Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Control de un Robot Animatrónico con Tecnología Brain – Computer Interface

Autor: Ángel Maresca Bustos
Tutor: José María Maestre Torreblanca
José Ramón Domínguez Frejo

Dpto. de Ingeniería de sistemas y automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Control de un Robot Animatrónico con Tecnología Brain – Computer Interface

Autor:
Ángel Maresca Bustos

Tutor:
José María Maestre Torreblanca
José Ramón Domínguez Frejo

Dpto. de Ingeniería de sistemas y automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Trabajo Fin de Grado: Control de un Robot Animatrónico con Tecnología Brain – Computer Interface

Autor: Ángel Maresca Bustos

Tutor: José María Maestre Torreblanca,
José Ramón Domínguez Frejo

Profesor
Titular:

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El secretario del Tribunal

A mi familia

A mis amigos

A mis profesores

Gracias…

Agradecimientos

Por fin un párrafo que sale del corazón y no de la cabeza… Muchas gracias a todos los que han podido y querido acompañarme en este largo, intenso y bonito camino…

Gracias a mis padres y a mi hermano, por apoyarme siempre de manera incondicional y respaldarme en mis muchas horas de estudio y trabajo liberándome de otras labores, por alimentarme con exquisitos platos siempre que lo he necesitado y por comprender mis momentos de abstracción después de horas de estudio, también a mi hermano por estar preparado siempre para distraerme con algún partido de play o de ping-pong…

Gracias a mi novia Irene, por acompañarme durante todo este tiempo de manera activa y presente, siendo mi compañera incondicional, de estudio y descanso, gracias por confiar siempre en mí y por escucharme horas y horas hablando de lo mismo con la misma ilusión que la primera vez.

Gracias a mis amigos, a Miguel, por ser mi compañero de batallas ingenieriles, qué buena dupla… a César, Adolfo, Manu… por ayudarme a desconectar (que es una labor muy importante), confiar siempre en mí y por alegraros de mis éxitos como si fueran vuestros…

Gracias también a todos aquellos profesores que saben transmitir las cosas más apasionantes de la ingeniería con ilusión, paciencia y humildad. Sin duda el paso por la ETSI curte a cualquiera, y es un camino duro, pero encontrarte con apasionados es muy gratificante.

Gracias a mi perro Tango, por la compañía, a mi familia y amigos por el apoyo, a mis profesores, por los conocimientos y en cierta manera también a internet por los conocimientos… A todos por formar parte de este camino y ayudarme a conseguir mis éxitos.

Ángel Maresca Bustos

Sevilla, 2023

Resumen

Controlar robots con la mente… ¿Es posible?, hoy en día ya existen tecnologías en desarrollo que permiten el control de brazos robóticos o sillas de ruedas con la mente pero… Hasta qué punto un estudiante, con un dispositivo de ondas EEG (Encefalografía) y un Robot Animatrónico impreso en 3D puede llevar a cabo esta misión?

En este proyecto, se presenta el reto de fusionar la tecnología Interfaz Cerebro-Computadora de la empresa Emotiv, que permite el procesamiento y envío de ordenes por medio de señales EEG, con un robot animatrónico diferencial creado por la empresa RobotCanCry, para conseguir el control de nuestro robot "Curro" mediante ondas cerebrales…

A lo largo del proyecto, se introducirán y explicarán las puestas en marcha, programación, alcances y limitaciones de estas tecnologías. El envío de órdenes por medio de señales EEG, requiere de un entrenamiento previo por parte del usuario, se explicarán por tanto las distintas técnicas de entrenamiento que se han seguido: pensamiento en colores, operaciones matemáticas… así como los resultados obtenidos y las limitaciones de cada una. Se hará un gran uso del control por medio de expresiones faciales, ya que veremos cuáles son las limitaciones del control mental y por qué no es tan viable como quizás podríamos pensar, así como veremos como sí es posible mediante expresiones faciales, siendo este tipo de control el utilizado para el prototipo final.

Entre otros muchos retos, nos enfrentaremos a problemas en la programación de los movimientos del robot, la sincronización y conexión inalámbrica Emotiv – Robot, ciertos inconvenientes en el uso del dispositivo Emotiv… Todos intentarán resolverse de la mejor manera, logrando así un robot motorizado, con funciones de interacción con el entorno y comunicación con el usuario, sistemas de seguridad antichoque, captura de fotografías…

Este proyecto ilusionante de principio a fin pasará por distintas etapas que llevarán tanto a mi persona, como al robot Curro, a un nivel superior de aprendizaje.

Abstract

Controlling robots with the mind... Is it possible? Nowadays, there are already technologies in development that allow the control of robotic arms or wheelchairs with the mind, but to what extent can a student, with an EEG (Electroencephalography) wave device and a 3D-printed animatronic robot, accomplish this mission?

In this project, the challenge is presented to merge the Brain-Computer Interface (BCI) technology from Emotiv, which allows processing and sending commands through EEG signals, with a differential animatronic robot created by RobotCanCry, to achieve control of our robot "Curro" through brainwaves.

Throughout the project, the implementation, programming, scope, and limitations of these technologies will be introduced and explained. Sending commands through EEG signals requires prior training by the user. Therefore, the different training techniques that have been followed will be explained, such as color thinking, mathematical operations, along with the results obtained and the limitations of each technique. A significant use will be made of control through facial expressions since we will explore the limitations of mental control and why it may not be as viable as we might think. Additionally, we will demonstrate the feasibility of control through facial expressions, which will be used for the final prototype.

Among many other challenges, we will face problems in programming the robot's movements, synchronization and wireless connection between Emotiv and the robot, and certain inconveniences in the use of the Emotiv device. All these challenges will be attempted to be resolved in the best possible way, resulting in a motorized robot with interaction capabilities with the environment and communication with the user, anti-collision safety systems, and photo capture capabilities.

This exciting project, from start to finish, will go through different stages that will lead both myself and the robot “Curro” to a higher level of learning.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGO

Notación

EEG	Electroencefalografía
BCI	Brain Computer Interface
MEG	Magnetoencefalografía
fMRI	Imagen por resonancia magnética funcional
A	Amperios
AC	Alternative Current
Cm	Centímetros
dB	Decibelios
G	Gramos
GHz	Gigahercios
Hz	Hercios
GND	Ground
VCC	Voltage Common Colector
PWM	Pulse Width Modulation
LED	Light Emiting Diode
USB	Universal Serial Bus
ROS	Robot Operating System
μs	microsegundos
<	Menor que
>	Mayor que
=	Igual que
→	Entonces

1 INTRODUCCIÓN

Cualquier tecnología suficientemente avanzada es equivalente a la magia.

- Arthur C. Clarke-

1.1. Motivación

Todos somos conscientes de la velocidad a la que avanza la tecnología hoy en día, casi cualquier elemento que podamos observar a nuestro alrededor está dotado de electrónica, y cada vez más, de inteligencia. La ingeniería tiene como uno de sus principales objetivos, entre otros, diseñar sistemas que faciliten, y, por ende, mejoren, la experiencia del usuario con la tecnología. Si nos centramos un poco más en el ámbito de la robótica, no nos resulta tan extraño realizar numerosas actividades de la vida cotidiana con la ayuda de robots inteligentes, Thermomix, Roomba, Alexa, videojuegos con realidad virtual… Es por ello que en este proyecto se ha intentado indagar por una vía que quizás esté todavía un poco menos trabajada… El control de un robot animatrónico con la ayuda de una Interfaz Cerebro - Computadora, es decir, controlar a nuestro robot con estímulos cerebrales.

Este proyecto se ha visto motivado por la búsqueda de innovación en el control de robots, pero ha cobrado aún más sentido cuando se han estudiado las numerosas aplicaciones que este tipo de sistemas podrían tener en un futuro más bien próximo. Una de las más interesantes es la ayuda que se le podría brindar a personas con lesiones cerebrales o espinales ya sea para desarrollar su vida cotidiana con mayor comodidad, como podría ser mover una silla de ruedas, o controlar sus prótesis robóticas, como para enriquecer sus posibilidades de ocio con la tecnología, controlando un robot, un coche de radiocontrol, o jugando a videojuegos…

Soy consciente que ya existen tecnologías que cubren algunas de estas funciones, pero es muy importante poder comenzar a desarrollar este tipo de sistemas a un nivel más cotidiano, no tan sofisticado, que permita a un usuario corriente poder acceder a este tipo de experiencias.

1.2. Objetivo

El objetivo de este proyecto se enfoca en realizar el control de un robot animatrónico por medio de la tecnología BCI (Brain-Computer Interface). Se estudiarán principalmente dos vías, el control del robot mediante comandos mentales, y el control de este mediante expresiones faciales. De este modo, se harán experimentos para comprobar ante qué tipo de control la respuesta es mejor y cuáles

son las ventajas e inconvenientes de cada método.

Así mismo se pretende mostrar el proceso de entrenamiento que ha sido necesario para poder asociar unas acciones de movimiento determinadas del robot a pensamientos o expresiones faciales, con idea de que el presente documento sirva de ayuda para todo aquel que quiera iniciarse en el mundo de la Interacción Cerebro - Computadora por medio de la tecnología Emotiv: Instalación del software, configuración del entorno, desarrollo de aplicaciones de control.

El robot con el que vamos a trabajar es 'Currito', un robot animatrónico desarrollado por la empresa de la industria animatrónica Robots Can Cry mediante impresión 3D. 'Currito' es un robot móvil diferencial programado en Arduino, que posee además la capacidad de movimiento de las cejas, el cuello, la cresta, y la rotación de su cuerpo, así como de una cámara Pi, una tira de leds, un sensor ultrasónico, y un micrófono con altavoces.

Por otro lado, contamos con la tecnología BCI (Brain-Computer Interface) de Emotiv, una empresa que se dedica al desarrollo de tecnologías que permiten la comunicación hombre-máquina por medio de la mente. Los dispositivos Emotiv y los algoritmos de machine-learning que la empresa ha desarrollado, permiten convertir las ondas cerebrales en señales digitales, lo que permitirá a su vez entrenar comandos mentales que asociemos a acciones de nuestro robot.

En este proyecto se desarrollará todo el proceso de comunicación y conexión entre la diadema de Emotiv y Currito, así como el desarrollo de los algoritmos y técnicas utilizadas para poder controlar a nuestro robot con la mente. Pretendiendo así, que este trabajo sirva como guía de instalación al usuario que quiera aprender a configurar el control de 'Currito' con los comandos mentales que este entrene.

Figura 1: Imagen Robot Curro

1.3. Estructura de la memoria

En este documento, se irán desarrollando los distintos conceptos que abarcan este proyecto de la siguiente manera:

En primer lugar se analizará el **Estado del arte**, es decir, nos pondremos en contexto de cómo se encuentra el objeto de estudio, en este caso la Interfaz Cerebro – Computadora, y las aplicaciones de EEG, cómo surgió, cómo ha ido evolucionando, y en qué punto nos encontramos ahora.

Acto seguido pasaremos a explicar la **tecnología Emotiv**, que es la empresa que proporciona el dispositivo de medición de EEG que usaremos para controlar a nuestro robot, así como la que nos proporciona el software para su programación. Se explicará de manera guiada cómo introducirse en el mundo de los dispositivos EEG, cómo configurarlos, cómo entrenarlos, qué usos podemos darle, etc.

Una vez vista la tecnología Emotiv, presentaremos a nuestro **robot Curro** y explicaremos cómo se creó, describiremos sus características físicas, así como cada uno de sus componentes (sensores, actuadores…) y explicaremos cómo se realizan todas las conexiones de cada uno de los elementos que hacen posible su funcionamiento.

Ya presentados los dos principales elementos de nuestro proyecto: la tecnología Emotiv, y nuestro robot, pasaremos a explicar todo el proceso de entrenamiento que se llevará a cabo para entrenar las órdenes mentales que le enviaremos a nuestro robot. Veremos distintas técnicas de entrenamiento mental, y también estudiaremos el entrenamiento mediante comandos faciales, que veremos cómo cobra gran importancia en el proyecto.

Después explicaremos cada uno de los programas que se han creado hacer posible el control de nuestro robot. Se detallarán los diagramas de flujo que se han seguido en cada caso y se explicarán cada una de las funciones que componen los programas.

Por último, se comentarán las conclusiones obtenidas del proyecto, así como los inconvenientes que se han ido encontrando y cómo se han solventado.

2 ESTADO DEL ARTE

"Si nuestro cerebro fuera tan sencillo como para poder entenderlo, seríamos tan tontos que, de todos modos, no lo podríamos entender."

Jostein Gaarder

2.1 BCI: Interfaz Cerebro Computadora

Los sistemas de Interfaz Cerebro Computadora (BCI en inglés) se basan en la captación de las señales eléctricas que emite nuestro cerebro para la generación de comandos de computador que controlen diversos sistemas electrónicos: silla de rueda con motores, control de robots, drones…

Esta tecnología surge de la necesidad de establecer un nuevo canal de comunicación entre el humano y su entorno, un canal que no dependa de vías nerviosas o musculares. Estas interfaces se basan en la captación de señales asociadas a procesos mentales, obtenidas gracias a señales neurofisiológicas adquiridas a través de **EEG, MEG, fMRI**, entre otras.

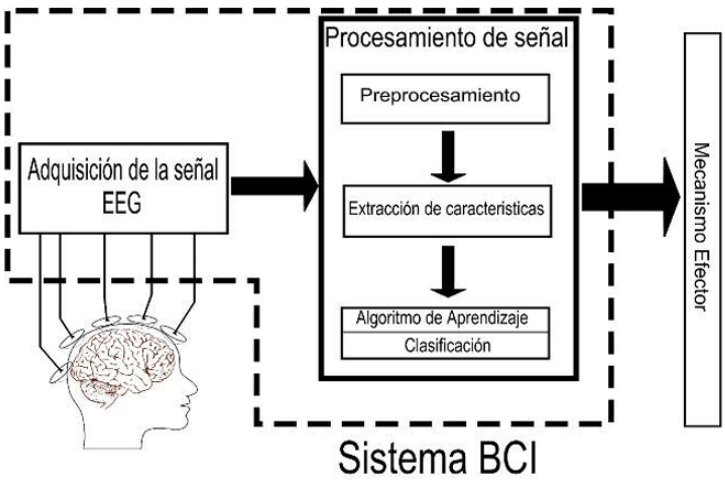

Figura 2: Esquema funcional BCI

Cuando a mediados del siglo XX se comenzó a investigar con esta nueva vía de comunicación del ser humano con el entorno, las aplicaciones y el uso de esta nueva tecnología tenía diversas limitaciones, entre ellas podemos destacar: El elevado coste de los recursos necesarios para su implementación y la necesidad de uso de técnicas invasivas para la captación de señales, se requería de implantes invasivos (intervenciones quirúrgicas) para la lectura de señales, y obligaban a desarrollar estas tecnologías únicamente desde un punto de vista clínico y en ocasiones muy particulares. [1]

Posteriormente, surgieron técnicas no invasivas, son sistemas que utilizan electrodos de superficie con los que se registran electroencefalogramas (EEG) y potenciales evocados (PE). En la actualidad se combinan ambas técnicas dependiendo de las necesidades que se vayan a cubrir en cada caso.

2.2 Adquisición de señales

2.2.1 Métodos Invasivos y No Invasivos:

Como se explicó brevemente en el apartado anterior, existen varios métodos de adquisición de señales, estos podemos clasificarlos en dos grupos, invasivos y no invasivos.

No Invasivas:

- **Electroencefalografía (EEG):** Es uno de los métodos de adquisición de señales más utilizados en los BCI. Se basa en la colocación de electrodos en el cuero cabelludo para medir la actividad eléctrica del cerebro. Es un método no invasivo y seguro.
- **Magnetoencefalografía (MEG)**: Es similar a la EEG, pero utiliza sensores para medir los campos magnéticos generados por la actividad eléctrica del cerebro. Requiere equipos especializados y es más costoso que la EEG.
- **Imágenes por resonancia magnética funcional** (fMRI): Permite medir la actividad cerebral mediante la medición de la cantidad de oxígeno que utiliza el cerebro. Es un método costoso y no es práctico para aplicaciones de tiempo real.

Invasivas:

- **Electrocorticografía (ECoG)**: Implica la colocación de electrodos directamente sobre la superficie del cerebro. Es un método invasivo y requiere cirugía, pero ofrece una mayor resolución espacial y temporal que la EEG.
- **Tomografía por emisión de positrones (PET)** y **tomografía por emisión de fotón único (SPECT):** Son métodos de imagenología molecular que permiten medir la actividad cerebral mediante la inyección de sustancias radiactivas. Son métodos costosos y no son prácticos para aplicaciones de tiempo real.

En el desarrollo de este proyecto se ha trabajado con la tecnología Emotiv, esta tecnología realiza la adquisición de señales mediante encefalografía (EEG), esto es así debido a que Emotiv es una empresa que pretende acercar el mundo de BCI al usuario corriente, y esta técnica de adquisición es la más segura, y la que mejor resultado da en relación calidad - costo.

2.2.2 Emotiv EEG

La tecnología Emotiv se basa en la adquisición de señales mediante EEG, a continuación, se explicará de manera resumida en qué consiste esta técnica:

Un electroencefalograma (EEG) es un estudio que mide la actividad eléctrica en el cerebro mediante pequeños discos de metal (electrodos) colocados sobre el cuero cabelludo. Las neuronas cerebrales se comunican a través de impulsos eléctricos y están activas todo el tiempo, incluso mientras

dormimos. Esta actividad se manifiesta como líneas onduladas en un registro electroencefalográfico.

Las señales atraviesan el cuero cabelludo y el cráneo, lo que provocan un empobrecimiento de la señal obtenida, es por ello que se suele emplear un gel EEG o líquido conductor con el fin de reducir la impedancia creada por temperatura, superficie del electrodo, cuero cabelludo, etc.

Los electrodos de un dispositivo de EEG detectan señales en diversas frecuencias de EEG. Para identificar estas señales de EEG en bruto como ondas con diferentes frecuencias, se utiliza un algoritmo llamado Transformada Rápida de Fourier (FFT).

Los cuatro tipos principales de ondas cerebrales se clasifican por su frecuencia: beta, alfa, theta y delta.

En cuanto a las funciones asociadas con estas cuatro frecuencias cerebrales principales, se ha encontrado que cada una de ellas tiene diferentes roles en el cerebro, aunque no existe una correspondencia lineal uno a uno entre una banda de frecuencia y una función cerebral específica.

Ondas cerebrales identificadas por EEG

Ondas Beta (rango de frecuencia de 14 Hz a aproximadamente 30 Hz)

Las ondas beta están más estrechamente asociadas con estar consciente o en un estado despierto, atento y alerta. Las ondas beta de baja amplitud están asociadas con la concentración activa o con un estado mental ocupado o ansioso. Las ondas beta también están asociadas con decisiones motoras (supresión de movimiento y retroalimentación sensorial de movimiento). Cuando se miden con un dispositivo de EEG, las señales a menudo se denominan ondas beta de EEG.

Ondas Alpha (rango de frecuencia de 7 Hz a 13 Hz)

Las ondas alfa a menudo se asocian con un estado mental relajado, tranquilo y lúcido. Las ondas alfa se pueden encontrar en las regiones occipital y posterior del cerebro. Las ondas alfa se pueden inducir cerrando los ojos y relajándose, y rara vez están presentes durante procesos cognitivos intensos como el pensamiento, el cálculo mental y la resolución de problemas. En la mayoría de los adultos, las ondas alfa varían en frecuencia de 9 a 11 Hz. Cuando se miden con un dispositivo de EEG, a menudo se las denomina ondas alfa de EEG.

Ondas Theta (rango de frecuencia de 4 Hz a 7 Hz)

La actividad cerebral dentro de un rango de frecuencia comprendido entre 4 y 7 Hz se denomina actividad Theta. El ritmo theta detectado en la medición de EEG a menudo se encuentra en adultos jóvenes, particularmente en las regiones temporales y durante la hiperventilación. En personas mayores, la actividad theta con una amplitud superior a unos 30 milivoltios (mV) se observa con menos frecuencia, excepto durante la somnolencia. Cuando se miden con un dispositivo de EEG, a menudo se las denomina ondas theta de EEG.

Ondas Delta (rango de frecuencia hasta 4 Hz)

La actividad delta se encuentra predominantemente en los bebés. Las ondas delta están asociadas con etapas profundas del sueño en sujetos mayores. Las ondas delta se han documentado interictalmente (entre convulsiones) en pacientes con convulsiones de ausencia, que implican lapsos de atención breves y repentinos.

Las ondas delta se caracterizan por ondas de baja frecuencia (alrededor de 3 Hz) y de gran amplitud. Los ritmos delta pueden estar presentes durante la vigilia: responden a la apertura de los ojos y también pueden mejorar con la hiperventilación. Cuando se miden con un dispositivo de EEG, a menudo se las denomina ondas delta de EEG. [2]

Dispositivo EEG

Emotiv EEG es un producto ofrecido por la empresa Emotiv que consiste en un dispositivo que realiza un electroencefalograma de alta resolución con 14 canales ubicados y nombrados según el sistema internacional 10-20 [7] más dos canales de referencia, posee un giroscopio de dos ejes, es inalámbrico y tiene una batería para 12 horas de uso continuo. Se comunica con el computador por medio de un receptor USB que recibe las señales y las deriva a su software de detección de expresiones faciales, comandos mentales…

Figura 4: Dispositivo Emotiv

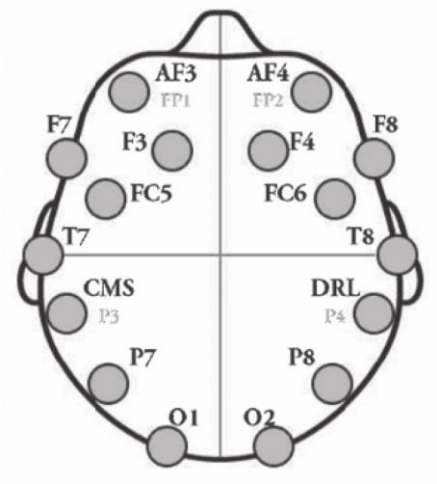

Figura 3: Localización de los electrodos en el kit Emotiv

2.2 Usos del EEG

La monitorización de nuestra actividad cerebral puede ser de gran utilidad en muchos campos de la ciencia, tanto en medicina, como en tecnología, así como en investigación y entretenimiento.

2.2.1 Usos pasivos

- **Rendimiento y bienestar**: atletas, biohackers, y cualquier usuario interesado puede utilizar EEG para monitorizar su actividad cerebral al igual que se puede hacer con las pulsaciones o los pasos que damos en un día. Al igual que los pasos que damos en un día pueden ayudarnos a conocer la actividad física que llevamos, el EEG puede aportarnos información sobre funciones cognitivas como la atención, el estrés, la distracción…

  Esto puede darnos información acerca de cómo se comporta nuestro cerebro antes distintos estímulos durante el día. A su vez, estos resultados obtenidos podemos usarlos para desarrollar técnicas que nos ayuden a reducir el estrés o a mejorar nuestra concentración, al igual que si viéramos que no nos movemos nada en todo el día, decidiríamos salir a andar o a hacer algo de ejercicio para "solucionar" el problema que hubiéramos encontrado.

