda el desarrollo de una interfaz cerebro-computadora (Brain-Computer Interface, BCI) destinada al control de una silla de ruedas eléctrica mediante la detección y análisis de ondas alfa generadas por el cerebro, capturadas a través de un electroencefalograma (EEG). El principal objetivo de este sistema es proporcionar asistencia a personas que padecen enfermedades neurológicas graves que afectan severamente su capacidad de movilidad. La solución propuesta destaca por su simplicidad operativa, dado que emplea un número reducido de electrodos y no requiere un extenso período de entrenamiento para los usuarios.

Este estudio fue utilizado como referencia en el desarrollo de nuestro propio proyecto, el cual comparte un enfoque similar en la implementación de un sistema BCI para el control de una silla de ruedas. A partir de la revisión detallada de sus métodos y resultados, adaptamos y optimizamos el diseño de nuestros propios circuitos.

#### **8.5. A Neural Network Based Brain-Computer Interface for Classification of Movement Related EEG**

La tesis elaborada por Pontus Forslund y presentada en Linköping en diciembre de 2003, trata sobre el diseño de una interfaz cerebro-computadora (BCI) basada en electroencefalografía (EEG), enfocada en la clasificación de movimientos de la mano en dos dimensiones. El objetivo de dicho proyecto es desarrollar un sistema de comunicación intuitivo para individuos con discapacidades motoras graves.

En este estudio, se registraron señales EEG de un sujeto mientras operaba un joystick en cuatro direcciones. Las señales capturadas fueron procesadas mediante un modelo autorregresivo para extraer características relevantes, que luego fueron empleadas como entradas para una red neuronal artificial, diseñada para clasificar la dirección de los movimientos.

Este estudio ha sido utilizado como referencia para nuestro proyecto, específicamente en lo que respecta al reconocimiento de señales EEG y su asignación a los comandos de control del movimiento de la silla de ruedas controlada mediante señales cerebrales.



## 9. Lista de códigos

### 9.1. Códigos principales

#### 9.1.1. bias.py

```
1 from bias_dsp import FilterBias, ProcessingBias
2 from bias_reception import ReceptionBias
3 import numpy as np
4 from bias_graphing import GraphingBias
5 from bias_motors import MotorBias
6 from bias_ai import AIBias
7
8 class BiasClass:
9     # Constructor
10     def __init__(self, n, fs, channels, port, baudrate, timeout):
11         # Define propieties for the class
12         self._n = n
13         self._fs = fs
14         self._number_of_channels = channels
15         self._duration = self._n / self._fs
16         self._port = port
17         self._baudrate = baudrate
18         self._timeout = timeout
19         self._commands = ["forward", "backwards", "left", "right", "stop", "rest"]
20         self._samples_trainig_command = 100
21
22         # Create objects as propieties in order to apply the rest of the code in Bias
23         class
24             self._biasReception = ReceptionBias(self._port, self._baudrate, self._timeout
25 )
26             self._biasFilter = FilterBias(n=self._n, fs=self._fs, notch=True, bandpass=
27 True, fir=False, iir=False)
28             self._biasProcessing = ProcessingBias(n=self._n, fs=self._fs)
29             self._biasGraphing = GraphingBias(graph_in_terminal=True)
30             self._biasMotor = MotorBias(echo_forward=18, trigger_forward=17,
31 echo_backwards=23, trigger_backwards=22, echo_right=5, trigger_right=6,
32 echo_left=25, trigger_left=24, led_forward=16,
33 led_backwards=20, led_left=21, led_right=26, buzzer=12, motor1_in1=13,
34 motor1_in2=19, motor2_in1=7, motor2_in2=8)
35             self._biasAI = AIBias(self._n, self._fs, self._number_of_channels, self.
36 _commands)
37
38     def train_ai_model(self, save_path, saved_dataset_path):
39         self._biasAI.collect_and_train(reception_instance=self._biasReception,
40 filter_instance=self._biasFilter,
41                                     processing_instance=self._biasProcessing,
42 samples_per_command=self.
43 _samples_trainig_command, save_path=save_path,
44 saved_dataset_path=saved_dataset_path,
45 real_data=True)
46
47     def app_run(self):
48         while True:
49             # Receive eeg data
50             signals = self._biasReception.get_real_data(channels=self.
51 _number_of_channels, n=self._n)
52
53             # Graph signals
54             for ch, signal in signals.items():
55                 t = np.arange(len(signals[ch])) / self._fs
56                 self._biasGraphing.graph_signal_voltage_time(t=t, signal=np.array(
57 signal), title="Signal {}".format(ch))
58
59             # Apply digital filtering
60             filtered_data = self._biasFilter.filter_signals(signals)
```

```

51         # Calculate the time vector
52         t = np.linspace(0, self._duration, self._n, endpoint=False)
53
54         # Graph signals
55         for ch, signal in filtered_data.items():
56             # Graph filtered signal
57             self._biasGraphing.graph_signal_voltage_time(t=t, signal=signal,
58 title="Filtered Signal {}".format(ch))
59
60         # Process data
61         times, eeg_signals = self._biasProcessing.process_signals(filtered_data)
62
63         # Plot 4 signals with its respective bands
64         for ch, signals in eeg_signals.items():
65             # Plot the interpolated signals
66             for band_name, sig in signals.items():
67                 self._biasGraphing.graph_signal_voltage_time(t=times[ch], signal=
68 sig, title=f"{band_name.capitalize()} interpolated. {ch}")
69
70         # Plot
71         self._biasGraphing.plot_now()
72
73         #command = self._biasAI.predict_command(eeg_data=eeg_signals)
74         #self._biasMotor.move_if_possible(command)

```

### 9.1.2. app.py

```

1 import numpy as np
2 from bias import BiasClass
3 from bias_reception import ReceptionBias
4 from bias_dsp import FilterBias, ProcessingBias
5 from bias_motors import MotorBias
6 from bias_ai import AIBias
7 from bias_graphing import GraphingBias
8
9 def main():
10     while True:
11         show_menu()
12         choice = input("Enter your choice: ")
13
14         if choice == '1':
15             pass
16
17         if choice == '2':
18             pass
19
20         if choice == '3':
21             run_motor_control()
22
23         if choice == '4':
24             # Define propiedades para la instancia de Bias
25             n = int(input("Enter number of data points: "))
26             fs = int(input("Enter sampling frequency: "))
27             number_of_channels = int(input("Enter number of channels: "))
28             port = '/dev/serial0'
29             baudrate = 115200
30             timeout = 1
31             # Crear objeto y correr la aplicaci n
32             biasInstance = BiasClass(n=n, fs=fs, channels=number_of_channels, port=
33 port, baudrate=baudrate, timeout=timeout)
34             biasInstance.app_run()
35
36         if choice == '6':
37             # Define propiedades para la instancia de Bias
38             n = int(input("Enter number of data points: "))
39             fs = int(input("Enter sampling frequency: "))
40             number_of_channels = int(input("Enter number of channels: "))
41             port = '/dev/serial0'

```

```

41         baudrate = 115200
42         timeout = 1
43         # Crear objeto y entrenar el modelo AI
44         biasInstance = BiasClass(n=n, fs=fs, channels=number_of_channels, port=
port, baudrate=baudrate, timeout=timeout)
45         saved_dataset_path = None
46         save_path = None
47
48         loading_dataset = input("Do you want to load an existing dataset? (y/n):
")
49         if loading_dataset.lower() == "y":
50             saved_dataset_path = input("Enter the name of the file where dataset
was saved (without extension): ")
51         else:
52             save_new_dataset = input("Do you want to save the new dataset? (y/n):
")
53             if save_new_dataset.lower() == "y":
54                 save_path = input("Enter the path where you want to save the
dataset (without extension): ")
55
56         biasInstance.train_ai_model(save_path, saved_dataset_path)
57
58 def show_menu():
59     print("EEG-based Wheelchair Control System")
60     print("1. Capture EEG Data with graphing in Terminal")
61     print("2. Capture and combine EEG")
62     print("3. Motor movement")
63     print("4. App run")
64     print("5. Extract Features")
65     print("6. Train Model")
66     print("7. Predict Action")
67     print("8. Exit")
68
69 def run_motor_control():
70     # Crear instancia de MotorBias y correr el control de motores
71     biasMotor = MotorBias(
72         echo_forward=18, trigger_forward=17,
73         echo_backwards=23, trigger_backwards=22,
74         echo_right=5, trigger_right=6,
75         echo_left=25, trigger_left=24,
76         led_forward=16, led_backwards=20,
77         led_left=21, led_right=26,
78         buzzer=12, motor1_in1=13,
79         motor1_in2=19, motor2_in1=7,
80         motor2_in2=8
81     )
82
83     while True:
84         command = input("Enter command (forward/left/backwards/right/stop): ").strip
()
85         biasMotor.move_if_possible(command)
86
87 if __name__ == "__main__":
88     main()

```

## 9.2. Inteligencia artificial

### 9.2.1. bias\_ai.py

```

1 from tensorflow.keras.models import Sequential # type: ignore
2 from tensorflow.keras.layers import Dense, Flatten, Conv1D, MaxPooling1D, Dropout,
    InputLayer, BatchNormalization # type: ignore
3 from tensorflow.keras.regularizers import l2 # type: ignore
4 from sklearn.model_selection import train_test_split
5 from sklearn.preprocessing import LabelBinarizer
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.decomposition import PCA
8 from bias_reception import ReceptionBias

```

```

9 from bias_dsp import FilterBias, ProcessingBias
10 from scipy.signal import welch
11 from scipy.stats import skew, kurtosis, entropy
12 from scipy.signal import cwt, morlet
13 import numpy as np
14 import random
15 import time
16
17 def main():
18     n = 1000
19     fs = 500
20     online = True
21     number_of_channels = 4
22     port = '/dev/serial0'
23     baudrate = 115200
24     timeout = 1
25     biasReception = ReceptionBias(port, baudrate, timeout)
26     biasFilter = FilterBias(n=n, fs=fs, notch=True, bandpass=True, fir=False, iir=False)
27     biasProcessing = ProcessingBias(n=n, fs=fs)
28     commands = ["forward", "backwards", "left", "right", "stop", "rest"]
29     biasAI = AIBias(n=n, fs=fs, channels=number_of_channels, commands=commands)
30     train = input("Do you want to train model? (y/n): ")
31     if train.lower() == "y":
32         saved_dataset_path = None
33         save_path = None
34         loading_dataset = input("Do you want to load a existent dataset? (y/n): ")
35         if loading_dataset.lower() == "y":
36             saved_dataset_path = input("Write the name of the file where dataset was saved: ")
37         else:
38             save_new_dataset = input("Do you want to save the new dataset? (y/n): ")
39             if save_new_dataset == "y":
40                 save_path = input("Write the path where you want to save the dataset: ")
41             biasAI.collect_and_train(reception_instance=biasReception, filter_instance=biasFilter, processing_instance=biasProcessing,
42                                     trials_per_command=1, save_path=save_path,
43                                     saved_dataset_path=saved_dataset_path, real_data=False)
44             # Generate synthetic data
45             signals = generate_synthetic_eeg(n_samples=n, n_channels=number_of_channels, fs=fs, command="left")
46             #signals = biasReception.get_real_data(channels=number_of_channels, n=n)
47
48             filtered_data = biasFilter.filter_signals(signals)
49             # Process data
50             times, eeg_signals = biasProcessing.process_signals(filtered_data)
51             predicted_command = biasAI.predict_command(eeg_data=eeg_signals)
52             print(f"Predicted Command: {predicted_command}")
53
54 class AIBias:
55     def __init__(self, n, fs, channels, commands):
56         self._n = n
57         self._fs = fs
58         self._number_of_channels = channels
59         self._features_length = len(["mean", "variance", "skewness", "kurt", "energy",
60                                     "band_power", "wavelet_energy", "entropy"])
61         self._number_of_waves_per_channel = len(["alpha", "beta", "gamma", "delta", "theta"])
62         self._num_features_per_channel = self._features_length * self._number_of_waves_per_channel
63         self._commands = commands
64         self._model = self.build_model(output_dimension=len(self._commands))
65         self._is_trained = False
66         self._pca = PCA(n_components=0.95) # Retain 95% of variance
67         self._scaler = StandardScaler()

```

```

68         # Create a dynamic label map based on the provided commands
69         self._label_map = {command: idx for idx, command in enumerate(self._commands)}
70     }
71     self._reverse_label_map = {idx: command for command, idx in self._label_map.items()}
72
73     # Define getter
74     def ai_is_trained(self):
75         return self._is_trained
76
77     def collect_and_train(self, reception_instance, filter_instance,
78                           processing_instance, trials_per_command,
79                           save_path=None, saved_dataset_path=None, real_data=True):
80         """
81         Collects EEG data, extracts features, and trains the model.
82         """
83         X = []
84         y = []
85
86         if saved_dataset_path is None:
87             for trial in range(trials_per_command):
88                 for command in self._commands:
89                     # Get real data or generate synthetic data
90                     if real_data:
91                         print(f"Think about {command}. Trial: {trial}")
92                         signals = reception_instance.get_real_data(channels=self._number_of_channels, n=self._n)
93                     else:
94                         print(f"Trial: {trial}")
95                         signals = generate_synthetic_eeg(n_samples=self._n,
96                                                         n_channels=self._number_of_channels, fs=self._fs, command=command)
97
98                     filtered_data = filter_instance.filter_signals(signals)
99                     _, eeg_signals = processing_instance.process_signals(
100                         filtered_data)
101
102                     # Extract features and append to X
103                     features = self.extract_features(eeg_signals)
104
105                     X.append(features)
106                     y.append(self._label_map[command])
107
108                     if real_data:
109                         time.sleep(1)
110
111                 if real_data:
112                     print("Changing command. Be ready")
113                     time.sleep(20)
114
115         # Convert X and y to numpy arrays
116         X = np.array(X)
117         y = np.array(y)
118
119         if save_path:
120             # Save the dataset as a compressed NumPy file
121             np.savez_compressed(f"{save_path}.npz", X=X, y=y)
122             print(f"Dataset saved to {save_path}.npz")
123
124         else:
125             data = np.load(f"{saved_dataset_path}.npz")
126             X, y = data['X'], data['y']
127
128         # Convert y to one-hot encoding
129         lb = LabelBinarizer()
130         y = lb.fit_transform(y)
131
132         # Train the model with the collected data

```

```

130         self.train_model(X, y)
131
132     def build_model(self, output_dimension):
133         model = Sequential([
134             InputLayer(shape=(self._number_of_channels, self.
135 _num_features_per_channel)), # Adjusted input shape to match the feature count
136             Conv1D(filters=64, kernel_size=3, activation='relu'),
137             #BatchNormalization(),
138             MaxPooling1D(pool_size=1),
139             Dropout(0.5),
140             Flatten(),
141             Dense(100, activation='relu'), #, kernel_regularizer=l2(0.01)),
142             #BatchNormalization(),
143             Dropout(0.5),
144             Dense(50, activation='relu'), #, kernel_regularizer=l2(0.01)),
145             #BatchNormalization(),
146             Dropout(0.5),
147             Dense(output_dimension, activation='softmax') # 6 output classes (
148 forward, backward, etc.)
149         ])
150         model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
151 accuracy'])
152         return model
153
154     def extract_features(self, eeg_data):
155         features = []
156         # Iterate over each channel in eeg_data
157         for ch, signals_per_channel in eeg_data.items():
158             channel_features = []
159             assert(len(signals_per_channel) == self._number_of_waves_per_channel)
160             # Iterate over the signals of each channel
161             for band_name, signal_wave in signals_per_channel.items():
162                 signal_wave = np.array(signal_wave)
163
164                 # Statistical Features
165                 mean = np.mean(signal_wave)
166                 variance = np.var(signal_wave)
167                 skewness = skew(signal_wave)
168                 kurt = kurtosis(signal_wave)
169                 energy = np.sum(signal_wave ** 2)
170
171                 # Frequency Domain Features (Power Spectral Density)
172                 freqs, psd = welch(signal_wave, fs=self._fs) # Assuming fs = 500 Hz
173
174                 # Band Power
175                 band_power = np.sum(psd) # Total power within this band
176
177                 # Use scipy.signal.cwt instead of pywt
178                 scales = np.arange(1, 31)
179                 coeffs = cwt(signal_wave, morlet, scales)
180                 wavelet_energy = np.sum(coeffs ** 2)
181
182                 # Entropy
183                 signal_entropy = entropy(np.histogram(signal_wave, bins=10)[0])
184                 list_of_features = [mean, variance, skewness, kurt, energy,
185 band_power, wavelet_energy, signal_entropy]
186
187                 # Append all features together
188                 channel_features.extend(list_of_features)
189
190             assert(len(list_of_features) == self._features_length)
191
192             features.append(channel_features)
193
194         features = np.abs(np.array(features))
195         features = self._scaler.fit_transform(features) # Normalize
196         # Perform PCA if needed, currently commented out
197         # features = self._pca.fit_transform(features) # Dimensionality Reduction

```

```

194
195     # Adjust reshaping based on actual size
196     # Get the total number of features per channel
197     num_features_per_channel = features.shape[1]
198     assert(self._num_features_per_channel == num_features_per_channel)
199     # Reshape based on the number of samples, channels, and features
200     expected_shape = (self._number_of_channels, self._num_features_per_channel)
201     features = features.reshape(expected_shape)
202     return features
203
204 def train_model(self, X, y):
205     # X_processed = np.array([self.extract_features(epoch) for epoch in X])
206     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
207 random_state=42)
208     self._model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(
209 X_test, y_test))
210     self._is_trained = True
211
212 def predict_command(self, eeg_data):
213     if not self._is_trained:
214         raise Exception("Model has not been trained yet.")
215
216     # Extract features from the EEG data
217     features = self.extract_features(eeg_data)
218
219     # Ensure the features have the correct shape (1, number_of_channels,
220 number_of_features)
221     features = features.reshape(1, self._number_of_channels, self.
222 _num_features_per_channel)
223
224     # Make prediction
225     prediction = self._model.predict(features)
226
227     # Get the predicted label index
228     predicted_label_index = np.argmax(prediction, axis=1)[0]
229
230     # Convert the numerical prediction to the text label
231     predicted_command = self._reverse_label_map[predicted_label_index]
232
233     return predicted_command
234
235 def generate_synthetic_eeg(n_samples, n_channels, fs, command=None):
236     """
237     Generate synthetic raw EEG data for multiple channels.
238     The output is a dictionary where each channel has 1000 raw samples.
239     Simulate different tasks by altering the signal patterns.
240     """
241     t = np.linspace(0, n_samples/fs, n_samples, endpoint=False)
242     data = {}
243
244     for ch in range(n_channels):
245         # Simulate different frequency bands with some basic correlations
246         base_alpha = np.sin(2 * np.pi * 10 * t) # Alpha signal_wave (10 Hz)
247         base_beta = np.sin(2 * np.pi * 20 * t) # Beta signal_wave (20 Hz)
248         base_theta = np.sin(2 * np.pi * 6 * t) # Theta signal_wave (6 Hz)
249         base_delta = np.sin(2 * np.pi * 2 * t) # Delta signal_wave (2 Hz)
250         base_gamma = np.sin(2 * np.pi * 40 * t) # Gamma signal_wave (40 Hz)
251
252         alpha_power = 1.0
253         beta_power = 1.0
254         theta_power = 1.0
255         delta_power = 1.0
256         gamma_power = 1.0 # Adjust signal based on the command
257
258         if command == "forward":
259             alpha_power = 1.5
260             beta_power = 0.5
261         elif command == "backward":