- **Investigación del consumidor**: los datos de EEG pueden ayudar a las empresas a conocer la respuesta de los usuarios ante los servicios y estímulos que estos reciben. Averiguando así donde focalizan los clientes la atención y qué les resulta realmente satisfactorio y no. El uso

de las neuro tecnologías como el EEG para estudiar las reacciones de los consumidores se denomina neuromarketing.

- **Cuidado de la salud**: Observar datos anormales en un registro de EEG puede ayudar a diagnosticar diversos trastornos cerebrales como disfunción cerebral, traumatismo craneoencefálico, tumores cerebrales, accidente cerebrovascular, trastornos convulsivos como epilepsia…

  Normalmente este monitoreo se combina en medicina con pruebas cognitivas, técnicas de neuroimagen, o monitoreo de actividad cerebral con técnicas invasivas.

- **Estudio del sueño**: La polisomnografía, es una técnica que mide la actividad corporal además de realizar un escáner cerebral. Un monitoreo de EEG durante una sesión de estudio del sueño bajo supervisión de un médico puede servir como prueba de diagnóstico para los trastornos del sueño.

- **Neurociencia cuantitativa**: La neurociencia cuantitativa es un campo de la neurociencia que se centra en la aplicación de técnicas matemáticas y estadísticas para estudiar el cerebro y el sistema nervioso. Esta disciplina busca desarrollar modelos matemáticos que permitan comprender mejor la estructura y la función del cerebro.

  Los investigadores en neurociencia cuantitativa utilizan técnicas de adquisición y análisis de datos avanzadas para medir la actividad cerebral y modelar el comportamiento neuronal, con el objetivo de entender mejor cómo el cerebro procesa la información, cómo se forman los recuerdos y cómo funciona en general. Los modelos matemáticos y estadísticos desarrollados en este campo pueden utilizarse para predecir cómo el cerebro puede responder a diferentes estímulos, y para desarrollar terapias y tratamientos para trastornos neurológicos y psiquiátricos.

2.2.2 Usos activos

- **Juegos EEG**: La tecnología EEG llegó al campo del entretenimiento y la investigación en la década de 1960, aunque ha sido en las últimas décadas cuando ha crecido significativamente. Actualmente la tecnología EEG se utiliza en eventos en vivo para controlar efectos visuales y sonidos en tiempo real, creando una experiencia única y totalmente personalizada para el usuario.

  Cada vez vemos más empresas se atreven a impulsar juegos que implican el monitoreo de la actividad cerebral del usuario, ya sea para controlar la dificultad del juego en función del nivel de atención/relajación del usuario, o bien para la implementación de funciones como movimiento de objetos o lanzamiento de proyectiles con la actividad cerebral.

  Es importante comentar que la tecnología EEG puede tener limitaciones en cuanto a la precisión de la detección de la actividad cerebral y la interpretación de los datos. Durante el desarrollo de este proyecto, se irán explicando estas limitaciones y se observará hasta donde hemos podido exprimir esta tecnología. [3]

- **Interfaz cerebro – máquina**: El uso de la tecnología EEG para el control de robots es la principal motivación de este proyecto, su todavía escaso desarrollo nos permite explorar en un área en la cual podemos realizar pequeños avances que puedan servir de gran ayuda a futuros alumnos. [3]

Aun así en los últimos años la investigación en esta área ha ido creciendo de manera exponencial, entre las aplicaciones más interesantes podemos destacar:

- **Rehabilitación motora**: Para pacientes que han sufrido ictus o lesiones musculares, causando parálisis en extremidades inferiores o superiores. El objetivo es que los pacientes entrenen con BCI durante cierto tiempo, lo que les permitirá reforzar el vínculo que tiene su cerebro con ciertas actividades motoras
- **Restauración motora**: Pacientes con lesión medular que sufran parálisis parciales tienen impedidas acciones cotidianas como agarrar un vaso. Las BCI se pueden utilizar para detectar cuando el paciente está enviando estímulos para querer agarrar el vaso, estos estímulos registrados por el EEG pueden decodificarse y emplearse para enviar estímulos eléctricos a los músculos para facilitar la acción deseada. De esta manera sería posible que el paciente recupere algunos movimientos motores con la posibilidad de seguir usando su extremidad con estas neuro prótesis.

Figura 5: Imagen Neuro prótesis

- **Control de dispositivos robóticos**: Cada vez más se están utilizando BCI's no invasivas para el control de brazos robóticos, robots de navegación autónoma, drones, sillas de ruedas y como se va a desarrollar en este trabajo, robots móviles de control remoto, actualmente como entretenimiento, pero escalable a futuras aplicaciones de índole profesional. [3]

Figura 7: Silla Ruedas EEG

Figura 6: Brazo Robótico EEG

Figura 8: Robot Animatrónico EEG "Curro"

Una vez explicado los inicios de las EEG, su funcionamiento, las interfaces cerebro-computadora, y sus distintas aplicaciones hasta el día de hoy, damos por concluida esta sección de contextualización del proyecto.

3 TECNOLOGÍA EMOTIV

"La mente es un instrumento poderoso. Si no la controlas, ella te controlará a ti".

- Paulo Coelho -

Emotiv es una compañía dedicada al desarrollo de tecnología de interfaz cerebro-computadora (BCI, por sus siglas en inglés) que permite a los usuarios interactuar con dispositivos electrónicos utilizando su actividad cerebral.

La tecnología de Emotiv se basa en el uso de dispositivos de electroencefalografía (EEG) portátiles que registran la actividad eléctrica del cerebro a través de electrodos colocados en el cuero cabelludo. Los algoritmos de Emotiv analizan estos datos EEG para identificar patrones y correlacionarlos con diferentes estados mentales, emociones y movimientos.

A lo largo de este capítulo, se abordarán los siguientes puntos:

- Explicación del proceso de adquisición y procesamiento de señales EEG
- Tutorial de configuración de la diadema EPOC X
- Presentación de las aplicaciones desarrolladas por Emotiv
- Desarrollo extendido del CORTEX API empleado para la programación del algoritmo de control del robot.

3.1 Adquisición y Procesamiento de Señales

El principal objetivo de un software de adquisición de señales EEG se basa en filtrar de la manera más limpia posible este tipo de señales para su posterior procesamiento. Las señales de EEG capturadas están "contaminadas" con señales de movimientos musculares, es decir, cuando se miden señales EEG realmente se está midiendo una superposición de señales EEG y potenciales de activación muscular (EMG), y también señales EOG (Electrooculográficas, procedentes del movimiento del globo ocular que posee numerosas conexiones eléctricas).

Emotiv ha decidido aprovechar esta casuística que para otras empresas puede ser un problema.

Con los sensores que posee la diadema alrededor de la cara se pueden triangular las señales musculares y así construir sistemas de clasificación para identificar expresiones faciales como la sonrisa, el parpadeo…

En este proyecto, se estudiarán ambas opciones, el control del robot con comandos mentales, y el control del robot con expresiones faciales, veremos qué ventajas tienen cada uno, y analizaremos los resultados de cada sistema de control.

A continuación se hará una pequeña introducción a cerca de las redes neuronales más utilizadas y se

explicará el algoritmo que utiliza el software de Emotiv para el procesamiento de la actividad cerebral

3.1.1 Redes Neuronales

Las redes neuronales son un conjunto de algoritmos de aprendizaje automático que están diseñados para simular el funcionamiento de las neuronas del cerebro humano. Estas redes están compuestas por nodos interconectados, que procesan la información recibida y generan una respuesta. Cada nodo recibe una o varias entradas, que se multiplican por un peso asociado y se suman para generar una salida. Esta salida se aplica a una función de activación no lineal, que determina la respuesta del nodo. La salida generada por un nodo se conecta a otros nodos de la red, lo que permite que la información fluya y se procese a través de la red. Durante procesos de iteración, se ajustan los pesos de los parámetros en cada neurona buscando así el error mínimo al comparar la salida de la red con la salida esperada. Este proceso iterativo se conoce como entrenamiento de la red neuronal

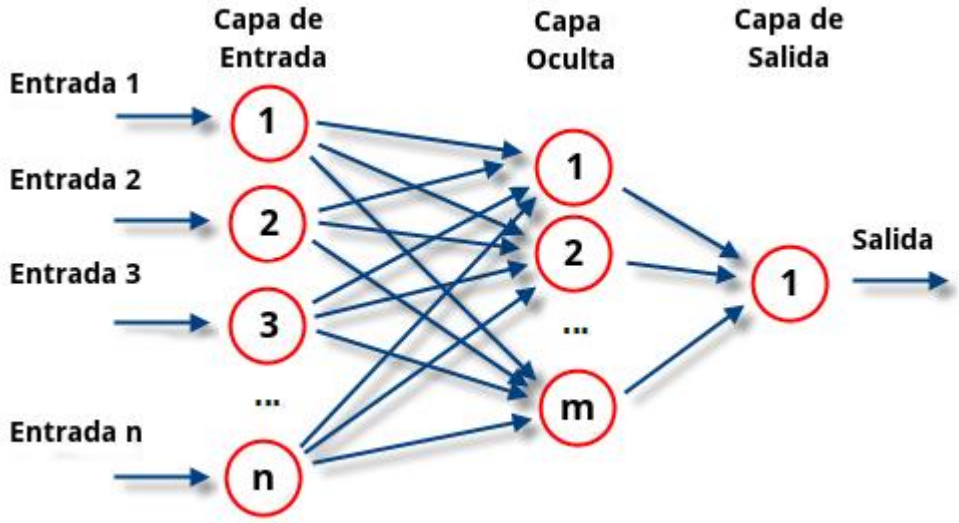

Figura 9: Esquema red neuronal [16]

Existen varios algoritmos para entrenar redes neuronales, entre los que se encuentran el algoritmo de retro propagación del error, el algoritmo de gradiente conjugado, el algoritmo de descenso del gradiente estocástico y el algoritmo de propagación hacia atrás a través del tiempo. Cada uno de estos algoritmos se utiliza para ajustar los pesos y parámetros internos de la red, de manera que la salida generada por la red sea lo más cercana posible a la salida esperada. El proceso de entrenamiento puede ser supervisado o no supervisado, dependiendo de si se cuenta o no con datos de salida esperados para la entrada correspondiente.

Una vez que se ha alcanzado el final de la red se obtiene una salida que será la predicción calculada por la red. Cuantas más capas posea la red y más compleja sea, también serán más complejas las funciones que pueda realizar.

3.1.2 Algoritmo utilizado por Emotiv

Para el caso que nos ocupa, la información que se ha podido encontrar que Emotiv ha proporcionado a cerca del software empleado para el entrenamiento de comandos mentales, indica que el software utiliza una red convolucional (CNN) para clasificar la actividad cerebral registrada durante el proceso de entrenamiento de comandos mentales.

Concretamente, Emotiv utiliza una variante de la CNN llamada Red Neuronal Convolucional de Estimulación Profunda (Deep Stimulation Convolutional Neural Network, DSCNN).

La característica de este algoritmo es la capacidad de procesar grandes cantidades de datos de EEG y aprender patrones complejos en la actividad cerebral del usuario, lo que permite una alta precisión en la clasificación de los comandos mentales.

Se ha podido encontrar que esta red está diseñada para trabajar de manera específica con señales de EEG y tiene en cuenta las características específicas de estas señales, como la variabilidad interindividual (las diferencias que existen entre las características de las ondas cerebrales de cada persona, dado que cada persona tiene una actividad cerebral única). Esto puede explicar por qué es necesario hacer un muestreo inicial de nuestra actividad cerebral cuando vamos a iniciar un entrenamiento.

A continuación, se explicará el funcionamiento básico de una CNN, la explicación se llevará a cabo tomando imágenes como ejemplos dado que es más intuitivo para la comprensión.

¿En qué consiste el algoritmo CNN?

Las CNN son un tipo de red neuronal profunda que se utiliza para el tratamiento de datos con estructura de matriz, como imágenes, vídeos, o señales EEG. Este tipo de red ha sido diseñada para trabajar con la estructura espacial de la imagen.

En una red neuronal multicapa, los datos se introducirían en un vector plano de pixeles:

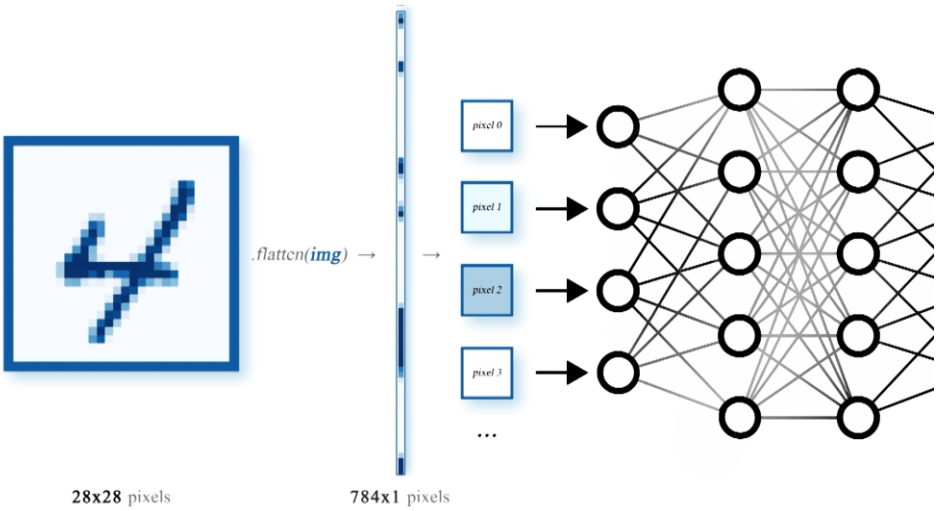

Figura 10: Procesamiento imagen red neuronal multicapa [19]

Lo que implicaría que no importa la posición que ocupe cada pixel en la imagen, la red trataría de igual forma estas otras imágenes:

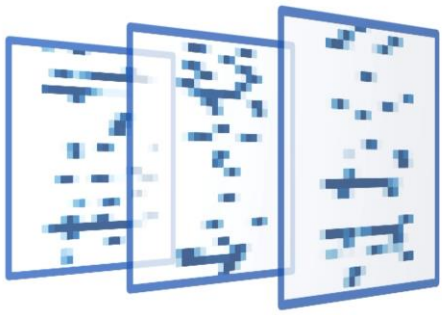

Figura 11: Ejemplos imagen input red neuronal [19]

Como un vector plano de pixeles, la realidad, es que para el tratamiento de imágenes, el valor que tome cada pixel va a estar condicionado por sus píxeles vecinos, esta relación entre pixeles, y que tengan una posición concreta en la imagen, es lo que permite tanto a nuestro cortex visual, como ahora a las redes neuronales convolucionales, identificar qué representa la imagen que se está analizando. [19]

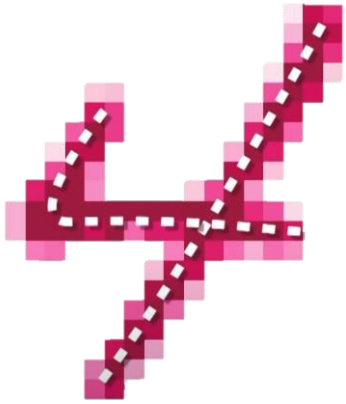

Figura 12: Imagen Input CNN [19]

La CNN es un tipo de red convolucional diseñada para extraer características de grandes matrices de datasets, esta extracción de características las realiza a través de operaciones matemáticas con **matrices de convolución.**

Convolución es el tratamiento de una matriz por otra llamada “filtro” o “kernel”.

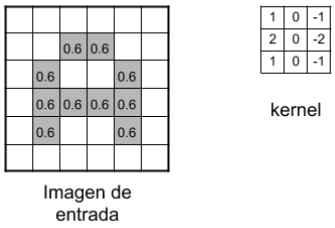

Figura 13: Imagen Entrada + Kernel (Convolución) [20]

El proceso de convolución consiste en ir aplicando la submatriz “kernel” sobre la matriz inicial y dar como resultado un nuevo pixel de la matriz tratada, cada pixel de la matriz tratada será el resultado de la operación realizada por el “kernel” sobre cada subconjunto de la matriz inicial.

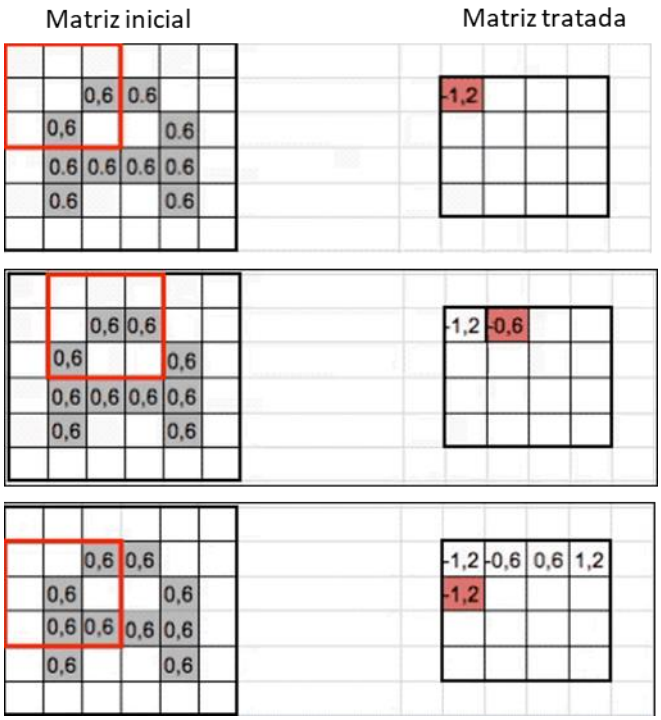

Figura 14: Operación Convolución [20]

El **filtro** que se utiliza para obtener la nueva matriz depende del resultado que se quiera conseguir, es decir, de las características que se estén buscando en la imagen.

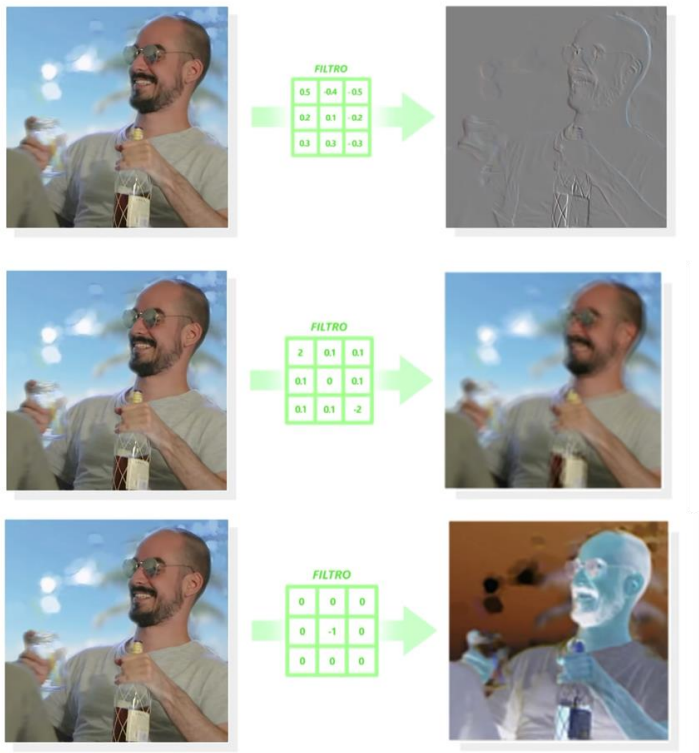

Figura 15: Ejemplos Operaciones Convolución [19]

Lo que debemos entender de este concepto, es que según el filtro que se aplique podremos extraer diferentes características de la imagen.

La elección de los valores del filtro no será una labor que debamos hacer nosotros, esta será, en esencia, la labor que realizará la Red Convolucional, que irá aprendiendo a ajustar dichos valores para lograr el objetivo que se busque en cada caso.

En una red convolucional, no solo se aplica un filtro por cada imagen o matriz de datos, se aplicarán tantos filtros como características diferentes se quieran encontrar.

Una primera capa convolucional, tendría la siguiente estructura:

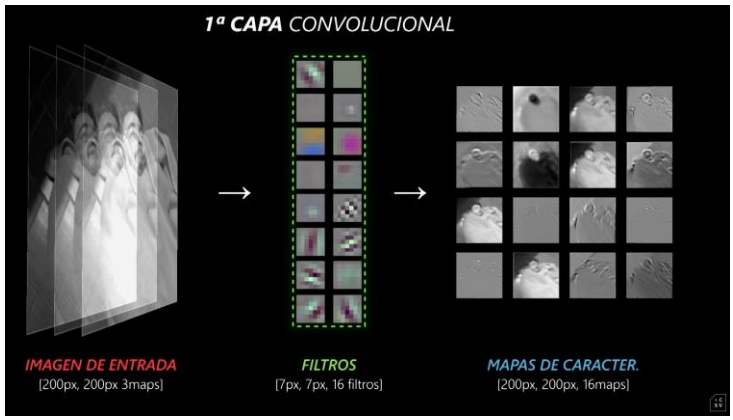

Figura 16: Capa Convolucional [19]

De una primera imagen de entrada, se crean **n** mapas de características (siendo **n** el número de filtros que hayamos aplicado). Estos **n** mapas de características, serán la entrada para la siguiente capa convolucional.

Tras realizar la convolución, se aplica una **función de activación no lineal** (p.e. ReLu (Rectified Linear Unit)), con el objetivo de introducir no linealidad en la red y permitir que aprenda relaciones y patrones complejos en los datos.

Tras el proceso de activación, se suele aplicar una capa de **pooling**, donde se busca reducir el tamaño de la matriz que va a ser tratada "eligiendo un pixel representante de sus vecinos", siendo este el que mejor preserva la característica buscada en la operación anterior.

Existen distintos tipos de "pooling", como "max pooling" o "average pooling":

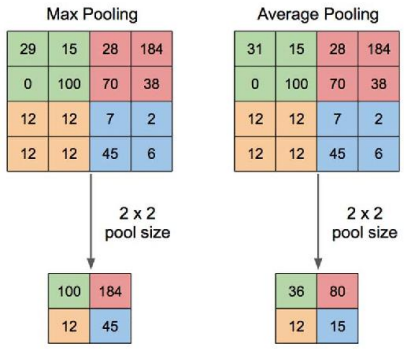

Figura 17: Tipos de pooling [21]

Al reducir el tamaño de las matrices, se combate la redundancia de información y se mejora la eficiencia computacional.

Para construir una primera capa convolucional, el procedimiento es el siguiente:

- **Padding**: no se ha mencionado hasta ahora, consiste en la extensión de los bordes de la matriz inicial con valores de 0 con el fin de evitar perder información de los verdaderos bordes.

- **Convolución**: se realizan **n** operaciones de convolución sobre la matriz inicial, dando como resultado **n** matrices de características, de menor resolución que la imagen actual, pero que almacenan información de manera más concentrada (almacenan características específicas)
- **Función de activación**: introducir no linealidad en la red y permitir un aprendizaje de relaciones complejas.
- **Pooling:** se reduce el tamaño de la matriz de características preservando la información más relevante [22]

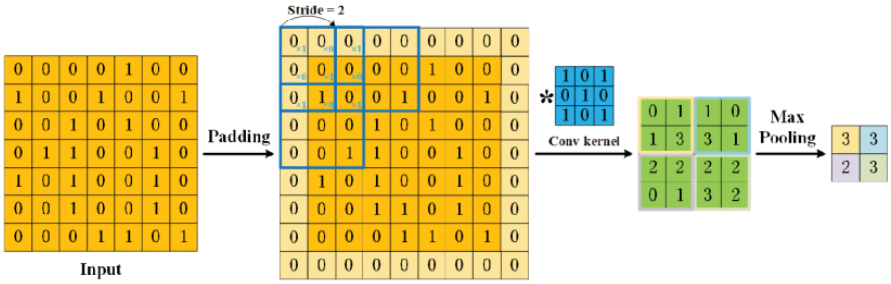

Figura 18: Capa Convolucional 2-D [22]

Para concluir con este capítulo sobre el algoritmo de las CNN, es interesante comentar la importante función que tiene la sucesión de las distintas capas, es aquí donde entra el concepto de “deep" del algoritmo.

Como ya se ha explicado, una matriz de características almacena en 1 pixel, información de **m**x**m** pixeles de la imagen inicial (siendo **m**x**m** la dimensión del filtro “kernel” utilizado), en la siguiente iteración, 1 pixel de la matriz de características almacena **p**x**p** pixeles de la primera matriz de características, que recordemos ya almacenaba **m**x**m** pixeles de la imagen inicial por cada pixel…

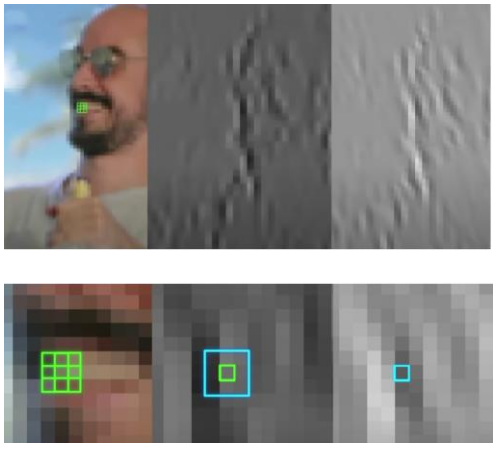

Figura 19: Resultado de varias convoluciones [19]

Es decir, cada vez se va almacenando más información en menos espacio, dando como resultado un proceso con estructura de embudo en el que cada vez tenemos menos resolución, pero más información. El proceso de CNN concluye con un proceso de clasificación a través de un conjunto de **capas completamente conectadas.**

La salida de esta red multicapa será dar la respuesta a qué era la imagen inicial.

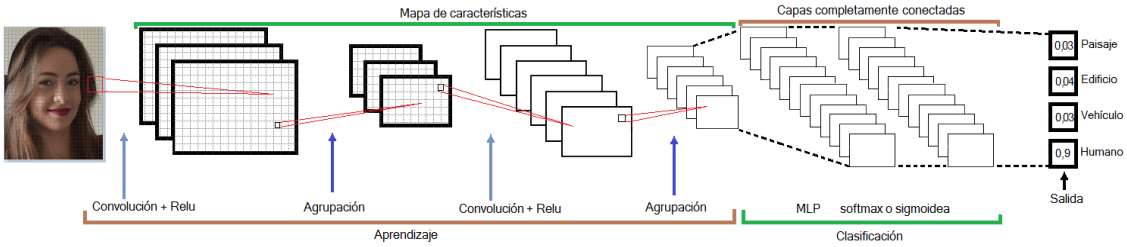

Figura 20: Capas CNN [23]

3.2 Configuración diadema EPOC X

Una vez conocida la tecnología que nos ocupa, procedemos a explicar los distintos pasos que hay que seguir para poner en marcha nuestra diadema. [4]

El dispositivo Emotiv que se ha utilizado para este proyecto es la diadema Epoc X.

Esta diadema cuenta con 14 canales EEG inalámbricos, cada uno asociado a un sensor de la diadema que permitirá obtener unos datos más precisos y completos de todo el cerebro.

Figura 21: Imagen Diadema Epoc X

Entre las características técnicas de la diadema Epoc X, podemos destacar:

- Conectividad bluetooth
- Batería recargable con capacidad de hasta 12 horas
- Banda rotatoria, que permite dos posiciones de la diadema.
- Sistema de rehidratación de sensores

Para iniciar la configuración de nuestro headset deberemos seguir los siguientes pasos:

Paso 1: Unboxing

Abrir el estuche de la diadema, en él, encontramos:

- Diadema Epoc X
- Bote de salinidad, con el que hidratamos los sensores de la Epoc X
- Un USB que permitirá la conexión vía bluetooth de la diadema con nuestro PC
- Cable de carga USB-C con el que cargaremos nuestra diadema
- Recambios de almohadillas para los sensores

Figura 22: Estuche Diadema EPOC X [4]

Paso 2: Carga de la diadema

Antes de usar nuestra diadema, debemos asegurarnos que está cargada, lo sabremos cuando el LED que hay a uno de los lados de la diadema se ponga de color verde.

Paso 3: Hidratación sensores

Este paso es muy importante, para un correcto uso de la diadema y una recepción fiable de la actividad cerebral es imprescindible conseguir una buena conectividad de los sensores. Para ello, deberemos hidratar de manera abundante los sensores.

Hay dos métodos para hidratarlos, para el primero, extraemos las almohadillas de la diadema y los colocamos en un recipiente con el líquido salino, dejamos que se impregnen de la solución y

posteriormente los escurrimos para extraer el sobrante. Acto seguido, volvemos a colocarlos en los sensores.

Otra manera de hidratarlos (esta es la que más se ha utilizado), es introducir directamente la solución salina por unos orificios que poseen los sensores que hacen que las almohadillas se impregnen de líquido. Cuando veamos que estas están húmedas, podremos pasar al siguiente paso. Será bastante común la rehidratación de los sensores pues pierden conectividad en frecuentemente.

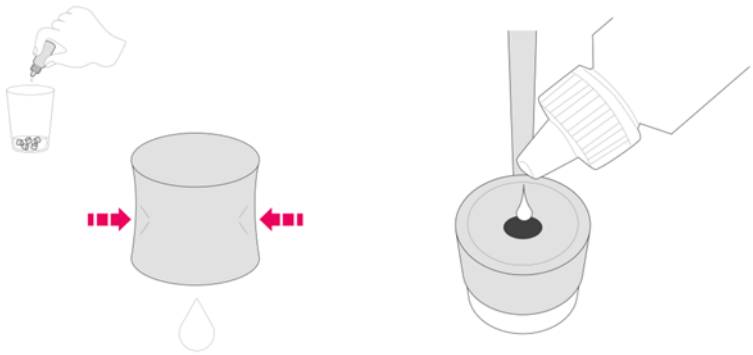

Figura 23: Dibujo Hidratación Sensores EPOC X [4]

Paso 4: Conectando Epoc X

Para conectar la diadema a nuestro PC, bastará con introducir en él el pincho USB, entonces se encenderá en él un LED verde. A continuación, pulsamos el botón de encendido de la diadema, escucharemos un pitido. En ese momento, un LED blanco se encenderá en la diadema y un segundo LED intermitente se encenderá en el USB indicando que se ha establecido la conexión.

Ahora que la diadema está configurada, se explicará la funcionalidad que tienen las distintas aplicaciones que el software de Emotiv permite descargar desde su página oficial.

3.3 Aplicaciones Emotiv

3.3.1 Emotiv Launcher.

Este software permite descargar en un solo paquete todas las herramientas que Emotiv pone a disposición del usuario de manera gratuita para la configuración de la diadema y el desarrollo de su uso. Dentro del paquete de instalación encontraremos; **EmotivPRO**, **EmotivBCI**, BrainViz, Virtual Brainwear, y el propio **EmotivLauncher.**

Este software podemos descargarlo para macOS y Windows, en su versión más completa, y para Ubuntu y Raspberry Pi es una versión Beta.

Cuando abrimos la aplicación, lo primero que debemos hacer es iniciar sesión con nuestro EmotivID, si no poseemos una cuenta deberemos crearla, para ello, es importante que tengamos localizado el ID de la diadema, el cuál podremos encontrar en el estuche de la misma, en un documento donde vienen sus especificaciones técnicas.

Una vez se ha iniciado sesión, con la diadema encendida, comprobamos que la aplicación la haya detectado y entonces pulsamos el botón 'Connect' y esperamos la conexión. Ya tendremos conectada nuestra diadema.

La conexión de la diadema se puede realizar tanto desde la aplicación del launcher, como desde cualquier otra aplicación de Emotiv. [5]

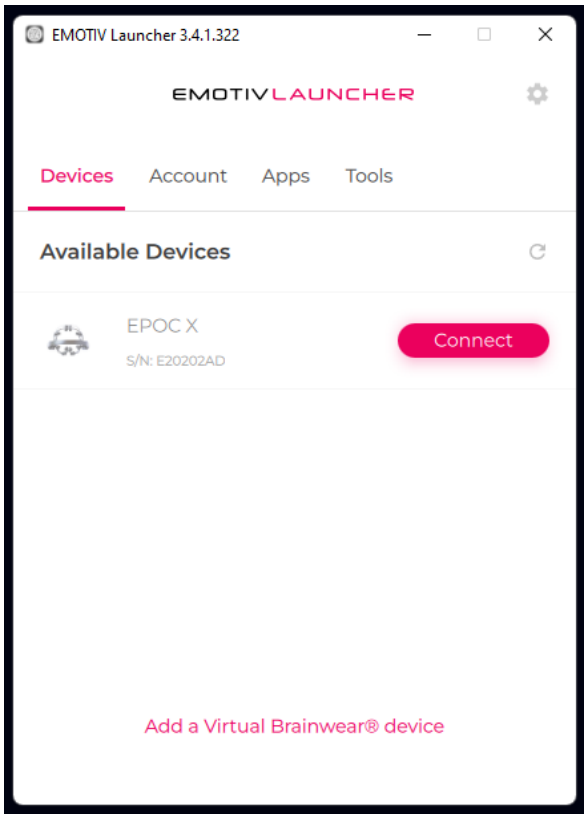

Figura 24: Pantalla Inicio Aplicación Emotiv Launcher [5]

3.3.2 EmotivPRO

Esta aplicación permite visualizar a tiempo real los datos que envía la diadema, en ella podremos ver las gráficas de los distintos canales y grabar algunas secuencias en las que hagamos algún movimiento específico como puede ser parpadear, hacer alguna mueca, y así analizar los diferentes comportamientos según las acciones.

3.3.3 EmotivBCI

→ Configuración Headset

Esta aplicación será de gran importancia para el desarrollo de nuestro proyecto, pues es el entorno que nos permitirá de una manera cómoda y sencilla entrenar los comandos mentales que en el futuro utilizaremos para controlar nuestro robot.

Cuando abrimos la aplicación lo primero que debemos hacer después de conectar la diadema será verificar que todos los sensores poseen una buena conectividad.

Si los sensores aparecen en gris significa que no hay un buen contacto entre el sensor y la cabeza y/o que el sensor no está bien hidratado

Si los sensores aparecen en rojo o naranja significa que el contacto no es del todo bueno, para ello probamos a dar algunos toques sobre el sensor, moverlo levemente, o si hiciera falta volver a hidratarlos.

Este proceso puede llevar un tiempo puesto que son muchos sensores y hay algunos que son más difíciles de conectar que otros.

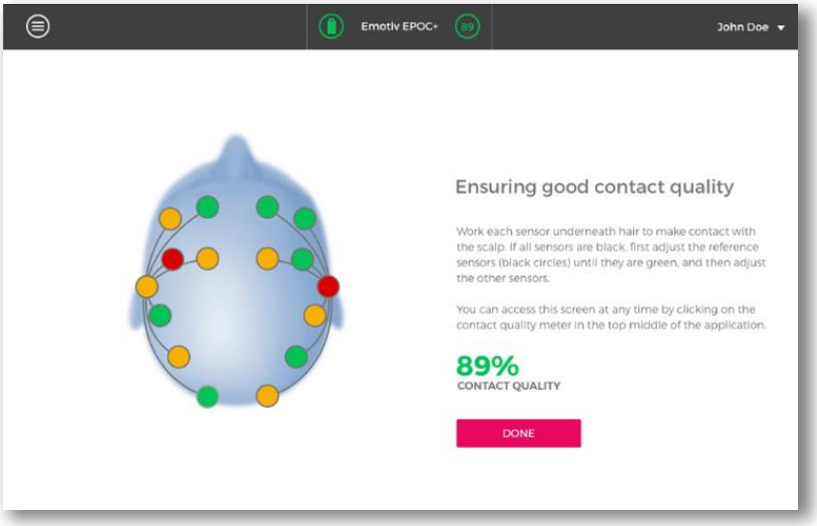

Figura 25: EEG quality

Cuando hayamos logrado tener todos los sensores en verde podremos pasar al siguiente paso, que consiste en comprobar la "EEG quality", este parámetro mide cómo es la actividad cerebral que debería estar recibiendo cada sensor y la compara con la actividad cerebral que estén recibiendo los nuestros. Por norma general, si intentamos relajarnos y quedarnos quietos unos segundos la "EEG quality" debería ser superior al 90%. Este es el porcentaje que recomienda Emotiv para poder optimizar al máximo nuestra experiencia con el entrenamiento de comandos mentales.

→ Perfiles

Una vez conectada y configurada la diadema, se abrirá un panel de perfiles, que de inicio estará vacío. La aplicación permite tener al usuario varios perfiles de entrenamiento.

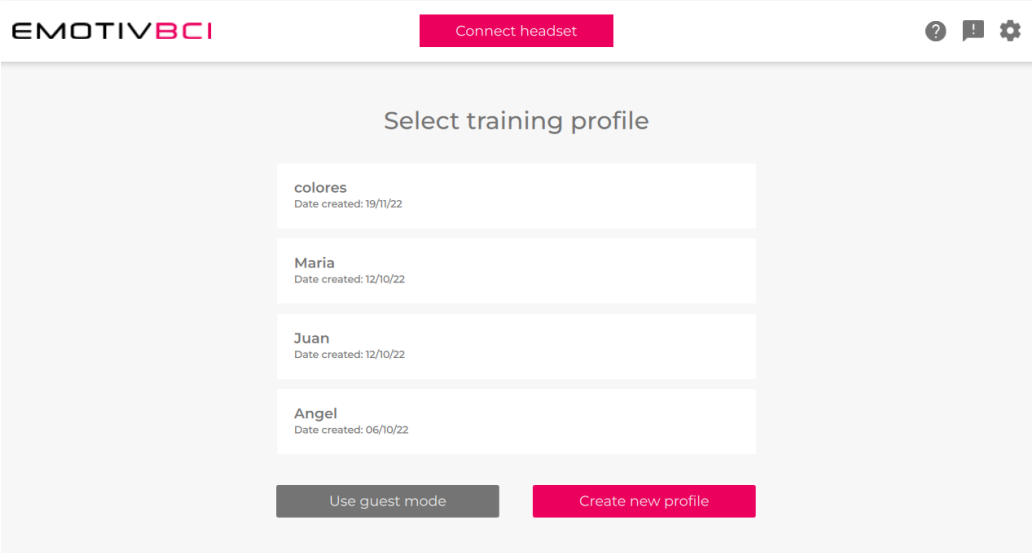

Figura 26: Ventana Gestión Perfiles | EmotivBCI [5]

→ Modos de entrenamiento

En este apartado se describen de manera resumida los distintos modos de entrenamiento que el usuario podrá realizar.

- **Mental commands:**

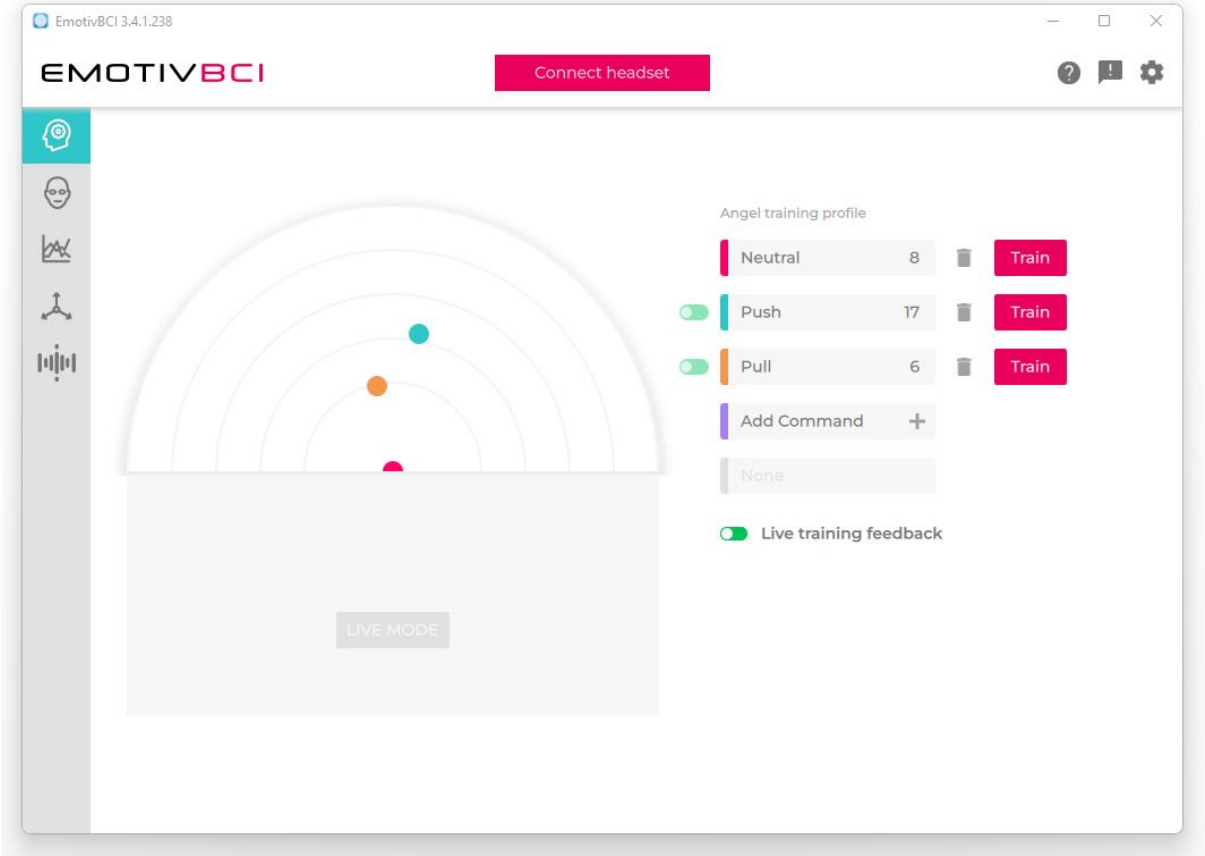

Figura 27: Interfaz Emotiv BCI | Entrenamiento Mental 1

La primera opción es el entrenamiento de comandos mentales, como se puede apreciar en la imagen, la ventana muestra una gráfica en forma de semicírculo (a priori vacía), donde se irán posicionando las acciones que vayamos entrenando.

El software de Emotiv permite el entrenamiento de hasta 4 acciones diferentes (sin contar la acción "neutral")

Antes de iniciar un entrenamiento el software nos hará un reconocimiento de la actividad cerebral que tengamos ese día, con objeto de ajustar el entrenamiento a nuestro estado. Esto lo realiza pidiendo que permanezcamos 30 segundos relajados con los ojos abiertos, y 30 segundos relajados con los ojos cerrados. Tras esto, podremos comenzar con el entrenamiento.

El primer comando que debemos entrenar es el "neutral", puesto que servirá de referencia para el resto de acciones. Para entrenar este comando bastará con estar relajados.

Para el entrenamiento del resto de acciones, la mecánica que seguiremos será la siguiente:

- Decidir qué pensamiento vamos a asociar a dicha acción.
- Entrenar tantas veces como podamos la acción pensando en aquello que hemos escogido.

Si lo estamos haciendo bien, y hemos elegido un pensamiento que se diferencie bien del resto de acciones, podremos observar como las acciones (representadas como puntos de colores en la gráfica) se van separando del punto central (que representa la acción "neutral")

Cuánto más entrenemos una acción, más robusto será el algoritmo y más preciso seremos a la hora de controlar la acción que queremos realizar.

Para facilitar el entrenamiento de los comandos al usuario, Emotiv muestra en cada entrenamiento la imagen de un cubo virtual, que se moverá por la pantalla según la acción que estemos entrenando (push: el cubo se aleja, pull: el cubo se acerca…)

Una vez que hayamos entrenado las acciones podremos comprobar cómo de bien lo estamos haciendo probando el "Live Mode" de la ventana.

Más adelante veremos cómo es realmente este proceso, las distintas opciones de pensamiento que más se recomiendan, y las dificultades que se han ido encontrando a medida que se entrenaban los comandos.

· Facial Expressions

Este modo es similar al anterior, en este caso podemos entrenar distintas expresiones faciales, como pueden ser sonreír, fruncir el ceño, subir las cejas, parpadear… Para facilitar el entrenamiento al usuario, al iniciar un entrenamiento (p.e: sonreír) aparecerá una cara virtual sonriendo, y así con el resto de comandos.

Al igual que en 'Mental Commands' este apartado posee un 'Live Mode' que nos permitirá observar cómo la cara virtual copia nuestras expresiones faciales.