```

```

258         alpha_power = 0.5
259         beta_power = 1.5
260     elif command == "left":
261         theta_power = 1.5
262         delta_power = 0.5
263     elif command == "right":
264         theta_power = 0.5
265         delta_power = 1.5
266     elif command == "stop":
267         alpha_power = 0.2
268         beta_power = 0.2
269         gamma_power = 0.2
270     else: # rest
271         alpha_power = 1.0
272         beta_power = 1.0
273         theta_power = 1.0
274         delta_power = 1.0
275         gamma_power = 1.0
276
277     # Generate signal with some added randomness and correlations
278     signal = (
279         alpha_power * base_alpha +
280         beta_power * base_beta +
281         theta_power * base_theta +
282         delta_power * base_delta +
283         gamma_power * base_gamma
284     )
285
286     # Add channel correlation (e.g., 10% of the previous channels signal)
287     if ch > 0:
288         signal += 0.1 * data[ch-1]
289
290     # Add random noise to simulate realistic EEG signals
291     noise = np.random.normal(0, 0.1, size=t.shape)
292     signal += noise
293
294     # Store the raw signal in the dictionary
295     data[ch] = signal
296
297     return data
298
299 '''
300 def generate_synthetic_eeg(n_samples, n_channels, fs):
301     """
302     Generate synthetic raw EEG data for multiple channels.
303     The output is a dictionary where each channel has 1000 raw samples.
304     """
305     t = np.linspace(0, n_samples/fs, n_samples, endpoint=False)
306     data = {}
307
308     for ch in range(n_channels):
309         # Create a raw EEG signal by summing several sine waves to simulate brain
310         # activity
311         signal = (
312             random.randrange(0, 10) * np.sin(2 * np.pi * random.randrange(8, 13) * t)
313             + # Simulate alpha signal_wave (8-13 Hz)
314             random.randrange(0, 10) * np.sin(2 * np.pi * random.randrange(13, 30) * t)
315             ) + # Simulate beta signal_wave (13-30 Hz)
316             random.randrange(0, 10) * np.sin(2 * np.pi * random.randrange(4, 8) * t)
317             + # Simulate theta signal_wave (4-8 Hz)
318             random.randrange(0, 10) * np.sin(2 * np.pi * random.randrange(1, 4) * t)
319             + # Simulate delta signal_wave (0.5-4 Hz)
320             random.randrange(0, 10) * np.sin(2 * np.pi * random.randrange(0, 50) * t)
321             # Simulate gamma signal_wave (30-100 Hz)
322         )
323
324     # Add random noise to simulate realistic EEG signals
325     noise = np.random.normal(0, 0.5, size=t.shape)

```



```

320     signal += noise
321
322     # Store the raw signal in the dictionary
323     data[ch] = signal
324
325     return data
326 '''
327
328 if __name__ == "__main__":
329     main()

```

## 9.3. Filtrado y procesamiento de señales

### 9.3.1. bias\_dsp.py

### 9.3.2. bias\_dsp\_task.py

### 9.3.3. bias\_graphing.py

```

1 import matplotlib.pyplot as plt
2 import plotext
3
4 class GraphingBias:
5     # Constructor
6     def __init__(self, graph_in_terminal):
7         self._graph_in_terminal = graph_in_terminal
8
9     # Graph signal in function of time
10    def graph_signal_voltage_time(self, t, signal, title):
11        if not self._graph_in_terminal:
12            # Plot given signal in th time domain
13            plt.figure(figsize=(12, 6))
14            if signal.ndim == 1:
15                plt.plot(t, signal)
16            else:
17                for i in range(signal.shape[0]):
18                    plt.plot(t, signal[i])
19            # Set graph paramters
20            plt.title(title)
21            plt.xlabel("Time [s]")
22            plt.ylabel("Magnitude")
23            plt.grid()
24
25        else:
26            plotext.clear_data()
27            plotext.plot(t, signal)
28            # Set graph parameters
29            plotext.title(title)
30            plotext.xlabel("Time [s]")
31            plotext.ylabel("Magnitude")
32            plotext.show()
33
34    def graph_signal_voltage_frequency(self, frequencies, magnitudes, title):
35        if not self._graph_in_terminal:
36            # Plot given signal in the frquency domain
37            plt.figure(figsize=(12, 6))
38            plt.plot(frequencies, magnitudes)
39            # Set graph parameters
40            plt.title(title)
41            plt.xlabel("Frequency [Hz]")
42            plt.ylabel("Magnitude")
43            plt.grid()
44
45        else:
46            plotext.clear_data()
47            plotext.plot(frequencies, magnitudes)
48            # Set graph parameters
49            plotext.title(title)

```

```

50         plottext.xlabel("Frequency [Hz]")
51         plottext.ylabel("Magnitude")
52         plottext.show()
53
54     def plot_now(self):
55         if not self._graph_in_terminal:
56             plt.tight_layout()
57             plt.show()
58         else:
59             pass
60
61     '''
62 #import matplotlib
63
64 #matplotlib.use('Agg') # Use the non-GUI backend
65
66 #import matplotlib.pyplot as plt
67 import plottext as plt
68
69 def graph_signal_voltage_time(t, signal, title, filename):
70     # Plot given signal in th time domain
71     #plt.figure(figsize=(12, 6))
72     if signal.ndim == 1:
73         plt.plot(t, signal)
74     else:
75         for i in range(signal.shape[0]):
76             plt.plot(t, signal[i])
77     plt.title(title)
78     plt.xlabel("Time [s]")
79     plt.ylabel("Magnitude")
80     plt.grid()
81     #plt.savefig(filename) # Save the plot as an image
82     #plt.close() # Close the plot to free memory
83     #print(f"Plot saved to {filename}")
84
85 def graph_signal_voltage_frequency(frequencies, magnitudes, title):
86     # Plot given signal in the frquency domain
87     #plt.figure(figsize=(12, 6))
88     plt.plot(frequencies, magnitudes)
89     plt.title(title)
90     plt.xlabel("Frequency [Hz]")
91     plt.ylabel("Magnitude")
92     plt.grid()
93
94 def plot_now():
95     plt.tight_layout()
96     plt.show()
97 '''

```

## 9.4. Motores

### 9.4.1. bias\_motors.py

```

1 from gpiozero import DistanceSensor, PWMLED, Buzzer, PWMOutputDevice
2 from gpiozero.pins.pigpio import PiGPIOFactory
3 import time
4
5 def main():
6     # Define motor instance
7     biasMotor = MotorBias(echo_forward=18, trigger_forward=17, echo_backwards=23,
8         trigger_backwards=22, echo_right=5, trigger_right=6,
9         echo_left=25, trigger_left=24, led_forward=16,
10        led_backwards=20, led_left=21, led_right=26, buzzer=12, motor1_in1=13,
11        motor1_in2=19, motor2_in1=7, motor2_in2=8)
12
13     while True:
14         # Get command
15         command = input("Enter command (forward/left/backwards/right/stop): ").strip()
16
17         ()