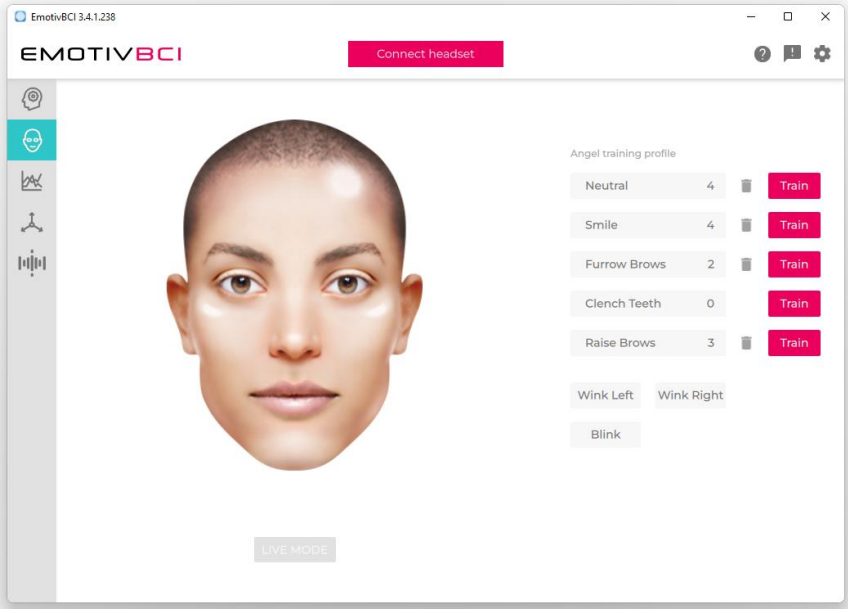

Figura 28: Interfaz Emotiv BCI | Entrenamiento facial

Las expresiones faciales también podrían ser una herramienta potencial para el control de nuestro robot.

· **Performance Metrics:** En esta sección, se pueden ver graficados los distintos estados mentales (estrés, concentración, relajación…)

En la página de Emotiv, se explica cómo se recopilan los datos para realizar las detecciones:

Las detecciones se basan en varios conjuntos de datos experimentales obtenidos de diferentes voluntarios en sus experimentos.

Por ejemplo, para el entrenamiento del estado "Frustración", se llevó a cabo un experimento con más de 30 sujetos de género mixto en el que, equipados con dispositivos EEG, además de monitoreo de respiración, GSR y esfigmomanómetro, se les ponía ante distintos juegos diseñados para generar frustración, por ejemplo, el juego PacMan, donde fallaban los controles o se invertían estos al azar para generar esa frustración en el usuario. [15]

La empresa no explica mucho más a cerca de sus métodos de entrenamiento, aunque si habla de que no todas las detecciones dependen de datos previos tomados en sus experimentos. También se realizan entrenamientos para cada individuo y se entrena al software, como es el caso de los comandos mentales.

· **Motion sensors:** En esta sección se representan en gráficas los movimientos que podamos realizar con la cabeza. Esto es posible gracias al acelerómetro y giróscopo que tiene la diadema integrados.

3.4 Cortex API

Con las aplicaciones que vienen instaladas por defecto al descargar el paquete de Emotiv, el usuario puede tener una primera toma de contacto con el software, ver las gráficas de las ondas generadas por el cerebro, realizar grabaciones ejecutando distintos patrones que generen picos de actividad cerebral, o el más importante en este proyecto, entrenar comandos mentales y expresiones faciales para el control de un robot. Para dar el siguiente paso, que es utilizar esos datos de entrenamiento para enviar las acciones al robot, necesitamos hacer uso del "Cortex API", un servicio de Emotiv pensado para la creación de aplicaciones que permitan a la Diadema de Emotiv interactuar con un tercero (en nuestro caso, un robot)

El "Cortex API" está creado con Websockets y utiliza el protocolo JSON.

Los requisitos para utilizar este servicio son:

- Crear una cuenta Emotiv: si ya hemos utilizado otras aplicaciones de Emotiv como EmotivPRO o EmotivBCI ya habremos creado nuestra cuenta. Si no, podemos crear una cuenta aquí.
- Licencia: con el EmotivID predeterminado tendremos acceso a la mayoría de servicios que ofrece la aplicación, por tanto no será necesario introducir ninguna licencia.
- Crear una App de Cortex: vamos a crear un ID de aplicación, que usaremos como nube personal para recibir los datos de la diadema, para ello, deberemos ir a la página de Emotiv (www.emotiv.com) e iniciar sesión, después ir al apartado de My Account Dashboard, y seguir los pasos indicados para crear una Application ID. Una vez hecho esto se nos asignará un ID de cliente y una contraseña (que sólo aparecerá una vez y que debemos guardar) que utilizaremos para "iniciar sesión" en nuestra App y permitir la transmisión de datos.

A continuación, hablaremos de los distintos ejemplos que "Emotiv Cortex" ha creado donde se desarrollan las principales funcionalidades de este servicio.

Ejemplos de Cortex

Emotiv pone a disposición del usuario un repositorio de GitHub con varios ejemplos donde se pueden ver las capacidades de la API de Cortex. Estos ficheros me han sido de vital ayuda para poder desarrollar este trabajo, dado que los he utilizado como esqueleto para crear el código en Python que se suscriba a los datos enviados por la diadema, extraiga las acciones requeridas, y las envíe por comunicación serial a Arduino para que el robot se mueva.

Nota: es importante mencionar que para poder ejecutar los siguientes ejemplos es necesario instalar la librería websocket-client, para ello, en nuestro terminal deberemos ejecutar la siguiente línea de código:

```
PS C:\Users\angel> pip install websocket-client
```

Si descargamos el fichero de GitHub con los ejemplos podremos encontrar algunos como:

· Cortex.py

Este fichero es uno de los más importantes, es el cerebro de la comunicación BCI. Aquí se diseña la clase "cortex()", que se importará más adelante en los otros ficheros que serán los que desempeñen una función específica.

En este fichero se crea el socket que establecerá la comunicación entre el usuario y la 'Cortex App' que recordemos será la que nos enviará los comandos y las señales que queramos obtener de la diadema. Además se han creado todas las funciones que permitirán poder hacer uso de las distintas funcionalidades del servicio de Cortex.

Este fichero realizará acciones como:

- Crear el socket de comunicación que permitirá la transmisión de datos entre la diadema y el PC. Para ello, debemos conectarnos a **localhost** mediante el puerto **6868**. En el fichero 'cortex.py'. Aparece así por defecto.
- Cargar el perfil de entrenamiento que hayamos seleccionado.
- Conectar y desconectar la diadema
- Solicitar los accesos necesarios para acceder a los datos de la diadema.
- Tratar los datos y cambiar su formato para que sea más fácil manejarlos
- Reportar errores que se pudieran dar durante la comunicación.

Este fichero no lo tocaremos, pero nos será de gran ayuda pues lo importaremos en nuestro fichero para desarrollar nuestra aplicación.

Para más información acerca del "Cortex API" podemos acceder a la guía de Emotiv.

· mental_command_train.py

En este fichero se crea una clase denominada "Train()" que como su propio nombre indica permitirá al usuario entrenar distintos comandos mentales. En él podremos fijar qué comandos queremos entrenar, para correr el fichero, deberemos introducir el ID de cliente y la contraseña que se nos haya asignado al crear la "Cortex App". También deberemos introducir el nombre del perfil donde vamos a ir guardando los entrenamientos, este puede ser un perfil nuevo, o bien uno existente que ya hayamos estado utilizando por ejemplo en la aplicación de EmotivBCI.

```
269  def main():
270
271      # Please fill your application clientId and clientSecret before running script
272      your_app_client_id = 'x3CQmJdyIhYSz04aR3JcvgsBVBzHpbon6rnJc5z9'
273      your_app_client_secret = 's4dPeGHZKT4BZTixNlbSAxjhlkGJnVWx0b6581jwDzBYNdQ2u97qWM57IzU146dQaeC4g05AKmzVkuL25vXzwLv3dZKiPD6AMZXXLthLqKWaEmeVH93kp6mrQtTjHT1E'
274
275      # Init Train
276      t=Train(your_app_client_id, your_app_client_secret)
277
278      profile_name = 'angel_1' # set your profile name. If the profile is not exited it will be created.
279
280      # list actions which you want to train
281      #actions = ['neutral', 'push', 'pull']
282      actions = ['push']
283      t.start(profile_name, actions)
284
285  if __name__ =='__main__':
286      main()
287  # ----------------------------------------------------------
```

Código 1: 'main' mental_command_train.py

Podemos utilizar este fichero para entrenar comandos pero la realidad es que el entrenamiento de los comandos se realiza de una manera mucho más cómoda desde la aplicación de EmotivBCI. Por tanto este fichero no será de gran importancia para nuestro trabajo.

· live_advance.py

Este fichero crea un entorno de "live mode" donde podemos comprobar cómo hemos entrenado nuestros comandos mentales, cuando lo iniciamos, se irá imprimiendo por pantalla las acciones que estemos ejecutando en ese momento. A partir de este fichero crearemos nuestro código para extraer los comandos que necesitamos y enviárselos al robot. Por ello le dedicaremos un punto más desarrollado más adelante.

Otros ficheros incluidos en el repositorio son: facial_expression_train.py (para entrenar comandos faciales), record.py (para hacer grabaciones) y sub_data.py (archivo que se suscribe a distintos datos que emite la diadema), entre otros.

4 ROBOT CURRO

"Con mucha diferencia, el mayor peligro de la Inteligencia Artificial es que las personas concluyen demasiado pronto que la entienden".

-Eliezer Yudkowsky-

En este capítulo, presentaremos de manera detalla al robot con el que se ha realizado el proyecto, las características físicas de este, así como una breve explicación de la programación de los diferentes componentes electrónicos que permiten dar vida a nuestro robot.

4.1 Características físicas.

Curro es un robot de tracción diferencial, es decir, un vehículo que utiliza un sistema de transmisión de dos ruedas independientes (cada rueda tiene un motor). En consecuencia su movimiento se basa en la diferencia de velocidades de las dos ruedas situadas en un mismo eje. Además, el robot incluye una rueda libre o loca que gira de manera pasiva, y sirve para dar estabilidad al robot.

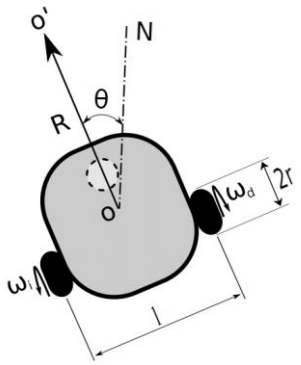

Figura 29: Esquema Robot Diferencial

A continuación muestro una tabla con las principales características físicas del robot.

Altura	37.8 - 46.0cm (según inclinación cresta)
Ancho	23.8cm
Largo	24.3cm
Peso	2.72 kg
Diametro Ruedas	13.5 cm
Distancia entre ruedas	19.0cm

Tabla 1: Características físicas robot [6]

Figura 30: Numeración Cuerpo "Curro"

Las partes numeradas de la imagen quedan definidas a continuación:

1.- Cabeza
2.- Cuello
3.- Cuerpo
4.- Cresta
5.- Ceja derecha
6.- Ceja izquierda
7.- Cámara Raspberry Pi
8.- Anillo LED
9.- Micrófono
10.- Sensor ultrasónico
11.- Abertura para conexión de USB a Arduino
12.- Abertura para conexión de Jack a Arduino.
13.- Rueda izquierda
14.- Rueda derecha
15.- Abertura puertos de conexión Raspberry Pi
16.- Interruptor basculante
17.- Rueda loca
18.- Altavoz
19.- Engranaje para rotación del cuerpo
20.- Sensor Shield V5
21.- Motor Shield L298P
22.- Arduino Uno
23.- Raspberry Pi 4B

El robot ha sido creado mediante impresión 3D con un material ABS (Acrylonitrile Butadiene Styrene) que es muy usado en la industria de la impresión 3D y que aporta al robot buena rigidez, resistencia a impactos y a su vez un peso moderado.

4.2 Software y componentes electrónicos.

En este apartado se describirán tanto el software empleado para la programación del robot como los componentes electrónicos que lo componen. También se explicará de manera breve como programar cada uno de los componentes para así tomar un primer contacto con ellos.

4.2.1 Software Arduino

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para creadores y desarrolladores.

Figura 31: Logo Arduino

Arduino IDE es la interfaz de programación que Arduino pone a disposición del usuario para el desarrollo de los proyectos. Está escrita en lenguaje Java y es compatible con Windows, Linux y MacOS, el lenguaje de programación que utiliza es C++. El software es gratuito y se puede descargar en el siguiente enlace.

Una de las grandes ventajas de Arduino, además de su enorme comunidad que permite encontrar solución a casi todos los problemas o ideas que te puedan surgir, es la cantidad de librerías, desarrolladas por Arduino o por otro usuarios, que podemos utilizar en nuestro proyecto para dar vida a nuestro robot, dotarlo de funcionalidades de movimiento, control por voz, etc. Además, contamos con la característica de la comunicación por puerto serial, Arduino ofrece la posibilidad de trabajar con el puerto serial, de manera que podamos comunicarnos con la placa que estemos utilizando para mostrar, enviar o recibir mensajes, y sobre todo para poder establecer comunicación con otros dispositivos como la Raspberry Pi. Esta característica permitirá la comunicación entre Arduino y la Raspberry Pi y será un punto fundamental en el desarrollo del proyecto.

4.2.2 Placa Arduino Uno

Toda la programación relacionada con los movimientos y acciones que el robot puede desempeñar van a estar programadas en una placa Arduino Uno que será el cerebro de nuestro animatrón.

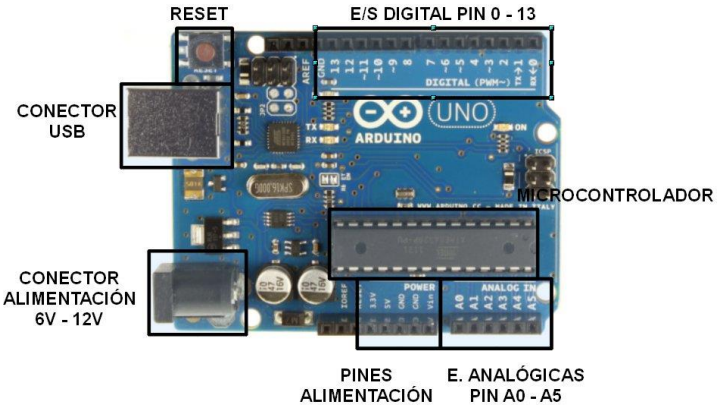

Figura 32: Esquema placa Arduino UNO [8]

Esta placa es una de las más utilizadas debido a su tamaño y versatilidad. Además, todas las librerías de Arduino están diseñadas para poder funcionar en esta placa.

Las principales características de la placa se muestran a continuación:

Características Placa Arduino UNO	
Microcontrolador	Atmel ATmega328 a 16MHz
Memoria SRAM	2Kb
Memoria EEPROM	1Kb
Memoria flash	32Kb (0.5 dedicados al arranque)
Límites de voltaje entrada	6-20 V
Voltaje entrada recomendado	7-12 V
Intensidad de corriente para E/S	40 mA
Nº Pines E/S	20
Nº Pines alimentación 5V	1
Nº Pines alimentación 3.3V	1
Nº Pines GND	2
Dimensiones	6.9cm x 5.3cm
Peso	25g

Tabla 2: Características Placa Arduino UNO [8]

De los 20 pines E/S que posee la placa, 14 son digitales (de los cuales 6 son PWM [3, 5, 6, 9, 10 y 11]) y 6 analógicos.

La placa posee además un botón de reseteo (para reiniciar el programa), y varios LEDs, entre ellos el indicador de encendido de la placa

4.2.3 Módulo controlador de motores L298P

Esta placa Motor Shield L298P es ideal para el control de motores DC, STEPPER, SERVO, etc. En nuestro caso la utilizaremos para el control de los servomotores de las ruedas, dado que nos permitirá controlar la velocidad y la dirección de ambos motores de manera independiente.

Para el manejo del robot, usaremos los pines 10, 11, 12 y 13. Donde los pines 10 y 11 irán dedicados al control PWM de las velocidades de los motores, y los pines 12 y 13 para el sentido de giro de estos. [6]

- La alimentación que recibe esta placa será directamente de la fuente de alimentación, dado que los motores de las ruedas necesitan más alimentación que los 5V del resto de actuadores.

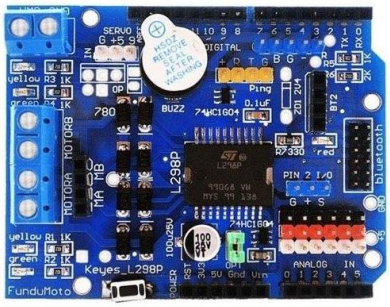

Figura 33: Módulo controlador de motores L298P

4.2.4 Módulo Sensor Shield V5.

Esta placa de expansión para Arduino UNO está pensada para conectar fácilmente multitud de módulos de expansión (sensores, actuadores, motores, etc.) a nuestro controlador. En nuestro caso la utilizaremos para controlar el resto de actuadores del robot (los servos, el micrófono, el sensor ultrasónico, la tira de leds) de una manera sencilla, puesto que no podremos hacerlo directamente en la placa de Arduino ya que el acceso a sus pines estará limitado debido a la conexión del motor shield. Por lo tanto, esta placa funciona como expansión de dichos pines.

- El escudo tiene unas dimensiones de 5.8cm x 5.8 cm.
- La alimentación que recibe esta placa es de 5V.

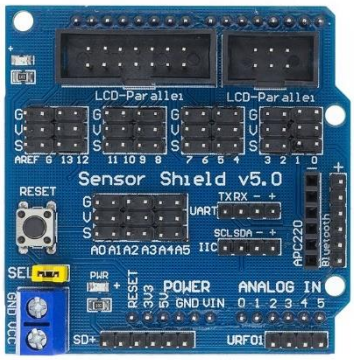

Figura 34: Modulo Sensor Shield V5 [9]

4.2.5 Raspberry Pi modelo 4B

La Raspberry Pi es un ordenador de bajo coste y formato compacto que junto con la placa de Arduino UNO, forman el núcleo de nuestro robot. Esta placa posee funcionalidades de desarrollo de programas de electrónica y robótica (permite conexión de sensores, actuadores…) como una placa de Arduino, pero además permite la instalación de un SO de manera que cubre todas las funcionalidades de un ordenador.

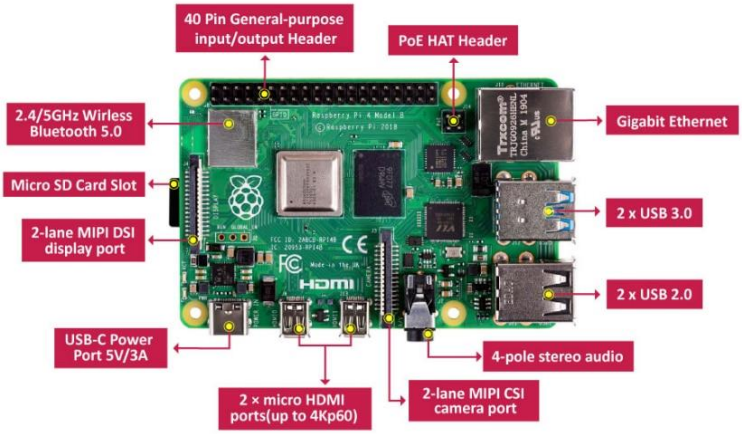

Figura 35: Raspberry Pi 4B [18]

El uso principal que le daremos a esta placa será la de conexión entre la placa de Arduino (mediante un puerto serial) y la diadema de Emotiv, vía Wi-Fi. Algunas de las características de esta placa se muestran a continuación:

Dimensiones:

- 8.6 cm x 5.7 cm

Peso:

- 45g

Conectividad:

- Conexión Wi-Fi Dual Band 2.4GHz y 5.0GHz
- Conexión Bluetooth 5.0
- Posee un puerto Ethernet, 2 puertos tipo USB 2.0 y otros 2 tipo USB 3.0 y 40 GPIO (General Purpose Input/Output)

Opciones multimedia:

- 2 puertos micro-HDMI capaces de reproducir a 4K 60fps
- 1 puerto para conexión de cámara Raspberry PI
- 1 puerto para conexión de pantalla MIPI DSI
- 1 puerto 4-pole para audio y vídeo

En la sección de Anexos se dedicará un punto a la instalación de un sistema operativo en la Raspberry PI.

4.2.6 Cámara Raspberry Pi V2 8MP

Esta es la cámara oficial de Raspberry Pi, tiene un sensor Sony IMX219 de 8 megapíxeles y es compatible con todas las tarjetas Raspberry Pi. Es uno de los periféricos más usados debido a su fácil conexión y programación en Python, donde encontraremos librerías dedicadas a esta cámara que permitirán al usuario tomar fotos y videos.

Posee además la capacidad de modificar el brillo, la apertura, la saturación, simulación de efectos como modo noche, modo soleado, entre otros…

En este proyecto no se ha utilizado la cámara del robot, queda como tarea pendiente para futuros proyectos relacionados con el robot, al que se le podría implementar un reconocimiento facial que detectara la persona que va a controlar el robot, para así poder cargar su perfil de comandos mentales.

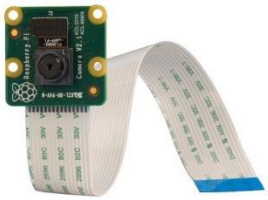

Figura 36: Cámara Raspberry Pi V2 8MP

Aunque no se haya usado la cámara en este proyecto, adjunto a continuación un enlace a un vídeo que muestra cómo realizar la conexión.

4.2.7 Anillo LED: Sparkfun neopixel 24x WS2812B RGB.

A través del anillo LED, mostraremos los movimientos del robot, así como los distintos estados por los que este pase.

Figura 37: Anillo NeoPixel, 24 LED's – WS2812 RGB [10]

Este es el Anillo NeoPixel de 24 LEDs de Adafruit, una pequeña tarjeta de diámetro externo de 2.66″(66mm),equipada con LED's 5050 WS2812 RGB.
Los WS2812s son direccionables cada uno, ya que el chip del controlador se encuentra dentro del LED. Cada Stick NeoPixel tiene una corriente constante de ~ 18mA, de esta manera el color será muy constante incluso si el voltaje varía, requiere 5V. [10]

El anillo posee una dimensiones de 6.6cm de diámetro.

4.2.8 Sensor ultrasónico HC-SR04.

El sensor ultrasónico situado en la parte frontal del robot le permitirá calcular la distancia al objeto más cercano y así poder frenar en caso de que llegue a una distancia establecida.

Figura 38: Sensor Ultrasónico HC-SR04

El sensor contiene 4 pines:

- Pin Vcc: alimentación sensor
- Pin Trig: pin envío señal ultrasónica
- Pin Echo: pin receptor de señal
- Pin GND: conexión a tierra.

El funcionamiento del dispositivo es sencillo, el transmisor envía una onda de sonido, dicha onda rebotará en el objeto más cercano y será recogida por el receptor, que es el módulo que se encuentra junto al transmisor. Calculará entonces el tiempo (t) que ha transcurrido desde que se envió la onda hasta que es recibida, posteriormente, conociendo que la velocidad del sonido (v_{sonido}) es de 340m/s. La distancia (D) al objeto más cercano será $D = v_{sonido} * \frac{t}{2}$.

Puede darse el caso que el receptor no reciba ninguna onda, esto será símbolo de que no hay ningun objeto cerca.

Este modelo tiene un rango de distancias sensible entre 3cm y 3m con una precisión de 3mm.

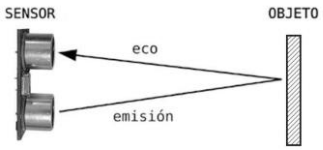

Figura 39: Funcionamiento Sensor Ultrasónico [2]

4.2.9 Micrófono KT-038.

El robot tiene integrado este micrófono capaz de detectar sonidos, nosotros lo utilizaremos para dar órdenes al robot con palmadas o chasquidos.

Figura 40: Micrófono KT-038

El micrófono entrega una señal digital (si solo queremos detectar sonido) y una analógica (si nos interesa el volumen de este y las características).

Cuenta con 2 LED's, uno que indica suministro de energía (L1) y otro que indica la recepción o no de sonido (L2).

Posee además un potenciómetro que permite regular la sensibilidad del micrófono.

El dispositivo posee 4 pines de conexión:

- Pines A0 y D0: indicarán un valor HIGH, si el sonido recibido es superior al umbral indicado en el potenciómetro, de forma analógica o digital, respectivamente.
- Pin G: conexión a tierra.
- Pin +: alimentación. [6]

Características Micrófono KY-038	
Tensión alimentación	5V
Chip principal y micrófono	LM393 y electret
Gama de frecuencias que reconoce	10 a 10000 Hz
Sensibilidad mínima la ruido	58 dB
Dimensiones	36 x 15 x 15 mm
Peso	4g

Tabla 3: Características técnicas micrófono KY-038 [3]

4.2.10 Mini altavoz mp3.

Este altavoz mp3 se usará para reproducir algunos sonidos con el robot. Su capacidad no es muy amplia pero es de muy fácil programación y cubre las necesidades que tenemos, véase reproducir algunas notas de alguna canción que asociemos a un estado del robot.

Tiene una potencia de 2W y requiere una alimentación de 5V. Irá integrado en el interior del robot.

Figura 41: Mini altavoz mp3

4.2.11 Servomotor Turnigy TGY50090.

Figura 42: Servomotor Turnigy TGY50090

Este modelo de servo se utilizará para las cejas (1 para cada ceja). Tienen capacidad de giro de 180º

Sus características más relevantes se muestran a continuación:

Características Servomotor Turnigy TGY50090	
Tensión alimentación	4.8V - 6V
Par a 4.8V	1.6 kg/cm
Par a 6V	2.0 kg/cm
Velocidad a 4.8V	0.08 sec/60º
Velocidad a 6V	0.07 sec/60º
Peso	9g
Dimensiones	23.1 x 12.0 x 25.9 mm

Tabla 4: Características servomotor Turnigy TGY50090 [3]

4.2.12 Servomotor Futaba 3003.

Este servomotor se encargará del movimiento de la cresta del animatrón. Estamos ante un servo analógico capaz de rotar 180º.

Figura 43: Servomotor Futaba 3003 [4]

Sus características más relevantes se muestran a continuación:

Características Servomotor Futaba 3003	
Tensión alimentación	4.8V - 6V
Par a 4.8V	3.17 kg/cm
Par a 6V	4.10 kg/cm
Velocidad a 4.8V	0.23 sec/60º
Velocidad a 6V	0.19 sec/60º
Peso	37g
Dimensiones	39.9 x 20.1 x 36.1 mm

Tabla 5: Características Servo Futaba 3003 [4]

4.2.13 Servomotor HXT12K M 11Kg.

Figura 44: Servomotor HXT12K M 11Kg [5]

Este último modelo de servomotor es el más potente de todos, lo usaremos en 4 ocasiones: 1 para el cuello, 1 para la rotación del robot, 2 para las ruedas de Currito.

Para las 2 primeras aplicaciones, el servo estará conectado a un trío de pines (señal, alimentación , tierra) por lo que podremos controlar la posición que queramos que alcance, tendrá un rango de giro

de 180º. Para las ruedas, los servos irán conectados al shield de motores, un pin digital controlará el sentido de giro y otro analógico (conectado a PWM) controlará la velocidad del servo. En este caso los servos podrán girar 360º.

Características técnicas:

Características Servomotor HXT12K M 11Kg	
Tensión alimentación	4.8V - 6V
Par a 4.8V	9.4 kg/cm
Par a 6V	11.0 kg/cm
Velocidad a 4.8V	0.20 sec/60º
Velocidad a 6V	0.16 sec/60º
Peso	55g
Dimensiones	40.7 x 19.7 x 42.9 mm

Tabla 6: Características Servomotor HXT12K M 11Kg [11]

Se muestra a continuación un resumen de las características de los distintos motores que contiene el robot, las partes móviles a las cuáles están asociados, y los rangos de pulsos mínimo y máximo que cada motor tiene y que permitirá controlar su movimiento en un rango óptimo.

ELEMENTO MÓVIL	PIN	PULSOMIN [µs]	PULSOMAX [µs]	SERVOMOTOR
Cresta	6	1000	2000	Futaba 3003
Ceja Izquierda	5	1000	2000	Turnigy TGY 50090
Ceja Derecha	A1	1000	1800	Turnigy TGY 50090
Cuello	A5	500	2500	HXT12K M 11kg
Rotacion cuerpo	A0	600	2400	HXT12K M 11kg

Tabla 7: Características Motores Robot

4.2.14 Batería Lipo.

Figura 45: Batería LiPo Ovonic

Para la alimentación de gran parte de nuestro robot, habiendo estudiado los requerimientos de la placa de Arduino, los 2 escudos, sensores, y sobre todo los servomotores, se ha decidido utilizar una batería Lipo de 5000mAh con un voltaje de 7.4V (recordemos los motores de las ruedas operan a unos 6V, y el resto de componentes necesitan 5V aproximadamente).

Características Batería LiPo	
Capacidad	5000mAh
Voltaje	7.4V
Amperaje máximo	5A
Tiempo de carga	Alrededor de 1h a 5A
Dimensiones	120 x 40 x 20 mm
Peso	80g

Tabla 8: Características Batería Li-Po

A continuación se hace una breve explicación de cada dato de nuestra batería: [12]

- La capacidad de la batería indica cuánta corriente puede suministrar la batería y se mide en mAh, es una manera de indicar cuánta carga (mA) puede suministrar la batería en 1 hora. De manera que si nuestra batería es de 5000mAh podría suministrar 5000mA durante 1h, 10000mA durante 30min, 2500mA durante 2h…
- La tasa de descarga o "C", indica la velocidad de descarga de la batería, es decir, la intensidad máxima que puede dar la batería de forma segura. Nuestra batería es de 50C, esto indica que tiene una descarga máxima continua de 50 x 5000 = 250000mA = 250A
- Las baterías Li-Po están formadas por celdas de 3.7V conectadas en serie, si nuestra batería es de 2s significa que está formada por dos celdas de 3.7V = 7.4V

Para cargar la batería es importante utilizar un cargador especial. Para más información a cerca de baterías Lipo dejo este enlace.

4.2.15 Batería portátil.

La alimentación de la Raspberry Pi 4B se realiza a través de un puerto USB-C, por ello, para alimentarla, usaremos una batería portátil que tengamos en casa. La inclusión de esta batería en el interior del robot aumentará de alguna manera el peso de este, pero no tenemos otra opción si queremos que Curro sea independiente.

Figura 46: Robot Curro con Batería Portátil

4.2.16 Interruptor Basculante.

Figura 47: Interruptor basculante

Utilizaremos este interruptor para el encendido y apagado del robot. Durante el desarrollo del proyecto, será de gran utilidad pues así podremos apagar cómodamente le suministro de batería y evitar descargas innecesarias.

Esto podremos hacerlo gracias a los 2 pines SPST de encendido y apagado que incorpora el robot.

4.3 Conexiones del Robot.

En este punto se explicará cómo se han realizado las conexiones de los distintos elementos que componen el robot. [6]

4.3.1 Placa Arduino UNO.

Excepto la cámara de la Raspberry Pi, todos los periféricos irán conectados de una manera u otra a la placa de Arduino UNO, que recordemos, es el cerebro de nuestro animatrón.

Para ello formaremos un castillo de 3 niveles:

- En la base encontraremos la placa Arduino Uno

- Encima de la placa Arduino UNO, colocaremos el motor shield, debemos saber que estos modelos son perfectamente compatibles y como se ha explicado con anterioridad, el motor shield permite añadir nuevas funcionalidades a la placa, además de preservar el resto de pines E/S, etc. A este módulo irá conectada la **alimentación de la batería**, y los **motores de las ruedas** (pines 10,11,12 y 13).
    - Motor Izquierdo:
        - Pin de dirección: 13
        - Pin de velocidad: 11
    - Motor derecho:
        - Pin de dirección: 12
        - Pin de velocidad: 10

- Por último, colocaremos el sensor shield sobre el motor shield, a este escudo irán conectados el resto de servomotores, el sensor ultrasónico, la tira de leds, el micrófono y el altavoz, de la siguiente manera:
    - Los 3 pines (**tierra**, **positivo** y **señal**) del **anillo de 24 LED's RGB** se conectarán a GND, VCC y señal del **pin digital 9** del sensor shield.
    - Los 4 pines del **sensor ultrasónico HC-SR04** se conectarán de la siguiente forma: la tierra, VCC y **trigger** del sensor a GND, VCC y señal del **pin digital 2**, y el pin **echo** del sensor a la señal del **pin digital 3**. Los pines 2 y 3 se encargarán de controlar este sensor.
    - El **micrófono KY-038** contiene 4 pines (A0 (receptor analógico), D0 (receptor digital, GND y "+") , para el proyecto se ha elegido la captación de sonido digital por lo que los pines **GND**, "**+**" y **D0** irán conectados a **tierra**, **positivo** y **señal** del **pin digital 8** del sensor shield.
    - El **altavoz** contiene 2 pines, **positivo** y **negativo**. Los conectaremos a positivo y tierra del **pin digital 7** del sensor shield.
    - En cuanto a las ruedas de los motores, se conectará el motor de la rueda izquierda (mirando de frente al robot) a los pines ("+" y "-") de la izquierda de la imagen. El motor de la rueda derecha se conectará a los otros dos pines ("+" y "-") de la derecha de la imagen).

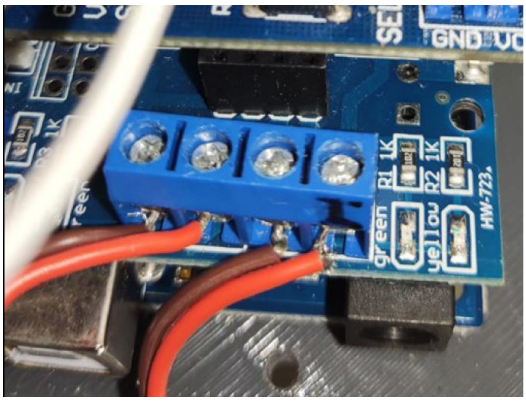

Figura 48: Conexión motores ruedas a Motor Shield

- El resto de motores se conectarán de la siguiente manera:

Elemento	Motor	Pin (Sensor Shield)
Rotación del cuerpo	HXT12K	Analog: A0
Cuello	HXT12K	Analog: A5
Ceja derecha	Turnigy TGY50090	Analog: A1
Ceja izquierda	Turnigy TGY50090	Digital: 5
Cresta	Futaba 3003	Digital: 6

Tabla 9: Conexiones Motores

La alimentación (7.4V) irá directamente conectada al motor shield, puesto que los motores necesitan más de 5V, será el motor shield el que a través de su salida de 5V alimente el sensor shield, y el sensor shield (a 5V) alimentará a la placa de Arduino UNO, asegurando así que estas dos ultimas placas reciben 5V y no más.

Tal y como se explicó anteriormente, a la batería irá empalmado el interruptor basculante, de esta manera tendremos fácil conexión y desconexión de la alimentación.

La **placa de Arduino** irá conectada a la **Raspberry Pi** a través del cable **USB-A**, a través del cuál se realizará la comunicación serial entre ambas placas.

4.3.2 Placa Raspberry Pi.

- La **placa Raspberry Pi 4B** estará alimentada a través de un cable **USB-C** por la **batería portátil**, que irá también introducida dentro del robot.
- La **cámara** se conectará al **puerto CSI** de la Raspberry, situado entre los puertos micro-HDMI y el conector Jack para audio y vídeo.

Figura 49: Conexion Cámara Pi a Raspberry

A continuación, muestro un esquemático del conexionado del robot:

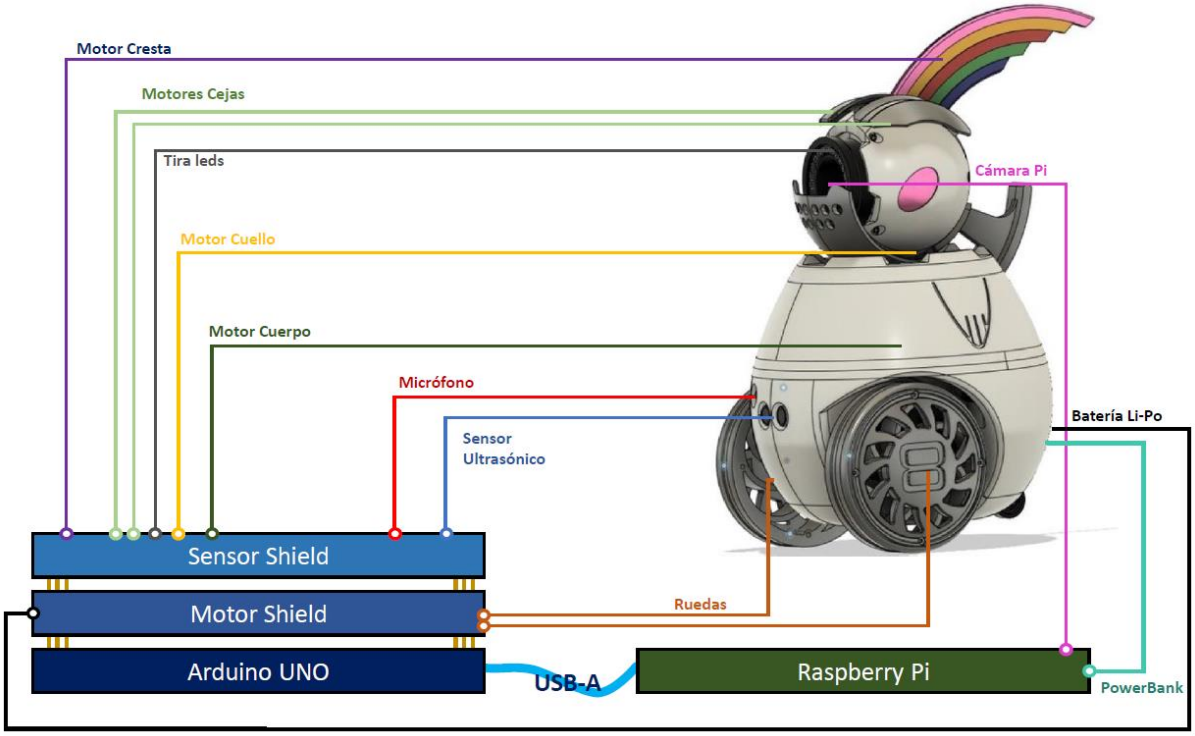

Figura 50: Esquemático Conexionado Robot

5 ENTRENAMIENTO

"La suerte es lo que sucede cuando la preparación se encuentra con la oportunidad"

- Séneca -

En este capítulo se desarrollará la metodología llevada a cabo para el entrenamiento de los comandos mentales y faciales para el control de acciones.

5.1 Tipos de control BCI

Tal y como se explicó en apartados anteriores, durante este proyecto se han explorado 2 vías de control por BCI, el control mediante **comandos mentales** y el control mediante **expresiones faciales**.

El primero de estos, es el más atractivo, pensar en poder controlar a nuestro robot solo con el pensamiento es algo que a cualquiera le produce asombro y curiosidad, más aún cuando es un tipo de control que hoy todavía no está muy avanzado. A continuación se mostrarán las distintas técnicas de entrenamiento que se han llevado a cabo para la asociación de órdenes a comandos mentales, iremos viendo cómo se presentan problemas de limitación por número de acciones, imprecisión de las órdenes, dificultad para segmentar pensamientos…

El entrenamiento mediante expresiones faciales, pese a que puede ser a priori menos impresionante, iremos viendo qué ventajas posee, cómo puede aportar versatilidad, robustez, precisión y comodidad al control de nuestro robot. A lo largo del capítulo se demostrará por qué se ha optado por esta vía para el control total del robot.

5.2 Control Mental | Tipos de Entrenamiento

5.2.1 Límite de acciones

El software Emotiv permite el entrenamiento de hasta 4 acciones diferentes asociadas a comandos mentales, es decir que podríamos pensar hasta 4 cosas diferentes para hacer que nuestro robot se moviera hacia delante, hacia atrás, a izquierda, o a derecha. Esto puede ser una limitación sí nuestro objetivo fuera controlar más variables a parte del movimiento traslacional del robot.

5.2.2 Precisión de las órdenes.

El entrenamiento de los comandos mentales para asociarlos a ciertas órdenes puede llegar a ser muy tedioso, quizás nos creemos que podemos controlar nuestros pensamientos pero la realidad es que es muy complicado concéntranos únicamente en un pensamiento específico y aislar nuestra mente de otras perturbaciones.

Además, pese a que distintos pensamientos pueden generar distintas ondas cerebrales, concepto que desarrollaremos de manera extendida en el capítulo de entrenamiento (pensar en colores, en operaciones matemáticas, en imágenes…) a la hora de la verdad, las ordenes que intentemos mandar pueden confundirse, y cuando queramos avanzar hacia delante el robot interprete que queremos ir hacia atrás, en el caso de que esto ocurriera y quisiéramos solucionarlo, tendríamos que ser capaces de aislar nuestra mente del "problema" que estamos teniendo para poder enviar la orden deseada. De alguna manera podemos estar limitando nuestro cerebro al envío de algunas ordenes muy concretas, anulando la posibilidad de pensar en posibles soluciones a perturbaciones, la posibilidad de hablar con alguien al mismo tiempo que controlamos nuestro robot.

Obviamente la precisión y la robustez de nuestro entrenamiento dependerá de lo bien que hayamos entrenado las ordenes, la capacidad de concentración del usuario, así como la habilidad para seleccionar los pensamientos adecuados para el envío de órdenes

5.2.3 Proceso de entrenamiento

Para comenzar con una sesión de entrenamiento, debemos seguir los siguientes pasos:

1. Debemos asegurarnos de que nuestro dispositivo Emotiv está cargado
2. Crear un nuevo perfil de entrenamiento, podemos crear tantos perfiles de entrenamientos como queramos.

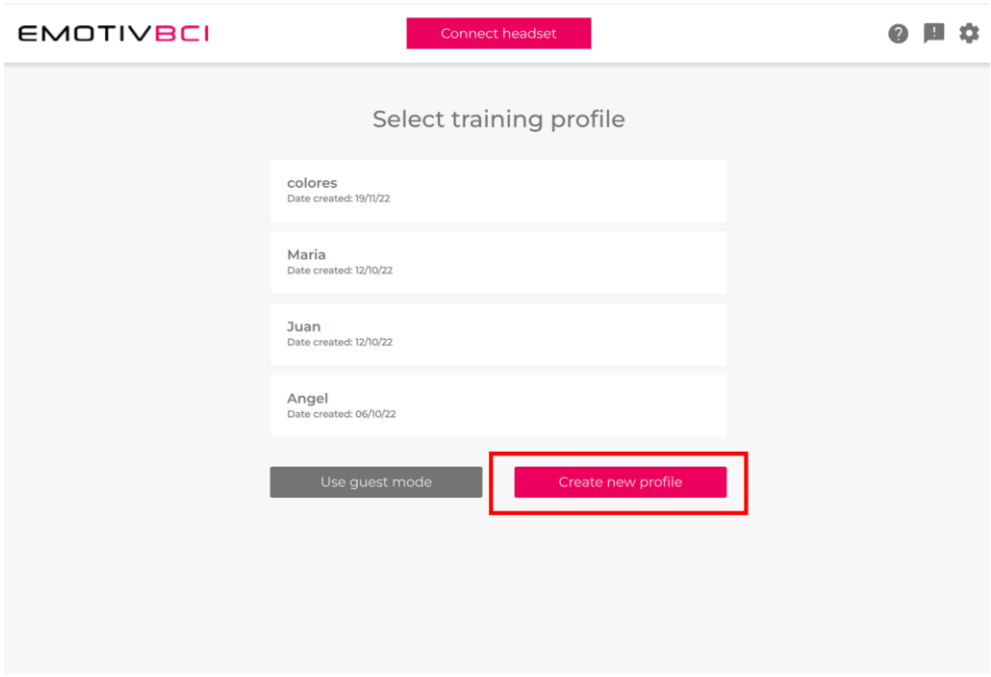

Figura 51: Creación nuevo perfil | Entrenamiento Mental

3. Conectar la diadema y comprobar la buena conectividad de los sensores, para ello deberemos

conectar la diadema con la aplicación BCI y seguir los pasos que nos vaya indicando el sistema. Deberemos tener a mano nuestro lubricador salino para humedecer los sensores. Recordemos que este proceso es muy importante, dado que permitirá que desarrollemos el entrenamiento con mayor precisión, y que las ondas que se capten sean lo más fiables posibles.

4. Cuando accedamos a la ventana de entrenamiento veremos esta pantalla:

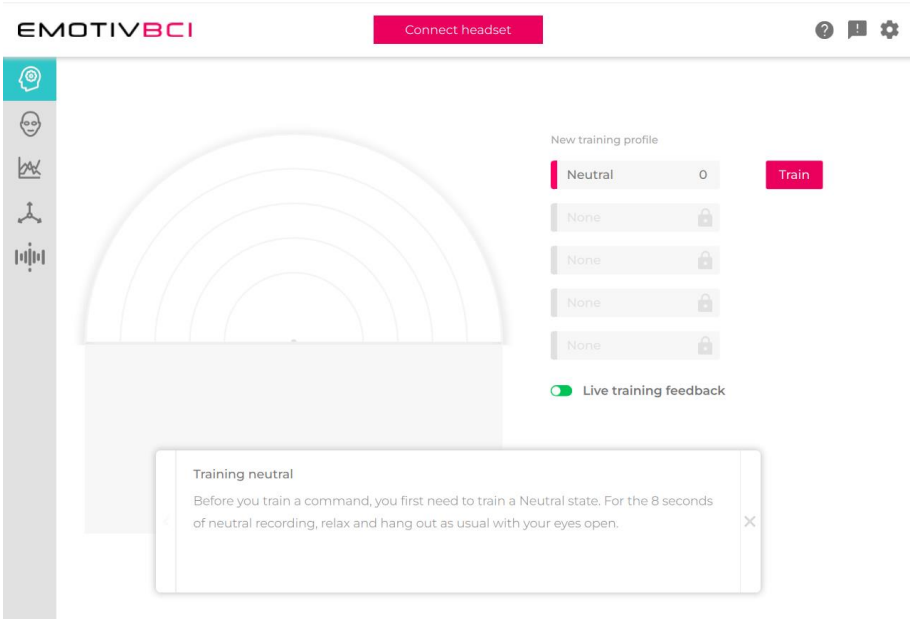

Figura 52: Inicio Entrenamiento | Entrenamiento Mental

5. La aplicación nos muestra un mensaje en el que nos dice que antes de empezar con el entrenamiento necesita registrar nuestro estado cerebral del día, de esa manera, tomará como referencia nuestro estado neutral. Así podremos permitirnos entrenar comandos distintos días sin que nuestro estado anímico, o de estrés, pueda influir demasiado en el entrenamiento.
6. El primer comando que debemos entrenar es el 'Neutral', que representa nuestra actividad cerebral básica, lo recomendable es entrenarlo al menos 10 veces antes de empezar con otro comando.
7. Una vez hayamos hecho esto, podremos ir entrenando otros comandos. El entrenamiento de cualquiera de los comandos tiene una duración de 8 segundos. Tras cada ciclo de 8 segundos, el sistema nos pregunta si queremos aceptar el entrenamiento como válido o no. Además, tendremos una referencia de cómo de bueno ha sido el entrenamiento en base a los datos guardados en iteraciones de entrenamientos anteriores.