```

```

13         biasMotor.move_if_possible(command)
14
15 class MotorBias:
16     def __init__(self, echo_forward, trigger_forward, echo_backwards,
17         trigger_backwards, echo_right, trigger_right,
18         echo_left, trigger_left, led_forward, led_backwards, led_left,
19         led_right, buzzer, motor1_in1,
20         motor1_in2, motor2_in1, motor2_in2):
21
22         # Configurar pin factory in order to use pigpio
23         factory = PiGPIOFactory()
24
25         # Configure ultrasonic sensors and LEDs
26         self._ultrasonic_forward = DistanceSensor(echo=echo_forward, trigger=
27         trigger_forward, pin_factory=factory)
28         self._ultrasonic_backwards = DistanceSensor(echo=echo_backwards, trigger=
29         trigger_backwards, pin_factory=factory)
30         self._ultrasonic_right = DistanceSensor(echo=echo_right, trigger=
31         trigger_right, pin_factory=factory)
32         self._ultrasonic_left = DistanceSensor(echo=echo_left, trigger=trigger_left,
33         pin_factory=factory)
34
35         # Configure LEDs
36         self._led_forward = PWMLED(led_forward, pin_factory=factory)
37         self._led_backwards = PWMLED(led_backwards, pin_factory=factory)
38         self._led_left = PWMLED(led_left, pin_factory=factory)
39         self._led_right = PWMLED(led_right, pin_factory=factory)
40
41         # Configure buzzer
42         self._buzzer = Buzzer(buzzer)
43
44         # GPIO Pin setup for Motor 1
45         self._motor1_in1 = PWMOutputDevice(motor1_in1, frequency=50, pin_factory=
46         factory)
47         self._motor1_in2 = PWMOutputDevice(motor1_in2, frequency=50, pin_factory=
48         factory)
49
50         # GPIO Pin setup for Motor 2
51         self._motor2_in1 = PWMOutputDevice(motor2_in1, frequency=50, pin_factory=
52         factory)
53         self._motor2_in2 = PWMOutputDevice(motor2_in2, frequency=50, pin_factory=
54         factory)
55
56     def move_if_possible(self, command):
57         try:
58             # Move forward
59             if command == "forward":
60                 distance = self._ultrasonic_forward.distance * 100
61                 # Maximum distance of 20 cm
62                 if distance < 20:
63                     # Forward is blocked
64                     self._led_forward.on()
65                     self._buzzer.on()
66                     print(f"Obstacle forward: {distance:.1f} cm. Blocked movement.")
67                 else:
68                     # Do the movement
69                     self._led_forward.off()
70                     self._buzzer.off()
71                     self.move_forward(25)
72             # Move backwards
73             elif command == "backwards":
74                 distance = self._ultrasonic_backwards.distance * 100
75                 # Maximum distance of 20 cm
76                 if distance < 20:
77                     # Backwards is blocked
78                     self._led_backwards.on()
79                     self._buzzer.on()
80                     print(f"Obstacle backwards: {distance:.1f} cm. Blocked movement.")

```

```

71         )
72         else:
73             # Do the movement
74             self._led_backwards.off()
75             self._buzzer.off()
76             self.move_backward(25)
77         # Turn left
78         elif command == "left":
79             distance = self._ultrasonic_left.distance * 100
80             # Maximum distance of 20 cm
81             if distance < 20:
82                 # Left is blocked
83                 self._led_left.on()
84                 self._buzzer.on()
85                 print(f"Obstacle on the left: {distance:.1f} cm. Blocked movement
86             ")
87         else:
88             # Do the movement
89             self._led_left.off()
90             self._buzzer.off()
91             self.turn_left(25)
92         # Turn right
93         elif command == "right":
94             distance = self._ultrasonic_right.distance * 100
95             # Maximum distance of 20 cm
96             if distance < 20:
97                 # Right is blocked
98                 self._led_right.on()
99                 self._buzzer.on()
100                 print(f"Obstacle on the right: {distance:.1f} cm. Blocked
101             movement.")
102         else:
103             # Do the movement
104             self._led_right.off()
105             self._buzzer.off()
106             self.turn_right(25)
107         # Brake
108         elif command == "stop":
109             # Make all parameters off
110             self.brake()
111             self._led_forward.off()
112             self._led_backwards.off()
113             self._led_left.off()
114             self._led_right.off()
115             self._buzzer.off()
116         else:
117             print("Invalid command")
118
119         time.sleep(1)
120         self.brake() # Stop after each command
121         self._led_forward.off()
122         self._led_backwards.off()
123         self._led_left.off()
124         self._led_right.off()
125         self._buzzer.off()
126
127     except KeyboardInterrupt:
128         print("Program stopped by user")
129
130 # Configure speed of motor depending on PWM
131 def set_motor_speed(self, motor_in1, motor_in2, speed, invert=False):
132     # Define positive speed
133     if speed > 0:
134         motor_in1.value = ((100.0 - speed) / 100.0) if invert else speed / 100.0
135         motor_in2.value = 0
136     # Define negative speed
137     elif speed < 0:
138         motor_in1.value = 0

```

```

136         motor_in2.value = ((100.0 - abs(speed)) / 100.0) if invert else abs(speed
) / 100.0
137     # If it's zero brake
138     else:
139         motor_in1.value = 0
140         motor_in2.value = 0
141
142     # Move wheelchair forward
143     def move_forward(self, speed):
144         self.set_motor_speed(self._motor1_in1, self._motor1_in2, speed, invert=True)
145         self.set_motor_speed(self._motor2_in1, self._motor2_in2, speed, invert=True)
146
147     # Move wheelchair backwards
148     def move_backward(self, speed):
149         self.set_motor_speed(self._motor1_in1, self._motor1_in2, -speed, invert=True)
150         self.set_motor_speed(self._motor2_in1, self._motor2_in2, -speed, invert=True)
151
152     # Turn wheelchair left
153     def turn_left(self, speed):
154         self.set_motor_speed(self._motor1_in1, self._motor1_in2, -speed, invert=True)
155         self.set_motor_speed(self._motor2_in1, self._motor2_in2, speed, invert=True)
156
157     # Turn wheelchair right
158     def turn_right(self, speed):
159         self.set_motor_speed(self._motor1_in1, self._motor1_in2, speed, invert=True)
160         self.set_motor_speed(self._motor2_in1, self._motor2_in2, -speed, invert=True)
161
162     # Brake wheelchair
163     def brake(self):
164         self.set_motor_speed(self._motor1_in1, self._motor1_in2, 0, invert=True)
165         self.set_motor_speed(self._motor2_in1, self._motor2_in2, 0, invert=True)
166
167 if __name__ == "__main__":
168     main()

```

## 9.5. Recepción de señales

### 9.5.1. bias\_reception.py

```

1 import serial
2 import time
3 import numpy as np
4 import json
5 from bias_graphing import GraphingBias
6 from signals import random_signal
7
8 def main():
9     # Set constants
10     n = 1000
11     fs = 500
12     number_of_channels = 4
13     # Receive data
14     biasReception = ReceptionBias()
15     signals = {}
16     signals = biasReception.get_real_data(channels=number_of_channels, n=n)
17
18     # Graph signals
19     biasGraphing = GraphingBias(graph_in_terminal=True)
20     for ch, signal in signals.items():
21         t = np.arange(len(signals[ch])) / fs
22         biasGraphing.graph_signal_voltage_time(t=t, signal=np.array(signal), title="
Signal {}".format(ch))
23
24 class ReceptionBias:
25     # Constructor
26     def __init__(self, port='/dev/serial0', baudrate=115200, timeout=1):
27         self._port = port
28         self._baudrate = baudrate

```

```

29     self._timeout = timeout
30
31     # Get the data from the RP2040 Zero
32     def get_real_data(self, channels, n):
33         # Initialize serial communication
34         self._ser = self.init_serial(self._port, self._baudrate, self._timeout)
35         try:
36             # Capture the signal from the UART
37             real_eeg_signals = self.capture_signals(channels=channels, n=n)
38             return real_eeg_signals
39         finally:
40             self._ser.close()
41
42     def capture_signals(self, channels, n):
43         # Initialize variables
44         signals = {f'ch{ch}': [] for ch in range(channels)}
45         start_time = time.time()
46         # Loop until we have enough samples
47         while len(signals[f'ch3']) < n:
48             if self._ser.in_waiting > 0:
49                 try:
50                     # Read one line to detect \n character
51                     data = self._ser.readline().decode('utf-8').strip()
52                     eeg_data = self.process_data(data)
53                     # Make an array with the data
54                     if eeg_data:
55                         for ch in range(channels):
56                             signals[f'ch{ch}'].extend(eeg_data[f'ch{ch}'])
57                 except Exception as e:
58                     print("Can't be decoded")
59
60         # Check the time it takes to read
61         elapsed_time = time.time() - start_time
62         print(f"elapsed time: {elapsed_time}")
63         # Ensure all signals have the correct length
64         for ch in range(channels):
65             signals[f'ch{ch}'] = signals[f'ch{ch}'][:n]
66
67         return signals
68
69     '''
70
71     def combine_signals(self, signals):
72         combined_signal = np.mean([signals[f'ch{ch}'] for ch in range(len(signals))],
73                                   axis=0)
74         return combined_signal
75     '''
76
77     # Initialize serial communication
78     def init_serial(self, port, baudrate, timeout):
79         return serial.Serial(port, baudrate, timeout=timeout)
80
81     # Load JSON data
82     def process_data(self, data):
83         try:
84             json_data = json.loads(data)
85             print(json_data)
86             return json_data
87         except json.JSONDecodeError as e:
88             print(f"Error decoding JSON: {e}")
89             return None
90
91 if __name__ == "__main__":
92     main()

```

### 9.5.2. CMakeLists.txt

```

1 cmake_minimum_required(VERSION 3.13)

```

```

2 include(pico_sdk_import.cmake)
3 project(reception)
4 pico_sdk_init()
5 add_executable(reception reception.c)
6 target_link_libraries(reception pico_stdlib hardware_adc hardware_uart)
7 pico_enable_stdio_usb(reception 1)
8 pico_add_extra_outputs(reception)