Como se mencionó con anterioridad, a la hora de entrenar una orden aparecerá un cubo virtual que simulará la acción que se suponga que estemos entrenando.

Igualmente se recomienda entrenar las acciones de 1 en 1, de tal manera que vayamos creando acciones robustas y nuestro cerebro tenga la capacidad de diferenciar a qué acción nos estamos refiriendo.

Existen 2 modos de entrenar, con Live training feedback activado o desactivado:

- Live training feedback desactivado: Cuando estemos entrenando la acción 'push', el bloque virtual que aparece en pantalla se moverá hacia el fondo de la pantalla, simulando una buena respuesta mente – cubo
- Live training feedback activado: Este modo debemos activarlo cuando el entrenamiento esté más avanzado, pues el cubo sólo se moverá si los estímulos cerebrales que reciben se

asemejan a los anteriormente almacenados en la base de datos. De esta manera podremos saber cómo de bueno está siendo dicho ciclo de entrenamiento. En contraposición, el cubo no nos ayudará a asociar el movimiento con el pensamiento puesto que estos dos irán relacionados.

8. Cuando creamos oportuno, podremos ir comprobando como va nuestro entrenamiento, accediendo al 'Live Mode'. En este modo, no estaremos entrenando, sino comprobando si el entrenamiento es bueno.

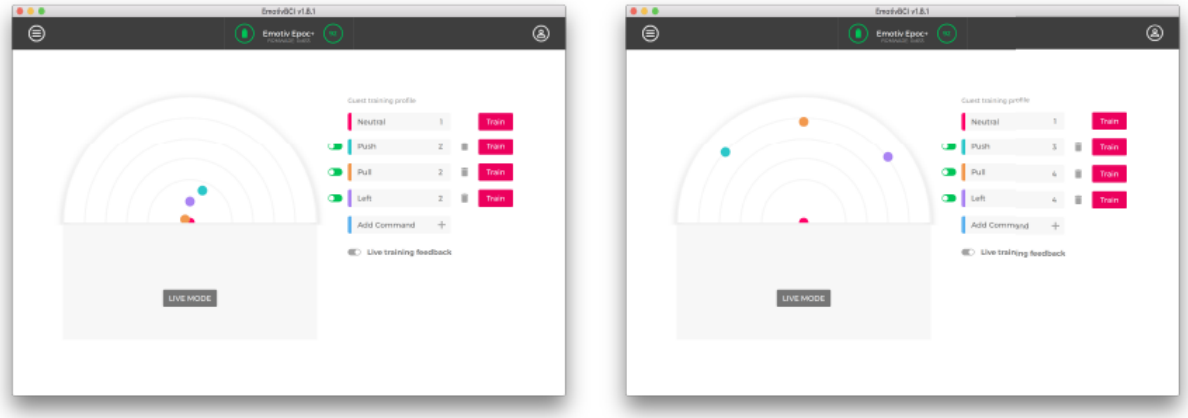

Figura 53: Mal entrenamiento Vs Buen entrenamiento

Lo separados que estén los distintos puntos de colores nos da información de lo bien que hemos entrenado los comandos. Un perfil que tenga los colores muy juntos tendrá una gran dificultad para diferenciar qué comando es el que está intentando mandar el usuario.

5.2.4 Entrenamiento basado en colores

Una de las técnicas utilizadas para la generación de acciones es el entrenamiento basado en colores. Hay algunos estudios que afirman que ciertos colores están asociados a la activación de distintas ondas cerebrales.

Por ejemplo, cuando pensamos en el color rojo, se ha observado un aumento en las ondas cerebrales beta en la región occipital del cerebro, que se relaciona con la percepción visual. También se ha demostrado que pensar en el color azul puede aumentar la actividad en las ondas cerebrales Alpha, que están asociadas con la relajación y la meditación.

El color verde se suele asociar con la naturaleza, quizás pensar en escenas que contengan mucho verde pueda ayudarnos a generar un estado mental determinado

Para el entrenamiento basado en colores, se han buscado muchas imágenes que contengan dicho color y se a ha recurrido a la visualización repetida de dichas imágenes, tratando también de llevar a nuestro cerebro al estado que nos conduzca dicho color.

He asociado el color rojo a la accion 'push', para entrenar he buscado en internet imágenes que contengan el color rojo y mientras las miraba intentaba pensar en dicho color.

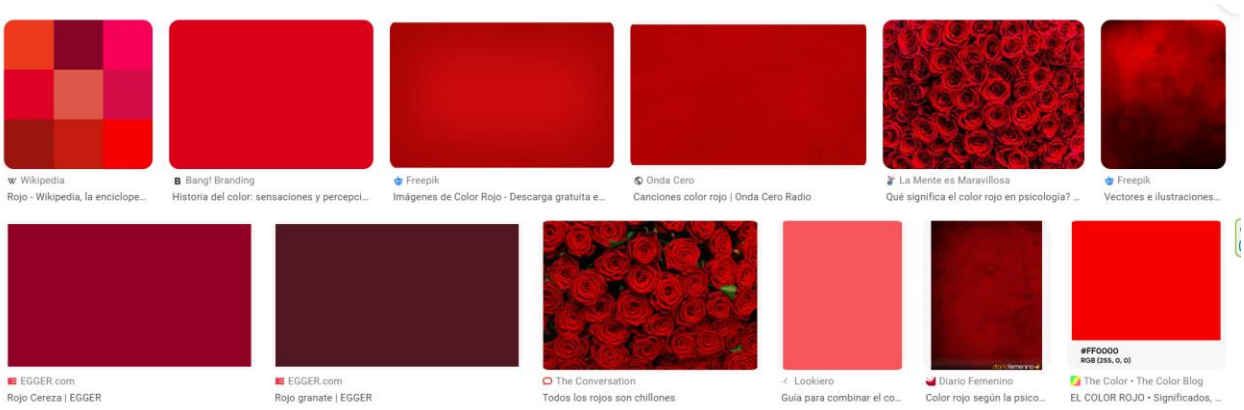

Figura 54: Entrenamiento basado en colores | Rojo

También he asociado el color azul a la acción ‘pull’, he seguido la misma estrategia.

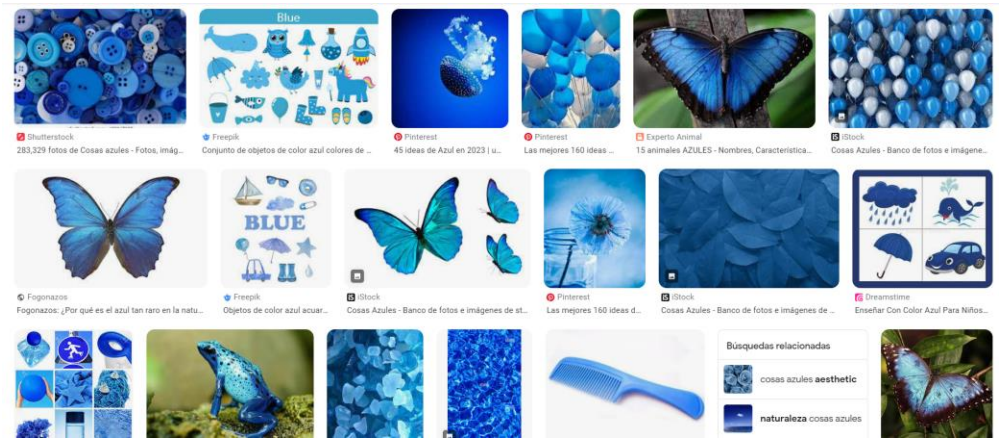

Figura 55: Entrenamiento basado en colores | Azul

Al principio el resultado es bastante bueno, parece ser que la actividad cerebral que se genera al pensar en estos dos colores es diferente, se puede comprobar en este gráfico:

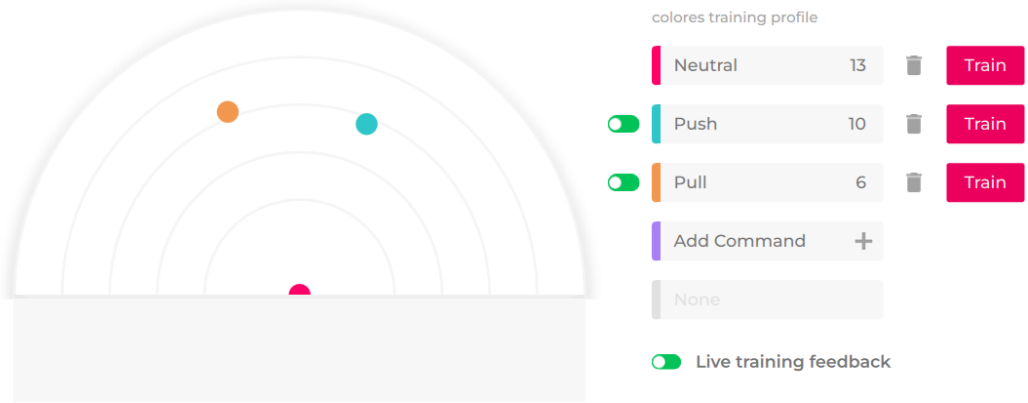

Figura 56: Entrenamiento basado en colores | Gráfico primeras respuestas

Debemos recordar que cuanto más separados estén los puntos entre sí, más “distancia” hay entre las actividades cerebrales que dan cada acción.

Podríamos creer que el entrenamiento está siendo todo un éxito, pero la realidad es que el número de veces que hemos entrenado las acciones es muy baja. El programa tiene una IA que te dice cómo de bien has entrenado el comando en el intento nº i, basándose en los resultados de los intentos i-1, i-2… Es elección nuestra aceptar o no dicho entrenamiento para añadirlo a la base de datos de la acción.

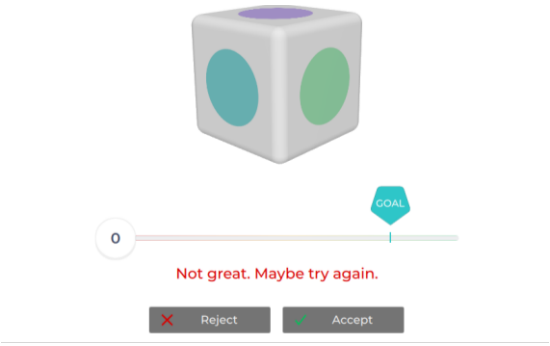

Figura 57: Entrenamiento basado en colores | Intento nº7 entrenamiento 'pull'

Este es el mensaje que aparece por pantalla al realizar el intento nº7 del entrenamiento del comando ‘pull’, pensando en el color azul. La secuencia que he seguido ha sido la misma sin embargo la IA cree que estoy pensando en algo bastante distinto. Para ser fiel al entrenamiento, voy a aceptar el intento a ver si el próximo consigo que se asemeje más. Recordemos que nuestro objetivo es poder pensar en aquello que decidamos siempre y cuando queramos, intentando evitar estar sujetos a unas condiciones muy concretas. Es por ello que cuántos más intentos grabemos, más fiable será el entrenamiento.

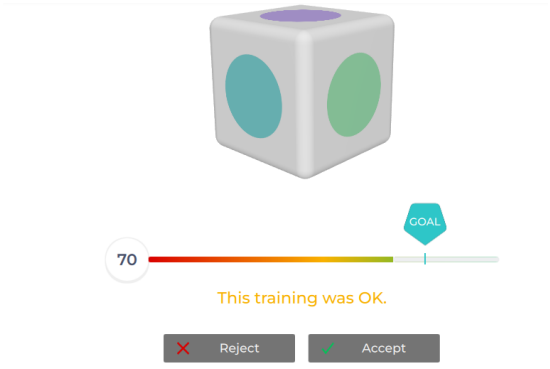

Figura 58:Entrenamiento basado en colores | Intento nº8 entrenamiento 'pull'

Tras algunos intentos más, se da por finalizado el entrenamiento basado en colores, las conclusiones que se pueden sacar son:

- Aparentemente pensar en distintos colores estimula distintas áreas del cerebro y eso genera distintas ondas cerebrales.
- El entrenamiento basado en colores está sujeto a estar mirando continuamente en un color para poder pensar en él. Hacerlo sin mirar es realmente complicado y la prueba de ello es que

cuando iniciamos el 'live mode' para comprobar la efectividad de nuestro entrenamiento, no se ejecutan las acciones asociadas a los colores en los que pensamos.

Resultado final de la gráfica:

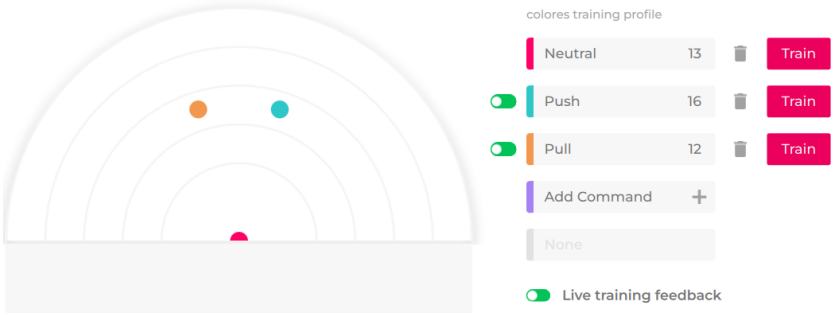

Figura 59: Entrenamiento basado en colores | Resultado final entrenamiento

5.2.5 Entrenamiento basado en estados emocionales

Otro de los métodos que hemos utilizado para el entrenamiento de los comandos mentales es jugar con los estados emocionales, es decir, intentar que el usuario asocie una acción determinada a un estado emocional como puede ser tristeza, alegría, agitación…

Para la realización de esta técnica es muy importante la concentración que tengamos, dado que necesitamos focalizarnos en aquello que estemos pensando que nos cause alegría o tristeza.

En este punto comprobaremos como es la respuesta del entrenamiento, por si fuera de utilidad poder combinarla luego con otras técnicas. Dado que ya sin haber comenzado podemos deducir que puede llegar a ser complicado conseguir llevar al cerebro a 4 estados distintos, de manera deliberada y precisa, aun así, veremos cómo responde el software.

5.2.6 Entrenamiento basado en acciones concretas

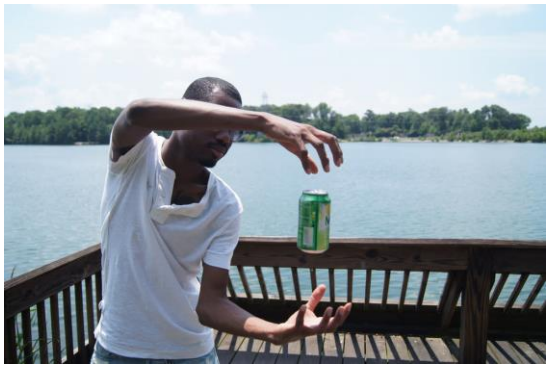

Figura 60: "Telequenisesis"

Uno de los métodos más intuitivos para mover un objeto hacia delante puede ser precisamente, pensar en mover el objeto hacia delante… Siguiendo esta lógica, durante este punto intentaremos relacionar las acciones que deseemos con pensamientos que involucren algun objeto realizando el movimiento que queremos.

Además, la interfaz de entrenamiento que proporciona EmotivBCI es muy conveniente para esta

técnica, dado que durante el entrenamiento de una acción 'x', estaremos observando en pantalla una imagen de un cubo virtual que se moverá en función de la orden que estemos entrenando.

Si estamos entrenando la orden 'push', cuando estemos en los 8 segundos de entrenamiento de dicha orden, donde tenemos que pensar (en este caso) en un cubo que se mueve hacia delante, veremos como el cubo virtual se introduce cada vez más en la pantalla, como si alguien lo estuviera empujando. Esto nos ayuda a asociar dicho pensamiento con la acción.

5.3 Control Facial | Tipos de Entrenamiento

A medida que se iba avanzando en el proyecto y se investigaba con distintos métodos de entrenamiento para el control mental, nos íbamos dando cuenta de los inconvenientes que encontrábamos en este tipo de control, que son principalmente dos:

- La dificultad de simultanear al menos 4 acciones diferentes asociadas a distintos pensamientos y obtener una precisión lo suficientemente buena como para poder dirigir a nuestro robot de manera deliberada en la mayoría de los casos.
- En segundo lugar, al asociar el control motor del robot con nuestra mente, estamos dejando totalmente inutilizado nuestro cerebro para poder pensar otras cosas, explicar el funcionamiento del robot en tiempo real, mantener una conversación mientras manejamos al robot… es decir, el control mental del robot requiere hoy en día una concentración extrema, y a diferencia de la capacidad que tenemos para mover nuestro cuerpo al mismo tiempo que hablamos, pensamos, etc., las necesidades de las señales de Emotiv implican una dedicación total al envío de la acción a la cual nos estemos refiriendo.

Principalmente por estos dos motivos se decidió explorar también el control por medio de expresiones faciales, este método es más sencillo, permite simultaneidad con otras funciones (hablar, pensar, caminar, reaccionar ante perturbaciones…), y permite asociar al menos 4 acciones distintas con una precisión y fiabilidad bastante superior a casos anteriores.

Tal y como se explicó en capítulos anteriores, Emotiv, además de captar señales EEG también procesa las señales EMG y EOG, estas señales están asociadas a los estímulos musculares enviados por los músculos de la cara, y a los estímulos enviados por los globos oculares, respectivamente.

La combinación de estos dos tipos de señales nos permitirá entrenar acciones de movimiento del robot con movimiento de cejas, sonrisa, parpadeo, guiño, y otros movimientos faciales.

La aplicación Emotiv BCI posee un apartado para el entrenamiento de estas acciones, aunque cabe destacar que este tipo de control requiere un entrenamiento menor de algunas partes como por ejemplo el movimiento de los ojos (guiño, parpadeo, movimiento de ojos) dado que estas acciones están muy bien diferenciadas y poseen comportamientos muy específicos y comunes entre todos los humanos.

5.3.1 Gráficas Expresiones Faciales

A continuación se mostrarán distintas capturas de las gráficas de las ondas captadas por la diadema, en ellas, se pueden identificar diferentes movimientos faciales:

- Parpadeo ojos (leve)

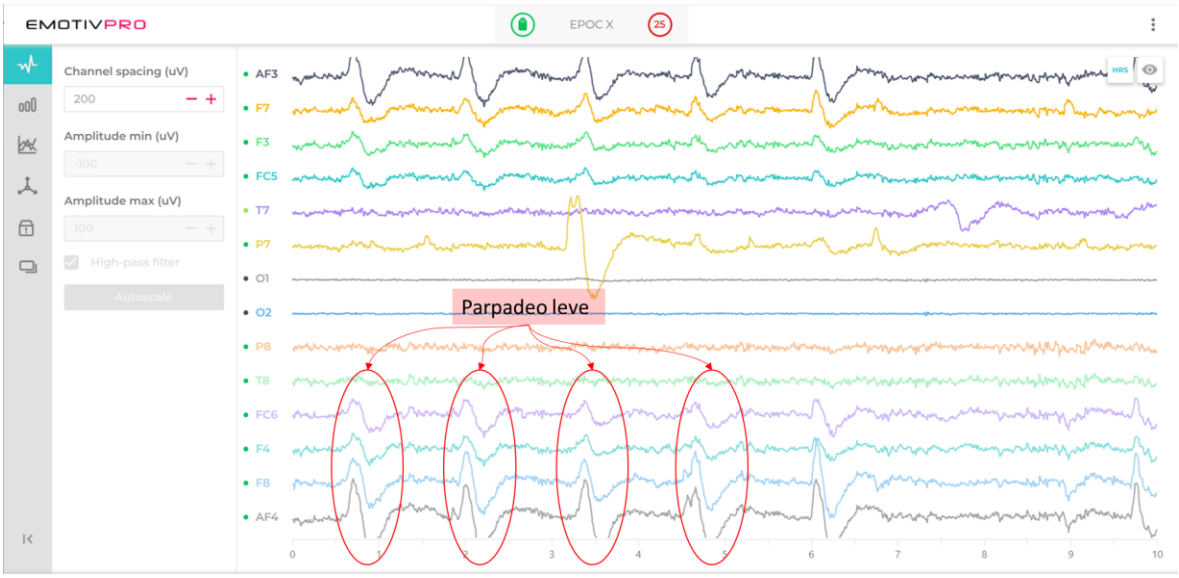

Figura 61: Gráfica Parpadeo Ojos (leve)

- Parpadeo ojos (fuerte)

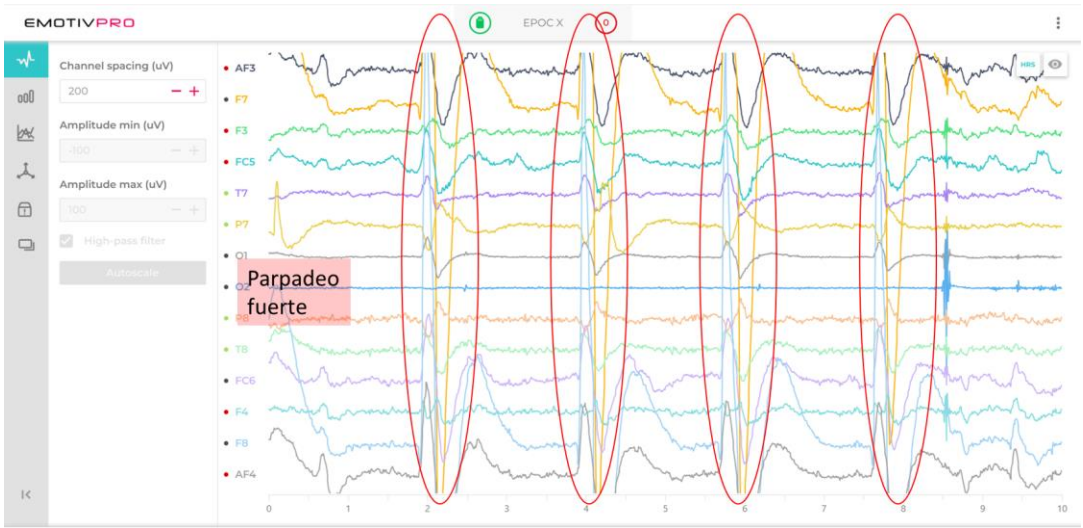

Figura 62: Gráfica Parpadeo ojos (fuerte)

- Guiño ojos

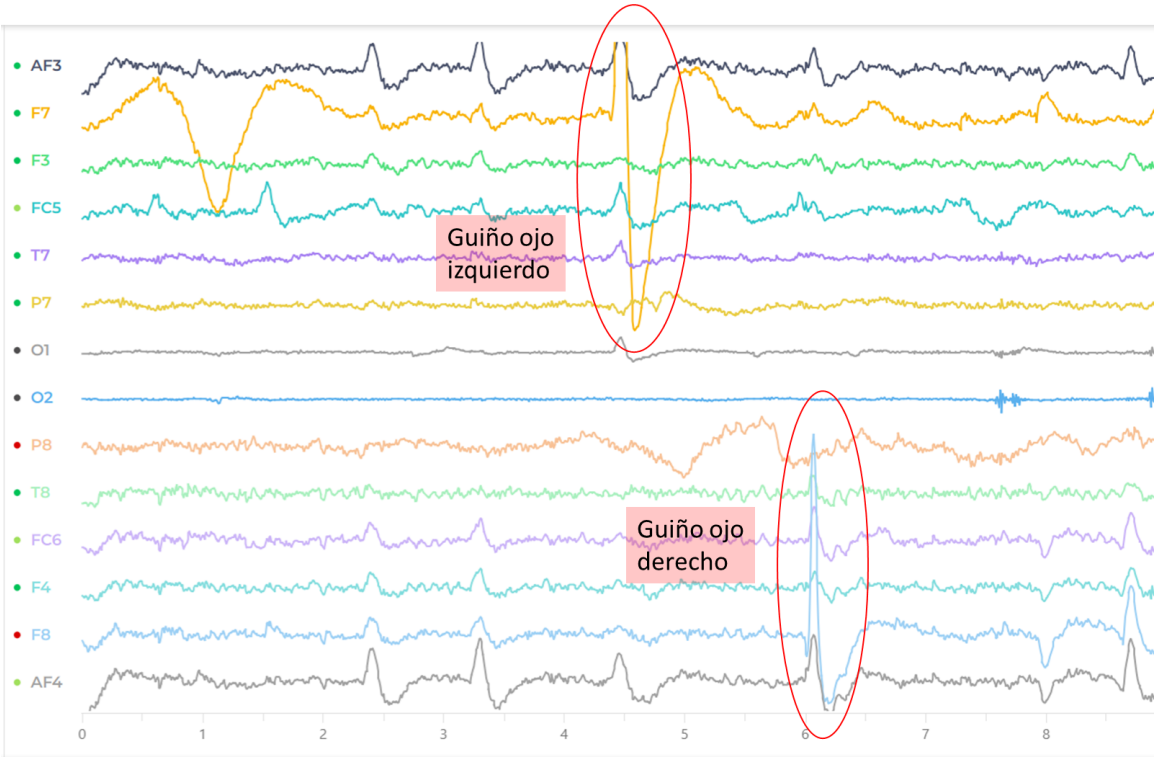

Figura 63: Gráfica guiño ojos

5.3.2 Órdenes - Acciones

El robot está programado para realizar las siguientes acciones por orden de comandos mentales / faciales:

Avanzar, retroceder, giro izquierda, giro derecha, sacar fotografía y apagarse

En el caso en el que nos encontramos, necesitaremos asociar una orden de expresión facial a cada una de las acciones.

- Avanzar → Elevación de cejas
- Retroceder → Enseñar los dientes
- Giro Izquierda → Guiño ojo izquierdo
- Giro Derecha → Guiño ojo derecho
- Sacar Fotografía → Sonreír
- Apagarse → Parpadeo intenso

Al igual que en el entrenamiento de comandos mentales aparece un cubo virtual que simula los movimientos que estamos entrenando, en el entrenamiento de expresiones faciales aparece una cara que adopta la expresión facial que se esté entrenando.

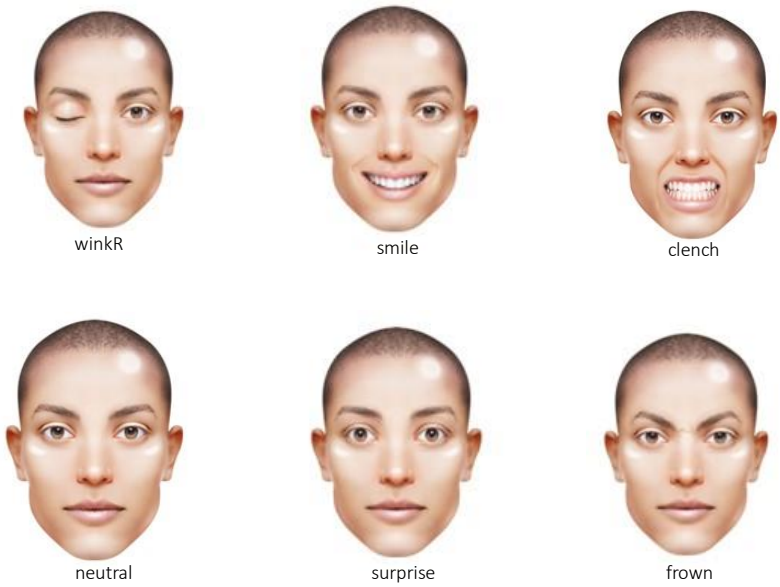

Figura 64: Expresiones faciales EmotivBCI

O bien si iniciamos el Live Mode (para comprobar cómo de fiable es nuestro entrenamiento) la cara irá adoptando la expresión que la diadema esté interpretando.

5.4 Problemas encontrados

Conectividad sensores

Sin duda este ha sido uno de los mayores inconvenientes que se ha encontrado durante la realización del proyecto. No es nada fácil conseguir una buena conectividad de los sensores, pese a intentar lubricar los sensores de manera contundente siguiendo las instrucciones del fabricante, casi siempre había algún sensor que no se conectaba bien. Este problema puede desencadenar que el entrenamiento que realicemos esté perturbado por señales de ruido que los sensores mal conectados emitan durante la transmisión.

Restricción de órdenes

El hecho de no poder compatibilizar más ordenes, o en el caso del entrenamiento mental, no lograr una buena precisión en dos o tres órdenes ha sido frustrante, dado que se han invertido muchas horas en entrenamiento y es realmente complicado entrenar los comandos, al menos con los conocimientos que he podido obtener hasta el momento. Aun así no dudo del potencial de esta tecnología y a la vista está que con esfuerzo se han logrado muy buenos resultados.

Sensibilidad de las acciones

Al realizar el entrenamiento de los comandos faciales, se ha podido observar cómo algunas acciones son más sensibles que otras, esto ocurría para la expresión “fruncir el ceño”, de alguna manera predominaba sobre las demás y en numerosas ocasiones la acción “saltaba” sin que el usuario

hubiera ejecutado la acción. Este es un problema recurrente con el que se tuvo que lidiar para evitar que el robot realizara acciones indeseadas, esto además ocurre debido a que no existe un botón de seguridad o algo por el estilo que permita al usuario determinar con certeza cuando quiere enviar la orden, de otro modo, si el usuario que controla el robot de repente estornudara, el robot podría interpretar cualquiera de las acciones y ejecutar algún movimiento indeseado.

Para solventar este problema, se pueden hacer varias cosas, por un lado, en la propia aplicación de Emotiv, se puede ajustar la sensibilidad de cada comando, de manera que si queremos que "fruncir el ceño" sea menos sensible podemos bajarle la sensibilidad. Además, en el programa de tratamiento y procesado de órdenes "PC_emotiv_py.py" se ha creado un sistema que también ajusta la sensibilidad de las órdenes. De manera resumida este sistema acepta una orden como válida sólo cuando esta se haya recibido en un determinado porcentaje durante el tiempo de muestreo.

6 PROGRAMACIÓN

"Lo mejor de los booleanos es que si te equivocas estás a un sólo bit de la solución correcta"

-Anónimo-

En este capítulo se explicará cómo se han ido desarrollando los distintos ficheros de código que se han utilizado para la realización de este proyecto, donde se han empleado los lenguajes de programación Python (para la Diadema de Emotiv y la conexión Emotiv - Robot) y Arduino (para la programación del robot)

6.1 Diagrama flujo conexiones

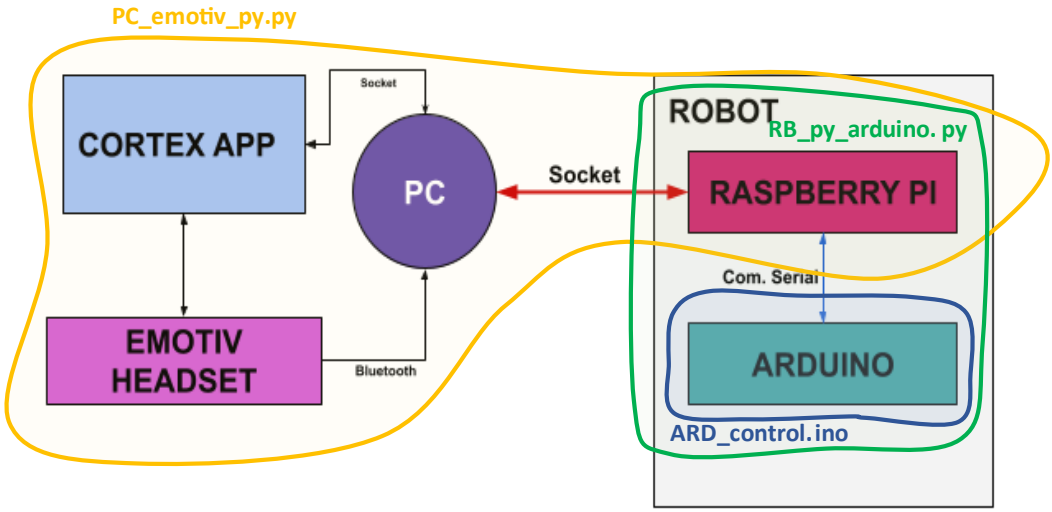

Figura 65: Diagrama Flujo Conexiones

En el **PC** se correrá el programa **PC_emotiv_py.py**

En la **Raspberry** se correrá el programa **RB_py_arduino.py**

En **Arduino** se correrá el programa **ARD_control.ino**

6.2 Programación PC_emotiv_py.py

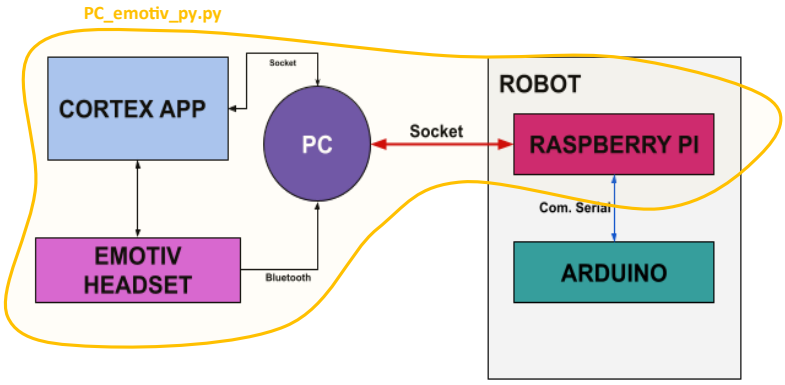

Figura 66: Diagrama Flujo Conexiones | PC_emotiv_py.py

Este fichero se encargará de recibir los datos que enviamos desde la diadema y los transforma en datos que el robot sea capaz de procesar para realizar las acciones y enviárselos.

Como se mencionó previamente, para la creación de este fichero se ha utilizado como esqueleto el fichero proporcionado por Emotiv llamado **live_advance.py**, en él se crea una clase llamada `LiveAdvance()`:

Su función inicial es imprimir por pantalla las acciones que se envían desde la diadema, de tal modo que podemos ver en tiempo real cuál es la acción que la diadema interpreta que queremos enviar.

```
class LiveAdvance()
```

Se ha transformado este código que es proporcionado por la empresa Emotiv para desarrollar las aplicaciones BCI para que se ajustes a nuestras necesidades. En principio el fichero lo que hacía era imprimir por pantalla las acciones que, tras el entrenamiento, emitimos a través de nuestros estímulos cerebrales.

Tras la evolución de este, el fichero se ocupa de:

- Establecer conexión vía socket con el robot, mediante red Wi-Fi comunicaremos a nuestro PC con el Robot.
- Llevar a cabo un control síncrono de esta comunicación para enviar las acciones sólo cuando el robot esté listo para recibirlas.
- Elaborar una función que se encargue de recoger durante x segundos las acciones que nuestro cerebro ordena al robot, para así hacer una selección de cuál ha sido la acción más “votada”, de manera que sea esta la que se envié como acción deseada al robot.

En definitiva, este fichero actúa como un bloque que recibe ordenes enviadas por nuestro cerebro a muy alta frecuencia, y tiene como salida una orden concreta que ha sido debidamente seleccionada, que le será enviada al robot para que realice la acción que se le pida.

A continuación se irá desglosando el código y comentando el funcionamiento de sus partes más importantes.

6.2.1 Atributos de la clase:

Esta clase tendrá el atributo 'c': **Cortex** (se importará aquí la clase superior del fichero cortex.py)

```
from cortex import Cortex
```

En el constructor de la clase **LiveAdvance()**: se iniciarán todas las asociaciones de las funciones que posee la clase Cortex para que esta nueva clase pueda usarlas. Además, se incluirán todas las líneas de código que queramos que se ejecuten al ejecutarse la clase.

```
def __init__(self, app_client_id, app_client_secret, **kwargs):

        self.c = Cortex(app_client_id, app_client_secret, debug_mode=True, **kwargs)
        self.c.bind(create_session_done=self.on_create_session_done)
        self.c.bind(query_profile_done=self.on_query_profile_done)
        self.c.bind(load_unload_profile_done=self.on_load_unload_profile_done)
        self.c.bind(save_profile_done=self.on_save_profile_done)
        self.c.bind(new_com_data=self.on_new_com_data)
        self.c.bind(new_fe_data=self.on_new_fe_data)
        self.c.bind(new_mot_data=self.on_new_mot_data)
        self.c.bind(get_mc_active_action_done=self.on_get_mc_active_action_done)
        self.c.bind(mc_action_sensitivity_done=self.on_mc_action_sensitivity_done)
        self.c.bind(inform_error=self.on_inform_error)

      # Creamos dos objetos Event y una cola
        self.evento1 = threading.Event()
        self.evento2 = threading.Event()

        self.cola = queue.Queue()
        self.t = threading.Thread(target = self.server, args = (self.cola,))
        self.t.start()

        self.cont_z = 0
        self.cont_w = 0
        self.cont_d = 0
        self.flag = 0

        self.tiempo_ini = time.time()
```

Código PC: 1. PC_emotiv_py.py | Función __init__()

6.2.2 Métodos de la clase:

La clase `LiveAdvance()`: posee distintos métodos que hacen posible el funcionamiento del código, entre ellos encontramos:

Métodos pred. de la clase	Descripción
start():	Para empezar un 'live mental process' y abrir el WebSocket
load_profile(profile_name):	Para cargar un perfil
unload_profile(profile_name):	Para descargar un perfil
get_active_action(profile_name):	Recoge las acciones de la detección de comandos mentales
get_sensitivity(profile_name):	Muestra las sensibilidades asignadas a cada acción mental
set_sensitivity(profile_name):	Para ajustar las sensibilidades de cada acción mental

Tabla 10: Métodos predeterminados de la clase Live_Advance()

6.2.3 Funciones 'CallBack':

Una función 'callback' es aquella que es pasada como argumento a otra función para que sea "llamada de nuevo" en un momento posterior.

Entre estas funciones encontramos algunas que simplemente son necesarias para acceder a la sesión de transmisión: solicitar acceso al perfil, guardar el perfil…

Las funciones más importantes se desarrollan a continuación:

6.2.3.1 Función on_save_profile_done()

```
def on_save_profile_done (self, *args, **kwargs):
        print('Save profile ' + self.profile_name + " successfully")
# En la variable stream elegimos a qué datos queremos subscribirnos, fac:
facial, com: mental, mot: motriz
        stream = ['fac']
        self.c.sub_request(stream)
```

Código PC: 2. PC_emotiv_py.py | Función on_save_profile_done()

En esta función seleccionaremos a qué tipo de datos queremos suscribirnos. En el caso de que queramos controlar al robot con los **comandos mentales** deberemos escribir:

```
stream = ['com']
```

Si queremos recibir datos de **expresiones faciales** debemos escribir:

```
stream = ['fac']
```

6.2.3.2 Funciones on_new_fe_data() y on_new_com_data()

Estas funciones tienen un gran papel en el código, dentro de estas es donde de manera predeterminada se recogen los datos que la diadema envía, los datos faciales en on_new_ fe_data() y los mentales en on_new_com_data().

Esto se hace mediante esta línea de código: `data = kwargs.get('data')` , tanto en una función como en otra.

Como sabemos que tal y como funciona el código y el envío de órdenes, a una frecuencia determinada se accede a estas funciones, lo que hemos hecho es crear el resto del código dentro de estas, asegurándonos así que las nuevas funcionalidades que introduzcamos se ejecutan de manera síncrona con la actualización de nuevos datos.

A partir de ahora centraremos la explicación en la función **on_new_fe_data**() que entra en juego cuando estamos trabajando con estímulos faciales.

La estructura que posee el proceso de actualización de datos, tratamiento, y posterior envío es la siguiente:
Flag = 0 | Esperando disponibilidad del Robot
Flag = 1 | Recogida de Datos
Flag = 2 | Envío de orden.

Las funciones que se utilizaran para ello son:

· **server():** función que asociaremos a un hilo que se encargará de la comunicación vía socket con el robot. Cuando iniciemos el programa se creará el hilo, se establecerá conexión con el cliente, y cuando esta se haya verificado, esperará a la recepción de la orden.
Posteriormente cuando el main le envíe la orden que haya sido seleccionada, el hilo enviará dicha orden a través del socket al robot, y el proceso se reiniciará (la conexión socket no, la conexión se queda activa)

· **choose_action_fac():** esta función recibe como argumento una lista con todas las ordenes que se han registrado durante el tiempo de recepción que hayamos establecido, la función devolverá la orden elegida (la más votada).

Otras herramientas:

Como se ha mencionado anteriormente, se creará un hilo que se encargará de establecer la comunicación con el robot vía sockets, además este hilo deberá comunicarse de manera local con el resto del programa. Esto lo hemos conseguido haciendo uso del método **event** de la librería *threading*.

Este método lo utilizaremos para sincronizar el hilo y el main, el main se quedará esperando al hilo hasta que este establezca la conexión con el robot, y posteriormente el hilo esperará al main a que este le envíe la orden seleccionada. Se han creado dos variables event: evento1 y evento2.

Para la gestión de los distintos estados del código, hemos creado la variable **flag**, que según el valor que tome (0, 1 ó 2) hará que el programa se encuentre en 3 estados distintos

flag = 0 | Esperando Disponibilidad Robot

Cuando se llegue a este estado, el main deberá esperar a que la variable self.evento1 se active a 1, esto significará que el hilo ha establecido correctamente la conexión con el robot.

Una vez hecho esto, se pone la variable flag = 1 (para pasar al siguiente estado), se inicializa el tiempo de muestreo y se crea la lista vacía que contendrá los datos recogidos.

```
if self.flag == 0:
            print("\n Main: Programa principal esperando al hilo")
# el main espera esta señal que indica que el robot está preparado para
recibir una orden
            self.evento1.wait()
            self.evento1.clear()
            self.flag = 1

            self.tiempo_ini = time.time()
            self.lista_com = []
```

Código PC: 3. PC_emotiv_py.py | Estado 0 - Esperando Disponibilidad del Robot

flag = 1 | Recogida de Datos

En este estado comienza la recogida de datos de la diadema durante 1 segundo, cuando acabe, se elegirá al comando que durante ese segundo ha aparecido más veces, esto se realizará dentro de la función `choose_action()` a la que se le pasará como argumento una lista con todos los datos recogidos durante ese segundo.

Si la acción elegida es 'neutral' se reiniciará el proceso de recogida de datos.

Si la acción elegida es cualquier otra, se pasará al siguiente estado asignado **flag = 2.**

```
# Recogida tupla de datos y almacenamiento en variable 'data'
data = kwargs.get('data')
        if self.flag == 1:
            if(time.time() <= 1+self.tiempo_ini):

#-----------------------Eleccion de comando facial -------------------------
                self.eyeAct = data.get('eyeAct') #guardamos en self.eyeAct el
comando que estamos mandando
                self.uAct = data.get('uAct')
                self.lAct = data.get('lAct')
                self.uPow = float(data.get('uPow'))
                self.lPow = float(data.get('lPow'))

                if self.uPow >= 0.5:
                    self.str = self.uAct
                elif self.lPow >=0.5:
                    self.str = self.lAct
```

```
                else: self.str = self.eyeAct
#-------------------------------------------------------------------------
                self.lista_fac.append(self.str) #añado cada comando enviado
por la diadema a una lista durante 1 segundo
            else:
                self.str = self.choose_action(self.lista_fac) #cuando ha
pasado el tiempo elijo la accion que más votos tenga

                if self.str == 'neutral' or self.str == 'low_intensity':
                    self.tiempo_ini = time.time()
                    self.lista_fac = []
                else: self.flag = 2
            self.tiempo_ini = time.time()
            self.lista_com = []
```

Código PC: 4. PC_emotiv_py.py | Estado 1 - Recogida de datos + elección de orden

Nota: La línea `data = kwargs.get('data')` se ejecuta en cada iteración del código, no se encuentra dentro de ningún estado, esto es así para evitar errores del código que corten la comunicación. Somos nosotros los que decidimos cuando cogemos esos datos y cuando no.

flag = 2 | Envío de Orden

En este estado, se asocia la acción escogida a la letra correspondiente y se envía a través de una cola (self.cola) que hemos creado para poder enviarle datos al hilo sin la necesidad de crearlo una y otra vez. Para ello hemos utilizado la librería queue.

Una vez que el hilo reciba el mensaje, este lo enviará a través del socket al robot.