```

### 9.5.3. pico\_sdk\_import.cmake

```

1 # This is a copy of <PICO_SDK_PATH>/external/pico_sdk_import.cmake
2
3 # This can be dropped into an external project to help locate this SDK
4 # It should be include()ed prior to project()
5
6 if (DEFINED ENV{PICO_SDK_PATH} AND (NOT PICO_SDK_PATH))
7     set(PICO_SDK_PATH $ENV{PICO_SDK_PATH})
8     message("Using PICO_SDK_PATH from environment ('${PICO_SDK_PATH}')")
9 endif ()
10
11 if (DEFINED ENV{PICO_SDK_FETCH_FROM_GIT} AND (NOT PICO_SDK_FETCH_FROM_GIT))
12     set(PICO_SDK_FETCH_FROM_GIT $ENV{PICO_SDK_FETCH_FROM_GIT})
13     message("Using PICO_SDK_FETCH_FROM_GIT from environment ('${PICO_SDK_FETCH_FROM_GIT}')")
14 endif ()
15
16 if (DEFINED ENV{PICO_SDK_FETCH_FROM_GIT_PATH} AND (NOT PICO_SDK_FETCH_FROM_GIT_PATH))
17     set(PICO_SDK_FETCH_FROM_GIT_PATH $ENV{PICO_SDK_FETCH_FROM_GIT_PATH})
18     message("Using PICO_SDK_FETCH_FROM_GIT_PATH from environment ('${PICO_SDK_FETCH_FROM_GIT_PATH}')")
19 endif ()
20
21 set(PICO_SDK_PATH "${PICO_SDK_PATH}" CACHE PATH "Path to the Raspberry Pi Pico SDK")
22 set(PICO_SDK_FETCH_FROM_GIT "${PICO_SDK_FETCH_FROM_GIT}" CACHE BOOL "Set to ON to
23     fetch copy of SDK from git if not otherwise locatable")
24 set(PICO_SDK_FETCH_FROM_GIT_PATH "${PICO_SDK_FETCH_FROM_GIT_PATH}" CACHE FILEPATH "
25     location to download SDK")
26
27 if (NOT PICO_SDK_PATH)
28     if (PICO_SDK_FETCH_FROM_GIT)
29         include(FetchContent)
30         set(FETCHCONTENT_BASE_DIR_SAVE ${FETCHCONTENT_BASE_DIR})
31         if (PICO_SDK_FETCH_FROM_GIT_PATH)
32             get_filename_component(FETCHCONTENT_BASE_DIR "${PICO_SDK_FETCH_FROM_GIT_PATH}" REALPATH BASE_DIR "${CMAKE_SOURCE_DIR}")
33         endif ()
34         # GIT_SUBMODULES_RECURSE was added in 3.17
35         if (${CMAKE_VERSION} VERSION_GREATER_EQUAL "3.17.0")
36             FetchContent_Declare(
37                 pico_sdk
38                 GIT_REPOSITORY https://github.com/raspberrypi/pico-sdk
39                 GIT_TAG master
40                 GIT_SUBMODULES_RECURSE FALSE
41             )
42         else ()
43             FetchContent_Declare(
44                 pico_sdk
45                 GIT_REPOSITORY https://github.com/raspberrypi/pico-sdk
46                 GIT_TAG master
47             )
48         endif ()
49
50         if (NOT pico_sdk)
51             message("Downloading Raspberry Pi Pico SDK")
52             FetchContent_Populate(pico_sdk)
53             set(PICO_SDK_PATH ${pico_sdk_SOURCE_DIR})
54         endif ()
55         set(FETCHCONTENT_BASE_DIR ${FETCHCONTENT_BASE_DIR_SAVE})
56     endif ()
57
58 if (NOT PICO_SDK_PATH)
59     message("Downloading Raspberry Pi Pico SDK")
60     FetchContent_Populate(pico_sdk)
61     set(PICO_SDK_PATH ${pico_sdk_SOURCE_DIR})
62 endif ()
63 set(FETCHCONTENT_BASE_DIR ${FETCHCONTENT_BASE_DIR_SAVE})

```

```

54     else ()
55         message(FATAL_ERROR
56             "SDK location was not specified. Please set PICO_SDK_PATH or set
57             PICO_SDK_FETCH_FROM_GIT to on to fetch from git."
58         )
59     endif ()
60 endif ()
61 get_filename_component(PICO_SDK_PATH "${PICO_SDK_PATH}" REALPATH BASE_DIR "${{
62     CMAKE_BINARY_DIR}}")
63 if (NOT EXISTS ${PICO_SDK_PATH})
64     message(FATAL_ERROR "Directory '${PICO_SDK_PATH}' not found")
65 endif ()
66 set(PICO_SDK_INIT_CMAKE_FILE ${PICO_SDK_PATH}/pico_sdk_init.cmake)
67 if (NOT EXISTS ${PICO_SDK_INIT_CMAKE_FILE})
68     message(FATAL_ERROR "Directory '${PICO_SDK_PATH}' does not appear to contain the
69     Raspberry Pi Pico SDK")
70 endif ()
71 set(PICO_SDK_PATH ${PICO_SDK_PATH} CACHE PATH "Path to the Raspberry Pi Pico SDK"
72     FORCE)
73 include(${PICO_SDK_INIT_CMAKE_FILE})

```

#### 9.5.4. reception.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>
5  #include "pico/stdlib.h"
6  #include "hardware/uart.h"
7  #include "hardware/adc.h"
8
9  #define UART_ID uart0
10 #define BAUDRATE 115200
11 #define NUMBER_OF_CHANNELS 4
12 #define NUMBER_OF_TOTAL_SAMPLES 1000
13 #define ADC_DELAY_US -2000
14
15 // Value for conversion
16 const float CONVERSION_FACTOR = 3.3f * 1000 / (1 << 12);
17
18 // Array to store values of the samples
19 uint16_t values_mv[NUMBER_OF_CHANNELS][NUMBER_OF_TOTAL_SAMPLES];
20 // Flag of sampling done
21 static bool sampling_done = false;
22
23 // Function prototypes
24 void init_uart(uint8_t tx_pin, uint8_t rx_pin);
25 void init_adc(uint8_t adc_channel_0, uint8_t adc_channel_1, uint8_t adc_channel_2,
26     uint8_t adc_channel_3);
27 bool read_adc(struct repeating_timer *t);
28 void start_sampling(void);
29 void build_json(char *data, uint total_bytes);
30 void send_data(char *data);
31
32 int main(void) {
33     // Pins for ADC
34     const uint8_t ADC_PIN_CHANNEL_0 = 26;
35     const uint8_t ADC_PIN_CHANNEL_1 = 27;
36     const uint8_t ADC_PIN_CHANNEL_2 = 28;
37     const uint8_t ADC_PIN_CHANNEL_3 = 29;
38
39     // Pins for UART
40     const uint8_t UART_TX_PIN = 0;
41     const uint8_t UART_RX_PIN = 1;

```



```

41
42 // Amount of bytes to send
43 uint TOTAL_BYTES_TO_SEND = sizeof("{}\n") // Considering null character
44                             + (sizeof("\ch0\":[],") - sizeof("")) *
45                             NUMBER_OF_CHANNELS // Excluding null character
46                             + (sizeof("0000,") - sizeof("")) *
47                             NUMBER_OF_TOTAL_SAMPLES * NUMBER_OF_CHANNELS // Excluding null characters
48                             - (NUMBER_OF_CHANNELS + 1); // Excluding commas
49
50 // JSON data to send
51 char data_to_send[TOTAL_BYTES_TO_SEND];
52
53 stdio_init_all();
54 // Initialize UART
55 init_uart(UART_TX_PIN, UART_RX_PIN);
56 // Initialize ADC
57 init_adc(ADC_PIN_CHANNEL_0, ADC_PIN_CHANNEL_1, ADC_PIN_CHANNEL_2,
58          ADC_PIN_CHANNEL_3);
59 // Start the sampling
60 start_sampling();
61
62 while (true) {
63     if (sampling_done) {
64         //printf("Sampling done\n");
65         // Make the JSON to send it
66         build_json(data_to_send, TOTAL_BYTES_TO_SEND);
67         //printf("JSON: %s\n", data_to_send);
68         // Send the JSON
69         send_data(data_to_send);
70         // Clear sampling flag and restart sampling
71         sampling_done = false;
72         start_sampling();
73     }
74 }
75
76 return 0;
77 }
78
79 void init_uart(uint8_t tx_pin, uint8_t rx_pin) {
80     // UART setup
81     uart_init(UART_ID, BAUDRATE);
82     gpio_set_function(tx_pin, GPIO_FUNC_UART);
83     gpio_set_function(rx_pin, GPIO_FUNC_UART);
84     uart_set_format(UART_ID, 8, 1, UART_PARITY_NONE);
85     uart_set_fifo_enabled(UART_ID, true);
86 }
87
88 void init_adc(uint8_t adc_channel_0, uint8_t adc_channel_1, uint8_t adc_channel_2,
89              uint8_t adc_channel_3) {
90     adc_init();
91     // Initialize ADC channels
92     adc_gpio_init(adc_channel_0);
93     adc_gpio_init(adc_channel_1);
94     adc_gpio_init(adc_channel_2);
95     adc_gpio_init(adc_channel_3);
96 }
97
98 // Callback of the timer
99 bool read_adc(struct repeating_timer *t) {
100     // Counter of sampling
101     static uint sampling_count = 0;
102
103     // Verify that the count doesn't exceed the total number of samples
104     if (sampling_count >= NUMBER_OF_TOTAL_SAMPLES) {
105         // Set the values after having completed the sampling
106         sampling_done = true;
107         cancel_repeating_timer(t);
108         sampling_count = 0;
109     } else {

```

```

105     for (int channel = 0; channel < NUMBER_OF_CHANNELS; channel++) {
106         // Read ADC channel
107         adc_select_input(channel);
108         uint16_t adc_value = adc_read();
109         // Convert it to mV
110         values_mv[channel][sampling_count] = round(adc_value * CONVERSION_FACTOR)
111     };
112     // Next sample
113     sampling_count++;
114 }
115 return true;
116 }
117
118 void start_sampling(void) {
119     // Start timer for sampling
120     static struct repeating_timer timer;
121     sampling_done = false;
122     // Set the read_adc function as callback with sample delay of 2000 us in order to
123     // have 500 Hz of sampling frequency
124     add_repeating_timer_us(ADC_DELAY_US, read_adc, NULL, &timer);
125 }
126
127 // Function which transforms the adc_data to a JSON
128 void build_json(char *data, uint total_bytes) {
129     // Stores the total number of bytes for the JSON
130     char *str = (char*) malloc(total_bytes);
131     if (str == NULL) {
132         printf("Memory allocation failed\n");
133         return;
134     }
135     strcpy(str, "{");
136     // Print each value of the ADC channel until completing all channels with all the
137     // samplings
138     for (int channel = 0; channel < NUMBER_OF_CHANNELS; channel++) {
139         // Create the channel label
140         char label[8];
141         sprintf(label, "\"ch%d\":[" , channel);
142
143         // Concatenate the channel label to the JSON string
144         strcat(str, label);
145
146         // Create an auxiliary array
147         char aux[sizeof("0000,")];
148
149         // Print the values for the channel array
150         for (int sampling_number = 0; sampling_number < NUMBER_OF_TOTAL_SAMPLES;
151             sampling_number++) {
152             if (sampling_number < NUMBER_OF_TOTAL_SAMPLES - 1) {
153                 sprintf(aux, "%d,", values_mv[channel][sampling_number]);
154             } else {
155                 sprintf(aux, "%d", values_mv[channel][sampling_number]);
156             }
157             strcat(str, aux);
158         }
159
160         if (channel < NUMBER_OF_CHANNELS - 1) {
161             strcat(str, "],");
162         } else {
163             strcat(str, "]}\\n");
164         }
165     }
166     strcpy(data, str);
167     free(str);
168 }
169
170 // Send data by UART

```

```

169 void send_data(char *data) {
170     uart_puts(UART_ID, data);
171 }
172
173
174 /*
175 // This code works
176 #include <stdio.h>
177 #include <stdlib.h>
178 #include <string.h>
179 #include <math.h>
180 #include "pico/stdlib.h"
181 #include "hardware/uart.h"
182 #include "hardware/adc.h"
183
184 #define UART_ID uart0
185 #define BAUDRATE 115200
186 #define NUMBER_OF_CHANNELS 4
187 #define NUMBER_OF_TOTAL_SAMPLES 1000
188 #define ADC_DELAY_US -2000
189
190 // Value for conversion
191 const float CONVERSION_FACTOR = 3.3f * 1000 / (1 << 12);
192
193 // Array to store values of the samples
194 uint16_t values_mv[NUMBER_OF_TOTAL_SAMPLES][NUMBER_OF_CHANNELS];
195 // Flag of sampling done
196 static bool sampling_done = false;
197
198 // Function prototypes
199 void init_uart(uint8_t tx_pin, uint8_t rx_pin);
200 void init_adc(uint8_t adc_channel_0, uint8_t adc_channel_1, uint8_t adc_channel_2,
201             uint8_t adc_channel_3);
202 bool read_adc(struct repeating_timer *t);
203 void start_sampling(void);
204 void build_json(char *data, uint total_bytes);
205 void send_data(char *data);
206
207 int main(void) {
208     // Pins for ADC
209     const uint8_t ADC_PIN_CHANNEL_0 = 26;
210     const uint8_t ADC_PIN_CHANNEL_1 = 27;
211     const uint8_t ADC_PIN_CHANNEL_2 = 28;
212     const uint8_t ADC_PIN_CHANNEL_3 = 29;
213
214     // Pins for UART
215     const uint8_t UART_TX_PIN = 0;
216     const uint8_t UART_RX_PIN = 1;
217
218     // Amount of bytes to send
219     uint TOTAL_BYTES_TO_SEND = sizeof({}) + sizeof("\s000\":[0000,0000,0000,0000],")
220     * NUMBER_OF_TOTAL_SAMPLES;
221
222     // JSON data to send
223     char data_to_send[TOTAL_BYTES_TO_SEND];
224
225     stdio_init_all();
226     // Initialize UART
227     init_uart(UART_TX_PIN, UART_RX_PIN);
228     // Initialize ADC
229     init_adc(ADC_PIN_CHANNEL_0, ADC_PIN_CHANNEL_1, ADC_PIN_CHANNEL_2,
230             ADC_PIN_CHANNEL_3);
231     // Start the sampling
232     start_sampling();
233
234     while (true) {
235         if (sampling_done) {
236             //printf("Sampling done\n");

```

```

234         // Make the JSON to send it
235         build_json(data_to_send, TOTAL_BYTES_TO_SEND);
236         //printf("%s", data_to_send);
237         // Send the JSON
238         send_data(data_to_send);
239         // Clear sampling flag and restart sampling
240         sampling_done = false;
241         start_sampling();
242     }
243 }
244
245     return 0;
246 }
247
248 void init_uart(uint8_t tx_pin, uint8_t rx_pin) {
249     // UART setup
250     uart_init(UART_ID, BAUDRATE);
251     gpio_set_function(tx_pin, GPIO_FUNC_UART);
252     gpio_set_function(rx_pin, GPIO_FUNC_UART);
253     uart_set_format(UART_ID, 8, 1, UART_PARITY_NONE);
254     uart_set_fifo_enabled(UART_ID, true);
255 }
256 void init_adc(uint8_t adc_channel_0, uint8_t adc_channel_1, uint8_t adc_channel_2,
257              uint8_t adc_channel_3) {
258     adc_init();
259     // Initialize ADC channels
260     adc_gpio_init(adc_channel_0);
261     adc_gpio_init(adc_channel_1);
262     adc_gpio_init(adc_channel_2);
263     adc_gpio_init(adc_channel_3);
264 }
265 // Calback of the timer
266 bool read_adc(struct repeating_timer *t) {
267     // Counter of sampling
268     static uint sampling_count = 0;
269
270     // Verify that the count don't exceed the total number of samples
271     if (sampling_count >= NUMBER_OF_TOTAL_SAMPLES) {
272         sampling_done = true;
273         cancel_repeating_timer(t);
274         sampling_count = 0;
275     } else {
276         for (int channel = 0; channel < NUMBER_OF_CHANNELS; channel++) {
277             // Read ADC channel
278             adc_select_input(channel);
279             uint16_t adc_value = adc_read();
280             // Convert it to mV
281             values_mv[sampling_count][channel] = round(adc_value * CONVERSION_FACTOR);
282         }
283         sampling_count++;
284     }
285     return true;
286 }
287
288 void start_sampling(void) {
289     // Start timer
290     static struct repeating_timer timer;
291     sampling_done = false;
292     // Set the read_adc function as callback with sample delay of 2000 us in order to
293     // have 500 Hz of sampling frequency
294     add_repeating_timer_us(ADC_DELAY_US, read_adc, NULL, &timer);
295 }
296 void print_adc_values(void) {
297     for (int sampling_number = 0; sampling_number < NUMBER_OF_TOTAL_SAMPLES;
298         sampling_number++) {

```

```

298     for (int channel = 0; channel < NUMBER_OF_CHANNELS; channel++) {
299         printf("values_mv[%d][%d]: %d; ", sampling_number, channel, values_mv[
sampling_number][channel]);
300     }
301 }
302 }
303
304 // Function which transforms the adc_data to a JSON
305 void build_json(char *data, uint total_bytes) {
306     // Stores the total number of bytes for the JSON
307     char *str = (char*) malloc(total_bytes);
308     if (str == NULL) {
309         printf("Memory allocation failed\n");
310         return;
311     }
312
313     sprintf(str, "");
314     // Print each value of the ADC channel until completing all the samplings
315     for (int sampling_number = 0; sampling_number < NUMBER_OF_TOTAL_SAMPLES;
sampling_number++) {
316         char aux[30];
317
318         // Check if it isn't the last one
319         if (sampling_number < NUMBER_OF_TOTAL_SAMPLES - 1) {
320             sprintf(aux, "\"s%d\":[%d,%d,%d,%d]", sampling_number, values_mv[
sampling_number][0],
321                 values_mv[sampling_number][1], values_mv[sampling_number][2], values_mv[
sampling_number][3]);
322         } else {
323             sprintf(aux, "\"s%d\":[%d,%d,%d,%d]", sampling_number, values_mv[
sampling_number][0],
324                 values_mv[sampling_number][1], values_mv[sampling_number][2], values_mv[
sampling_number][3]);
325         }
326         // Concatenate the auxiliary string
327         strcat(str, aux);
328     }
329     // Concatenate the end of line to detect the end of the JSON
330     strcat(str, "]\n");
331     strcpy(data, str);
332     free(str);
333 }
334
335 // Send data by UART
336 void send_data(char *data) {
337     uart_puts(UART_ID, data);
338 }
339
340 */

```

## 9.6. Página Web

El sitio web ofrece una breve descripción sobre nosotros, nuestros objetivos, y proporciona enlaces a nuestras redes sociales, donde podrán contactarnos o conocer más acerca de nuestro trabajo.

Página Web

### 9.6.1. Lenguaje utilizado en la página web y su código

El desarrollo de la página web se realizó utilizando los lenguajes HTML y CSS.

Aquí se adjuntan los códigos de la página web:

Index.css

```

1  *{
2      box-sizing: border-box;

```

```

3 }
4
5 html{
6     scroll-behavior: smooth;
7 }
8
9 body{
10     font-family: 'Roboto', sans-serif;
11     margin: 0;
12     padding: 0%;
13     background-image: linear-gradient(to right, #0f3443 0%, #34e69f 100%);
14     color: white;
15 }
16
17 h1{ font-size: 3.5em;}
18 h2{ font-size: 2.7em;}
19 h3{ font-size: 2em;}
20 p{ font-size: 1.25em;}
21 ul{ list-style: none;}
22 li{ font-size: 1.25em;}
23
24 button{
25     font-size: 1.25em;
26     font-weight: bold;
27     padding: 10px 30px;
28     border-radius: 5px;
29     border: 2px solid rgba(0,0,0,0.3);
30     box-shadow: 2px 2px 10px rgba(0,0,0,0.5);
31     color: white;
32     background-color: grey;
33 }
34
35 button:hover{
36     background-color: rgb(101, 33, 165);
37 }
38
39 .container{
40     max-width: 1400px;
41     margin: auto;
42 }
43
44 .color-acento{ color: blueviolet; }
45
46 header{
47     background-color: rgb(14, 26, 52);
48 }
49
50
51 .social-icons {
52     display: grid;
53     font-size: 3rem;
54     grid-template-columns: repeat(5, 1fr);
55     grid-auto-flow: column;
56     grid-auto-columns: 1fr;
57     align-items: center;
58     padding-right: -100px;
59 }
60
61 header .logo{
62     margin: 0;
63     padding: 25px 30px;
64     font-weight: bold;
65     color: blueviolet;
66     font-size: 1.6em;
67     height: 15vh;
68 }
69 }
70

```

```

71 header .container{
72     display: flex;
73     flex-direction: column;
74     align-items: center;
75 }
76
77 header nav{
78     display: flex;
79     flex-direction: column;
80     text-align: center;
81     padding-bottom: 25px;
82 }
83
84 header a{
85     padding: 5px 12px;
86     text-decoration: none;
87     font-weight: bold;
88     color:rgb(183, 219, 251);
89 }
90
91 header a:hover{
92     color: blueviolet;
93 }
94
95 #hero{
96     display: flex;
97     align-items: center;
98     justify-content: center;
99     text-align: center;
100    flex-direction: column;
101    height: 90vh;
102    background-image: linear-gradient(
103        0deg,
104        rgba(0,0,0,0.5),
105        rgba(0,0,0,0.5)
106    )
107    ,url("he.jpg");
108    background-repeat: no-repeat;
109    background-size: cover;
110    background-position: center center;
111 }
112
113 #hero h1{
114     color:rgb(183, 219, 251);
115 }
116
117 #hero button{
118     font-size: 1.75em;
119 }
120
121 #somos-proya .container{
122     text-align: center;
123     padding: 200px 12px;
124 }
125
126 #nuestros-programas{
127     background-color: rgb(14, 26, 52);
128     color:rgb(183, 219, 251);
129     text-align: center;
130 }
131
132 #nuestros-programas .container{
133     padding: 150px 12px;
134 }
135
136 #nuestros-programas h2{
137     margin-top: 0;
138     font-size: 3.2em;

```

```

139 }
140
141 #nuestros-programas p{
142     display: none;
143 }
144
145 #nuestros-programas .carta{
146     background-position: center center;
147     background-size: cover;
148     padding: 50px 0px;
149     margin: 30px;
150     border-radius: 15px;
151 }
152
153 .carta:first-child{
154     background-image: linear-gradient(
155         0deg,
156         rgba(0,0,0,0.5),
157         rgba(0,0,0,0.5)
158     )
159     ,url("media/front-end.jpg");
160 }
161
162
163 .carta:nth-child(2){
164     background-image: linear-gradient(
165         0deg,
166         rgba(0,0,0,0.5),
167         rgba(0,0,0,0.5)
168     )
169     ,url("media/full-stack.jpg");
170 }
171
172 .carta:nth-child(3){
173     background-image: linear-gradient(
174         0deg,
175         rgba(0,0,0,0.5),
176         rgba(0,0,0,0.5)
177     )
178     ,url("media/python.jpg");
179 }
180
181
182 .carta:nth-child(4){
183     background-image: linear-gradient(
184         0deg,
185         rgba(0,0,0,0.5),
186         rgba(0,0,0,0.5)
187     )
188     ,url("media/full-stack.jpg");
189 }
190
191 .carta:nth-child(5){
192     background-image: linear-gradient(
193         0deg,
194         rgba(0,0,0,0.5),
195         rgba(0,0,0,0.5)
196     )
197     ,url("media/python.jpg");
198 }
199
200 .carta:nth-child(6){
201     background-image: linear-gradient(
202         0deg,
203         rgba(0,0,0,0.5),
204         rgba(0,0,0,0.5)
205     )
206     ,url("media/full-stack.jpg");

```



```

207 }
208
209 #caracteristicas{
210     background-image:linear-gradient(to right, #0f3443 0%,#34e69f 100%);
211     color:rgb(183, 219, 251);
212     text-align: center;
213     display: flex;
214     flex-direction: column;
215     justify-content: center;
216     align-items: center;
217     flex-wrap: wrap;
218 }
219
220 #caracteristicas .container{
221     padding: 150px 12px;
222 }
223
224 #caracteristicas h2{
225     margin-top: 0;
226     font-size: 3.5em;
227     color:rgb(183, 219, 251);
228     text-align: center;
229 }
230
231 #caracteristicas p{
232     display: none;
233 }
234
235 #caracteristicas .carta{
236     background-position: center center;
237     background-size: cover;
238     padding: 50px 0px;
239     margin: 30px;
240     border-radius: 15px;
241 }
242
243 #caracteristicas .container{
244     text-align: center;
245     padding: 250px 12px;
246     width: 100vw;
247 }
248
249 #contacto{
250     display: flex;
251     flex-direction: column;
252     justify-content: center;
253     align-items: center;
254     text-align: center;
255     background-color: rgb(14, 26, 52);
256     color:rgb(183, 219, 251);
257     height: 80vh;
258 }
259
260 #contacto h2{
261     font-size: 9vw;
262 }
263
264 #contacto button{
265     font-size: 5vw;
266 }
267
268 footer{
269     background-color: white;
270 }
271
272 footer p{
273     margin: 0;
274     padding: 12px;

```

```

275     color:  rgb(14, 26, 52);
276 }
277
278 footer .container{
279     height: 150px;
280     display: flex;
281     justify-content: center;
282     align-items: flex-end;
283 }
284
285 .fab.fa-linkedin {
286     color: rgb(183, 219, 251);
287 }
288
289 .fab.fa-instagram {
290     color: rgb(183, 219, 251);
291 }
292
293 .fab.fa-google{
294     color: rgb(183, 219, 251);
295 }
296
297 .fas.fa-music{
298     color: rgb(183, 219, 251);
299 }
300
301 @media (min-width: 850px){
302     header{
303         position: fixed;
304         width: 100%;
305     }
306
307     header .container{
308         flex-direction: row;
309         justify-content: space-between;
310     }
311
312     header nav{
313         flex-direction: row;
314         padding-bottom: 0;
315         padding-right: 20px;
316     }
317
318     #hero h1{
319         font-size: 5em;
320     }
321
322     body{
323         color:  rgb(14, 26, 52);
324     }
325     #somos-proya .container{
326         display: flex;
327         justify-content: space-evenly;
328     }
329
330     #somos-proya .texto{
331         width: 50%;
332         max-width: 600px;
333         text-align: initial;
334         padding-left: 30px;
335         display: flex;
336         flex-direction: column;
337         justify-content: center;
338     }
339
340     #somos-proya h2{
341         margin-top: 0px;
342         font-size: 4em;

```

```

343 }
344
345 #somos-proya .img-container{
346     background-image: url("Fabi.png");
347     background-size: cover;
348     background-position: center center;
349     height: 500px;
350     width: 400px;
351 }
352
353 #nuestros-programas .programas{
354     display: flex;
355     justify-content: center;
356 }
357
358 #nuestros-programas p{
359     display: block;
360     margin-bottom: 30px;
361 }
362
363 #nuestros-programas h2{
364     font-size: 4em;
365 }
366
367 #nuestros-programas h3{
368     margin-top: 0;
369 }
370
371 #nuestros-programas .carta{
372     padding: 50px;
373     background-size: 100% 150px;
374     background-repeat: no-repeat;
375     background-position-y: 0;
376     background-color: rgb(50, 50, 50);
377     box-shadow: 2px 2px 10px rgba(0,0,0,0.5);
378 }
379
380 #caracteristicas .programas{
381     display: flex;
382     flex-wrap: wrap;
383     justify-content: center;
384 }
385
386 #caracteristicas p{
387     display: block;
388     margin-bottom: 30px;
389 }
390
391 #caracteristicas h2{
392     font-size: 4em;
393 }
394
395 #caracteristicas h3{
396     margin-top: 0;
397 }
398
399 #caracteristicas .carta{
400     padding: 50px;
401     background-size: 100% 150px;
402     background-repeat: no-repeat;
403     background-position-y: 0;
404     background-color: rgb(50, 50, 50);
405     box-shadow: 2px 2px 10px rgba(0,0,0,0.5);
406     max-height: 80vh;
407     max-width: 60vh;
408 }
409
410 .carta:first-child{

```

```

411     background-image: linear-gradient(
412         0deg,
413         rgba(0,0,0,0.5),
414         rgba(0,0,0,0.5)
415     )
416     ,url("media/front-end-cropped.jpg");
417
418 }
419
420 .carta:nth-child(2){
421     background-image: linear-gradient(
422         0deg,
423         rgba(0,0,0,0.5),
424         rgba(0,0,0,0.5)
425     )
426     ,url("media/full-stack-cropped.jpg");
427 }
428
429 .carta:nth-child(3){
430     background-image: linear-gradient(
431         0deg,
432         rgba(0,0,0,0.5),
433         rgba(0,0,0,0.5)
434     )
435     ,url("media/python-cropped.jpg");
436 }
437
438
439 #caracteristicas{
440     background-image: url("media/background-2.jpeg");
441     background-repeat: no-repeat;
442     background-size: 500px 400px;
443     background-position: calc(100vw - 500px) 120px;
444 }
445
446 #caracteristicas .container{
447     text-align: initial;
448 }
449
450 #caracteristicas ul{
451     margin-left: 100px;
452 }
453
454 #contacto h2{
455     font-size: 5em;
456 }
457
458
459 #contacto button{
460     font-size: 2em;
461 }
462
463 footer .container{
464     justify-content: flex-end;
465 }
466 #contacto a{
467     font-size: 2rem;
468     text-align: left;
469 }
470 }
471
472 @media (min-width: 1200px) {
473     #caracteristicas{
474         background-position-x: calc(100vw - 800px);
475     }
476 }

```

Index.html

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <title>PROYECTO BIAS</title>
5   <link rel="stylesheet" href="index.css">
6   <link
7     rel="stylesheet"
8     href="https://use.fontawesome.com/releases/v5.8.2/css/all.css"/>
9 </head>
10 <body>
11   <header>
12     <div class="container">
13       
14       <nav>
15         <a href="#somos-proya"> Que es BIAS?</a>
16         <a href="#nuestros-programas"> Por qu nuestro proyecto?</a>
17         <a href="#caracteristicas"> Quienes Somos?</a>
18         <a href="#contacto">Contactanos</a>
19       </nav>
20     </div>
21   </header>
22
23   <section id="hero">
24     <h1>Bienvenido al <br>Proyecto BIAS</h1>
25   </section>
26
27   <section id="somos-proya">
28     <div class="container">
29       <div class="img-container"></div>
30       <div class="texto">
31         <h2> Qu es BIAS?</h2>
32         <p>BIAS es un proyecto innovador busca transformar la vida de
33         personas con discapacidades motoras mediante sillas de ruedas controladas por
34         se ales EEG. <br> Un dispositivo capta la actividad cerebral y la traduce en
35         comandos, permitiendo a los usuarios dirigir la silla con sus pensamientos.
36         Avanzar, girar o detenerse es posible, todo con la fuerza de la mente.<br> Esta
37         tecnolog a promete mayor independencia y una mejor calidad de vida para quienes
38         enfrentan barreras de movilidad. Un futuro donde la mente lidera el camino est
39         m s cerca que nunca.</p>
40       </div>
41     </div>
42   </section>
43
44   <section id="nuestros-programas">
45     <div class="container">
46       <h2> Por qu nuestro proyecto?</h2>
47       <div class="programas">
48         <div class="carta">
49           <h3>Inclusi n</h3>
50           <p><br><br><br>Buscamos la igualdad de oportunidades para
51           aquellos que m s lo necesitan y generar una independencia nunca lograda hasta
52           este momento, todo en vista de conseguir una vida m s plena e igualitaria.</p>
53         </div>
54         <div class="carta">
55           <h3>Igualdad</h3>
56           <p><br><br><br>Creemos que la igualdad es la base de una mejor
57           sociedad no solo para garantizar que todos reciban iguales oportunidades si no
58           para no perder gente talentosa que, de otra forma, no tendr a la oportunidad de
59           desarrollar su potencial</p>
60         </div>
61         <div class="carta">
62           <h3>Sentar un Precedente</h3>
63           <p><br><br><br>Este proyecto busca crear una alternativa realista
64           para la creaci n de una silla de ruedas EEG econ mica y accesible, un objeto el
65           cual en este pa s no se comercializa</p>
66         </div>
67       </div>
68     </div>
69   </section>

```

```

55 </section>
56
57 <section id="caracteristicas">
58   <div class="container">
59     <h2> Quienes Somos?</h2>
60     <div class="programas">
61       <div class="carta">
62         <h3>D az Meli n , Danilo</h3>
63         <p><br>Dise o de Esquem ticos. Dise o estructural. Creaci n
de P gina Web. Marketing y B squeda de Sponsors. Construcci n de Planos.
Dise o de sistema de movimiento. Programaci n IA <a href="https://instagram.
com/_danilodiaz">
64         <br><br><div class="social-icons"><i class="fab fa-instagram"
></i> </a><a href="mailto:danilodiaz934@gmail.com">
65         <i class="fab fa-google"></i></a><a href="https://www.
linkedin.com/in/danilodiazmelion/">
66         <i class="fab fa-linkedin"></i></a> </p></div>
67       </div>
68       <div class="carta">
69         <h3>Sojka, Santiago</h3>
70         <p><br><br><br>Soldadura de PCBs. Redes Sociales. Edici n de
videos. Creaci n de Contenido. Dise o de sistema de movimiento. Investigaci n.
Programaci n IA. Programaci n de Conexi n para el Sensor EEG.<a href="https
://instagram.com/sojkaa.sant">
71         <br><br><div class="social-icons"><i class="fab fa-instagram"
></i> </a><a href="mailto:santiagosojka@gmail.com">
72         <i class="fab fa-google"></i></a><a href="https://www.
linkedin.com/in/santiago-sojka-817198271/">
73         <i class="fab fa-linkedin"></i></a> </p></p></div>
74       </div>
75       <div class="carta">
76         <h3>Montenegro, Luciano</h3>
77         <p><br><br>Conversi n de Esquem ticos a PCBs. Dise o estructural.
Dise o de sistema de movimiento. Programaci n IA. Creaci n de contenido. <a href
="https://instagram.com/luchito.montenegro">
78         <br><br><div class="social-icons"><i class="fab fa-instagram"
></i> </a><a href="mailto:nias.project.impa@gmail.com">
79         <i class="fab fa-google"></i></a><a href="https://www.
linkedin.com/in/luciano-montenegro-3215aa304/">
80         <i class="fab fa-linkedin"></i></a> </p></p></div>
81       </div>
82       <div class="carta">
83         <h3>De Blasi, Luca</h3>
84         <p><br><br><br>Programaci n del sistema de movimiento.
Construcci n de Planos. Dise o estructural. Programaci n IA.
<a href="https://instagram.com/luca.deblasii">
85         <br><br><div class="social-icons"><i class="fab fa-instagram"
></i> </a><a href="mailto:nias.project.impa@gmail.com">
86         <i class="fab fa-google"></i></a><a href="https://www.
linkedin.com/in/luca-de-biasi-31164b304/">
87         <i class="fab fa-linkedin"></i></a> </p></p></div>
88       </div>
89       <div class="carta">
90         <h3>Adell, Nicol s Fabi n</h3>
91         <p><br><br>Programaci n del Sistema de Movimiento. Preparaci n
del Github. Redes Sociales. Programaci n IA. Programaci n de Conexi n para el
Sensor EEG.<a href="https://instagram.com/nicolas.adell">
92         <br><br><div class="social-icons"><i class="fab fa-instagram"
></i> </a><a href="mailto:nias.project.impa@gmail.com">
93         <i class="fab fa-google"></i></a><a href="http://www.
linkedin.com/in/nicolas-adell-354508297">
94         <i class="fab fa-linkedin"></i></a> </p></p></div>
95       </div>
96       <div class="carta">
97         <h3>Gil Soria, Ian</h3>
98         <p><br><br><br>Marketing y B squeda de Sponsors. Dise o de
Esquem ticos y PCBs. Programaci n IA. Encargado de Carpeta de Campo y
Documentaci n T cnica.<a href="https://instagram.com/ian_gilsooor">
99

```

```

100         <br><br><div class="social-icons"><i class="fab fa-instagram"
101     ></i> </a><a href="mailto:nias.project.impa@gmail.com">
102         <i class="fab fa-google"></i></a><a href="https://www.
103         linkedin.com/in/ian-lucas-gil-soria-a8090b2a8?utm_source=share&utm_campaign=
104         share_via&utm_content=profile&utm_medium=android_app ">
105         <i class="fab fa-linkedin"></i></a> </p></p></div>
106     </div>
107 </div>
108 </section>
109
110 <section id="contacto">
111     <h2>Contactanos</h2>
112     <div><a href="https://instagram.com/proyecto.bias">
113         <i class="fab fa-instagram"></i> </a> <p>@proyecto.bias</p></div>
114
115     <a href="mailto:bias.project.impa@gmail.com">
116         <i class="fab fa-google"></i></a><p>bias.project.impa@gmail.com</p>
117     <a href="https://tiktok.com/@proyecto_bias">
118         <i class="fas fa-music"></i></a> <p>@proyecto.bias (TikTok)</p>
119
120 </section>
121
122 <footer>
123     <div class="container">
124         <p>&copy; Proyecto BIAS 2024</p>
125     </div>
126 </footer>
127 </body>
128 </html>

```