Destacar también que es en este instante cuando hacemos uso del evento2, consiguiendo así que el hilo (que estaba bloqueado esperando dicho evento) se desbloquee y pueda seguir con su función.

```
if self.flag == 2:
            if self.str == 'eyes':
                self.action = 'P'
                self.cont_z = self.cont_z+1
            elif self.str == 'surprise':
                self.action = 'W'
                self.cont_w=self.cont_w+1
            elif self.str == clench:
                self.action = 'S'
                self.cont_d=self.cont_d+1
            elif self.str == 'winkR':
                self.action = 'D'
            elif self.str == 'winkL':
                self.action = 'A'
            elif self.str == 'smile':
                self.action = 'Z'

            else: self.action = 'H'
```

```
            print("\n",self.str)

# aquí hacemos el envio de la orden al hilo
            self.cola.put(self.action)
# activamos el evento 2 para que el hilo reciba la accion cuando queremos
            self.evento2.set()
# después de enviar el dato volvemos a poner la bandera a 0
            self.flag = 0
```

Código PC: 5. PC_emotiv_py.py | Estado 2 - Envío de orden

Comando Facial	Comando Mental	Letra	Acción Robot
neutral	neutral	-	-
surprise	push	W	Avance
clench	pull	S	Retroceso
winkR	right	D	Giro Dch
winkL	left	A	Giro Izq
eyes	-	P	Modo Reposo
smile	-	Z	Toma Foto

Tabla 11: Relación Órdenes Mentales/Faciales - Acciones Robot

6.2.3.3 Función choose_action()

La función `choose_action(self,lista)`, tal y como se ha explicado de manera resumida anteriormente, recibe como argumento una **lista** con los datos que se han recogido durante el segundo de muestreo.

En un principio los posibles datos que pueden aparecer en la lista son:

```
Opciones=['neutral','smile','frown','clench','surprise','blink','winkR',
'winkL','lookR','lookL']
```

Para la elección del comando, se recorre la lista de datos y se va rellenando una lista llamada **`frec_palabras`** con las apariciones de cada uno.

Una vez hecho esto se escoge el comando más votado y se devuelve el resultado por la función.

```
def choose_action(self,lista):
        total = len(lista)
        frec_palabras = []
        opciones = ['neutral', 'smile',
'frown','clench','surprise','blink','winkR','winkL','lookR','lookL']
        for w in opciones:
            frec_palabras.append(lista.count(w))
```

```
        #------- diccionario ------------
        # neutral - 0
        # smile - 1
        # frown - 2
        # clench - 3
        # surprise - 4
        # eyes - 5
        # lookR - 6
        # lookL - 7
        # winkR - 8
        # winkL - 9

        #--------------------------------

        frec_ojos = frec_palabras[5]
        # se actualiza la lista 'frec_palabras', ahora solo contempla las
acciones 'neutral', 'smile', 'surprise', 'frown', 'clench', 'eyes', 'lookR' y
'lookL'
        frec_palabras =
[frec_palabras[0],frec_palabras[1],frec_palabras[2],frec_palabras[3],frec_pal
abras[4],frec_ojos,frec_palabras[8],frec_palabras[9],frec_palabras[6],frec_pa
labras[7]]

        dicc = {"0":"neutral", "1":"smile", "2": "frown", "3": "clench", "4":
"surprise", "5": "eyes",
                "6": "lookR", "7": "lookL", "8": "winkR", "9": "winkL"}

        resultados =
{"neutral":str(frec_palabras[0]/total*100),"smile":str(frec_palabras[1]/total
*100), "frown":str(frec_palabras[2]/total*100),

"clench":str(frec_palabras[3]/total*100),"surprise":str(frec_palabras[4]/tota
l*100), "eyes":str(frec_palabras[5]/total*100),
                        "lookR":str(frec_palabras[6]/total*100),
"lookL":str(frec_palabras[7]/total*100),
"winkR":str(frec_palabras[8]/total*100),
"winkL":str(frec_palabras[9]/total*100)}

        votos_max = max(frec_palabras) #variable que almacena n° maximo de
votos
        index = frec_palabras.index(votos_max) #variable que almacena el
índice donde se encuentra la acción más votada
        print("\n", resultados)

        accion  = dicc[str(index)] #accion más votada

        #caso lookR y lookL: será suficiente cuando supere el 35%
        if accion == 'lookR' or accion == 'lookL':
             if float(resultados['lookL']) < 40.0 and
float(resultados['lookR']) < 40.0:
                accion = 'neutral'
        elif accion == 'clench' and float(resultados['clench']) < 85.0:
            accion = 'neutral'


        if float(resultados['winkL']) > 20.0 and float(resultados['winkR'])
== 0.0 :
            accion = 'winkL'
```

```
        elif float(resultados['winkR']) > 20.0 and float(resultados['winkL'])
== 0.0 :
            accion = 'winkR'

        if float(resultados['eyes']) > 20:
            accion = 'eyes'

        print("\n FACIAL: ",resultados[accion]) #imprime el % de votos de la
acción más votada
        print("\n FACIAL: Porcentaje accion más votada:
",str((votos_max/total)*100)," %")
        # if int(resultados[accion]) < 40:
        #     accion = 'low_intensity'

        print("\n FACIAL: la accion que ha ganado es: ",accion)
        return accion
```

Código PC: 6. PC_emotiv_py.py | Función choose_action()

6.2.3.4 Función server().

Esta función estará asociada al hilo que creemos para la conexión PC – Robot.

La función recibe como argumento la cola de ordenes que se le irán enviando, al inicio esta cola estará vacía y se ira actualizando a medida que vayamos enviando órdenes.

El hilo, al que hemos llamado 't', se crea al inicio del programa, en la función __init__() de la clase, de manera que solo se crea una vez.

```
self.t = threading.Thread(target = self.server, args = (self.cola,))
```

Como se ha mencionado, al hilo se le asocia la función server() y se le pasa como argumento una cola dinámica.

Cuando se ejecuta la función se configura el servidor y se crea el socket.

Posteriormente se espera a recibir un mensaje del cliente (el robot), indicando así que este esta encendido y que la conexión se ha establecido de manera exitosa. Se enviará entonces un mensaje al cliente para que este deje de 'llamar' al servidor (con esta estrategia conseguimos que no importe quién aparece primero (servidor o cliente).

Una vez sincronizados servidor y cliente, espero nuevamente a recibir un mensaje (`ready.decode() == 'TRUE':`) que indique que el robot está listo para recibir una orden nueva, es decir, que ya se ha configurado, o bien ha terminado con la acción que estaba realizando anteriormente. Como se explicará más adelante, este estado de "Espera de orden" se manifestará con una combinación de luces determinada.

En este punto el robot está listo para recibir una nueva orden por lo que el hilo deberá activar el evento1 para informar al main que puede iniciar el proceso de recogida de datos.

Posteriormente le hilo se queda esperando la activación del evento2 (`evento2.wait()`), de esta espera saldrá cuando la cola se actualice (nueva orden lista).

En caso de que no llegue el mensaje 'TRUE' indicando que el robot está preparado para una nueva orden, el programa mostrará un mensaje de error y se lo comunicará al cliente (robot) a través del socket.

```
def server(cola):

# Configuración del servidor, creación del socket....
    serverPort = 4444
    soc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    soc.bind(('', serverPort))
    print("El servidor está listo para recibir...")

# Inicializar el tiempo de espera en 3 segundos
    timeout = 3
    sincronizacion = 0

    while sincronizacion == 0: # espero señal del cliente, 'sincronizacion'
se activará cuando el Servidor reciba el mensaje del Cliente

        msg, clientAddress = soc.recvfrom(2048) # El servidor capta la
dirección del cliente y su mensaje
        if msg.decode() == 'call':
            print('Recibo llamda del socket cliente')
            resp = 'msg recibido'
            soc.sendto(resp.encode(), clientAddress) #Enviamos mensaje de
confirmacion al cliente para dedcirle que hemos recibido el mensaje
            print('Envio confirmacion al cliente')
            sincronizacion = 1


    while True and sincronizacion == 1:
# Establecer el tiempo de inicio de la espera
        tiempo_inicio = time.time()
        print('\n Esperando confirmación del Cliente de que está listo para
recibir una orden...')
        ready, clientAddress = soc.recvfrom(2048) #Esperando mensaje de
confirmación del Cliente que nos diga que está listo para recibir una orden

        if ready.decode() == 'TRUE':
            try:
                print('\n Ahora podrás mandar una orden al robot, si el robot
se apaga por inactividad, reinicia el programa...')

# aquí tengo que decirle al main que puede enviar la orden...
                evento1.set() # La variable 'evento1' se activa en este punto
con el objetivo de comunicar la main que puede enviar la orden.

# Espero ahora recibir orden del main
                print("\nHilo: Esperando orden del main")
                evento2.wait()
                evento2.clear()

                Message = cola.get() # Aquí he recibido la orden del main,
ahora el hilo se la pasará a través del socket al robot (cliente)
                soc.sendto(Message.encode(), clientAddress)
                print('\n Hilo: Mensaje enviado al Robot correctamente')

            except KeyboardInterrupt:
                break
```

```
        else:
            print('\n Hilo: Fallo, no ha llegado a través del Cliente el
mensaje que esperábamos')
# Si ha pasado más de 3 segundos desde que se inició la espera, enviar el
mensaje al cliente
            tiempo_actual = time.time()
            tiempo_transcurrido = tiempo_actual - tiempo_inicio
            if tiempo_transcurrido > timeout:
                mensaje = "FAIL" #No se ha recibido un mensaje en 3 segundos
                soc.sendto(mensaje.encode(), clientAddress)
```

Código PC: 7. PC_emotiv_py.py | Función server()

Una vez explicada esta función, damos por concluida la explicación del código desarrollado en Python para la comunicación Emotiv – PC – Robot. Al final del documento se mostrará todo el código donde se podrán ver más detalles, así como la casuística (muy similar) de selección de órdenes para los datos mentales.

6.3 Programación ARD_control.ino.

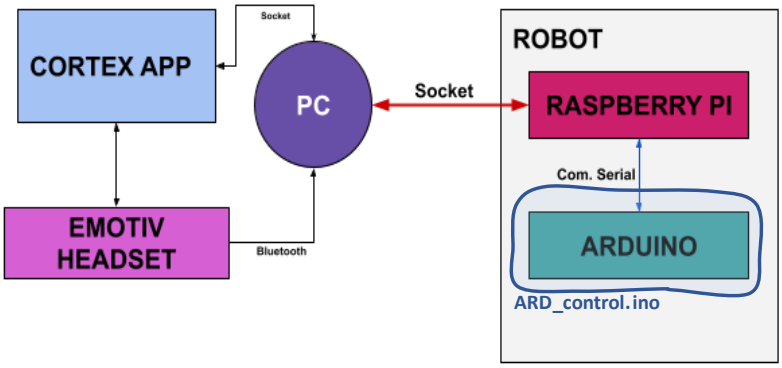

Figura 67: Diagrama Flujo Conexiones | ARD_control.ino

En este fichero se ha desarrollado todo el código correspondiente al **control del robot**, las funciones que se ejecutan al iniciar una nueva sesión con Currito, la recepción de comandos para los movimientos de este, con su correspondiente tiempo de muestreo, el programa que activa el sensor ultrasónico y que frena al robot en caso de aproximarse a menos de 10cm de un obstáculo, y todas las funciones auxiliares que dan vida al robot, control de la tira de leds, movimiento del cuello, cejas… Se ha incluido también en este programa funciones de modo reposo del robot y función de apagado del mismo. En definitiva, es en este fichero es donde se le da vida al robot, inyectándole toda la información que queremos a su cerebro, la placa de Arduino UNO.

Manejo del tiempo en el código

Una de las tareas importantes que se tuvo que abordar a la hora de realizar el código fue la gestión

del tiempo en el programa.

Cuando al robot se le asigna una orden, la variable `dir` toma un valor, que servirá para activar la función `motorGo()` que provocará un set en el movimiento del robot en esa dirección, por tanto el robot seguirá moviéndose en esa dirección hasta que alguien le diga lo contrario. El código de Arduino tiene implementado un sistema de comprobación de tiempo de movimiento, en el que a cada iteración del bucle irá comprobando si se ha cumplido el tiempo que había sido asignado a dicho movimiento.

La manera más eficiente que se ha encontrado para llevar un control del tiempo que debe medirse a cada acción del robot es ir llevando un conteo acumulado del tiempo que pasa en la variable `tiempoactual` de tal manera que cada vez que actualicemos `tiempoactual = millis()` es como si hiciéramos un reset a 0 en el tiempo que el robot ve que una acción lleva realizándose.

Veámoslo con un ejemplo:

Cuando se enciende el robot, la variable `millis()` ya empieza a contar…

En ese tiempo se hace el setup, se espera el chasquido por el micrófono, se asigna la nueva orden… en todo ese tiempo pueden haber pasado 20 segundos (20000ms).

Es por ello que al terminar de hacer esas acciones necesarias para el funcionamiento, pero que debemos excluir del tiempo que queremos controlar, debemos actualizar la variable '`tiempoactual = millis()`', de esta manera ahora '`tiempoactual = 2000`'.

Supongamos que se ha dado la orden de avanzar hacia delante, dicha acción lleva asociada un `t=5000` (5s). El robot ha empezado a avanzar, y a cada iteración verifica la siguiente condición:

```
if( millis() > tiempoactual + t ).
```

Cuando haya pasado 1 segundo de movimiento, `millis()=3000, tiempoactual=2000, t=5000`, por lo que **NO** se habrá cumplido la condición. En este caso, el robot se sigue moviendo y no entra en el '`if`'. Solo se cumplirá cuando hayan pasado 5000ms desde que actualizamos por última vez la variable `tiempaoctual`.

6.3.1 Descripción de la funcionalidad del código.

En este apartado se irá realizando una descripción de los distintos bloques incluidos en el código, se hará un barrido por toda la funcion `void loop()` y se irán analizando las diferentes casuísticas que se pueden dar, el correcto funcionamiento del código se debe a las numerosas condiciones impuestas que abrirán y cerrarán las puertas según el estado del robot, dado que al ser una ejecución en bucle debemos ser nosotros quienes decidamos cuando ejecutar cada orden.

Dentro del bucle, se ha dividido el código en distintos bloques que son:

- **Bloque 1: Inicio**

  ·Recepción de orden

- **Bloque 2: Asignación tiempo ‘t’**
- **Bloque 3: Chequeo de la situación**

  ·Recepción de orden

- **Bloque 4: Ejecución de orden asignada**
- **Bloque 5: Apagado**
- **Bloque 6: Chequeo Robot fuera de peligro.**

Bloque 1: Inicio

Cuando el usuario enciende el robot, este queda esperando una señal aguda, por medio del micrófono, que active su funcionamiento, esta señal se le suele dar con una palmada o un chasquido de dedos.

En ese momento se activa la función **`iniciando()`** que realiza un set-up del robot con una combinación de luces y “despierta” a Curro levantando su cabeza.

Figura 68: LEDS | Inicio Robot

Posteriormente, el robot quedará esperando una orden de movimiento, comienza el **proceso de recepción de orden**, la esperará durante 60 segundos, con una combinación de luces referidas a : 'Recepción de orden en estado **normal**'.

Figura 69: LEDS | Recepción de orden estado Normal

Pasado ese tiempo activará la función **apagando()** que llevará al robot a un estado de reposo inicial del cual solo podrá salir recibiendo nuevamente un chasquido de dedos o una palmada.

```
void loop() {

//------------------- PROCESO INICIO ----------------------------------
  //esperamos recibir un valor alto por el micrófono para iniciar el
programa

  while (x == 2) { //entra porque de inicio x = 2
    VALORMICRO = digitalRead(MICRO);

    if (VALORMICRO == HIGH) {
      Serial.println("inicio");
      iniciando();

//------------- SUBPROCESO RECEPCIÓN DE ORDEN -------------------
     Serial.println("INTRODUZCA DIRECCION(1): ");
      tactual = millis();
      playMelody(durations_listen,melody_listen,tam_listen,1000);
      while (Serial.available() == 0 && reposo == 0) {
        if (millis() > 60000 + tactual) { //si está 60 segundos sin recibir
una orden, entra en modo reposo, esperando el chasquido.
          reposo = 1; //sirve para salir del while
          orden_stop = 1; //sirve para apagar el currito y que entre en
reposo
        }
        if (peligro == 0){
        on_leds(0, 0, 0, 0, 4);
        }
        else if (peligro == 1){
          on_leds(0, 0, 0, 0, 7);
          }
        delay(6);
      }
```

Código Arduino: 1. Bloque recepción de orden (I)

En caso de recibir la orden, que recordemos llegará por el puerto serial con una letra asignada para cada orden, se iniciará un temporizador asociado a cada orden asignada y el robot se pondrá en movimiento.

Letras recibidas por puerto serial Vs variables del programa.

'W' → Avanza hacia delante (`dir = 1`)

'S' → Avanza hacia atrás (`dir = 2`)

'A' → Gira hacia la izquierda (`dir = 3`)

'D' → Gira hacia la derecha (`dir = 4`)

'Z' → Para el robot (`dir = 0`)

'P' → Apaga el robot (`orden_stop = 1`)

Nota:

- '`dir`':variable que determinará la acción de movimiento en el código.
- '`orden_stop`: será 0 en funcionamiento normal, y 1 en caso de que se quiera apagar el robot.
- '`reposo`': valdrá 1 cuando se haya agotado el tiempo de espera de señal.

El código correspondiente:

```
if (reposo == 0) {
        letra = Serial.read();
        Serial.flush();
        switch (letra) {
          case 'W':
            dir = 1;
            break;
          case 'S':
            dir = 2;
            break;
          case 'A':
            dir = 4;
            break;
          case 'D':
            dir = 3;
            break;
          case 'P':
            orden_stop = 1;
            break;
          case 'Z':
            dir = 0;
          default:
            dir = 0;
        }
```

```
      }
      x = 1; //para que salga del while.
    }
    tiempoactual = millis();
  }
```

Código Arduino: 2. Bloque recepción de orden (II)

En este punto del código, el robot ha recibido al orden que debe ejecutar, pero todavía no se ha puesto en movimiento.

Bloque 2: Asignación tiempo 't'.

En este bloque la variable `t`, que fija el tiempo de ejecución de una orden, tomará un valor u otro dependiendo de la dirección de movimiento que el robot deba tomar.

```
//-------------------- PROCESO ASIGNA TIEMPO 't' --------------------
// Determina el tiempo de ejecución de la orden, en caso de giro -> t =
//t2, en caso contrario -> t = t1
  if (dir == 3 || dir == 4) {
    t = t2;
  }
  else t = t1;
```

Código Arduino: 3. Bloque Asignación tiempo 't'

Bloque 3: Chequeo de la situación

A continuación se realiza la comprobación de si el robot debe parar su movimiento, se pueden dar 2 casuísticas:

a. El robot ha estado en movimiento el tiempo `t` asignado sin interrupción.
b. El robot ha topado con un obstáculo a menos de 10cm y por tanto requiere la recepción de una nueva orden.

```
//------------------- PROCESO COMPRUEBA SITUACION -------------------
  if (((millis() >= tiempoactual + t ) && ordendada == 1) || x == 3 ||
flag0 == 1) {

    motorGo(0, pwm); //parada del robot

    delay(1000);
```

Código Arduino: 4. Bloque Chequeo de la situación

Nota: En la primera iteración del bucle, en ningun caso entraría en el 'if' dado que el robot todavía no está en movimiento, pero este bloque de código se coloca físicamente antes del bloque de **ejecución del movimiento** puesto que cuando se cumpla la condicion, es justo después cuando necesitamos ejecutar la nueva orden de movimiento.

El bloque de ejecución de movimiento se explicará despues de este.

B3.Caso 1: El robot ha cumplido con el tiempo especificado o ha interrumpido su movimiento.

En este caso, lo primero que se hace es parar el movimiento del robot (en caso de interrupción ya estará parado) con la función `motorGo()`, posteriormente, se inicia un nuevo proceso de recepción de orden**,** con la siguiente subdivisión:

- Si el robot ha accedido a este bloque por haber terminado el tiempo de ejecución de movimiento, se esperará la nueva orden en modo **normal.** `(peligro = 0)`
- Si el robot ha accedido a este bloque por haber topado con un obstáculo, se esperará la nueva orden en modo **peligro,** con la variación de la combinación de luces de la tira de leds. `(peligro = 1)`

Figura 70. LEDS: Modo Normal Vs Modo Peligro

Tras la recepción de la nueva orden, comienza una nueva etapa de movimiento, esto último conlleva actualizar la variable `tiempoactual = millis()`, fijar la variable `ordendada = 0`, indicando así que el robot está preparado ponerse de nuevo en marcha.

Proceso ejecuta orden.

Se va a iniciar el movimiento, esto se realiza mediante la función `motorGo()`, acto seguido se asocia la combinación de leds que deberán ser encendidos según la orden que se haya dado (avance, retroceso, giro izq, giro dch) con la función `on_leds()`, por último la variable `ordendada = 1` (ya se ha dado la orden) y `flag0 = 0` (damos por finalizada la interrupción).

```
//---------------------- PROCESO EJECUTA ORDEN ---------------------------
//Entra si ha habido alguna interrupción (flag0) o  que sea la primera
//vez que se da la orden(ordendada),
// además, exige que x==1 (variable seguridad) y que no se haya
//solicitado parada (orden_stop == 0)
  if (((flag0 == 1 || ordendada == 0) && x == 1)&& orden_stop==0) {
```

```
    motorGo(dir, pwm);
    on_leds(0, 0, 0, 0, mov); // funcion leds en modo movimiento
    Serial.println(dir);
    delay(1);
    ordendada = 1;
    flag0 = 0;
  }
```

Código Arduino: 5. Bloque Ejecuta Orden

Nota: `x == 1` indicará que nos encontramos en funcionamiento normal (variable de seguridad)

B3.Caso 2: El robot debe seguir en movimiento.

En este caso, no se cumple ninguna de las condiciones de parada y por tanto no será necesario pasar por el proceso de ejecución de orden del caso 1.

Bloque 4: Apagado

El siguiente trozo de código se ejecutará cuando la orden que haya recibido el robot sea de PARADA, es decir, **`orden_stop == 1`**.

Entonces ejecutará una serie de acciones para apagar el robot, limpiar el buffer del puerto serial y devolverlo al estado inicial de reposo, además reseteara la variable `orden_stop` a 0.

```
//---------------------- PROCESO APAGADO ----------------------------
  if (orden_stop == 1) {
      apagando();
      delay(100);
      x = 2;
      orden_stop = 0;
      reposo = 0;
      while (Serial.available() != 0) {
      delay(6);
      Serial.read();
    }
      //exit(1);
    }
```

Código Arduino: 6. Bloque Apagado

Bloque 5: Chequeo Robot fuera de peligro.

A cada iteración del bucle, se calcula la distancia al objeto más cercano con la función `ultrasonic()`, este valor se almacena en la variable `DISTANCIA`.

B5.Caso 1: `DISTANCIA > 30 cm & peligro == 0`

En este caso no ocurre nada, el robot sigue su funcionamiento normal.

Figura 71: LEDS | Verde

B5.Caso 2: `15 < DISTANCIA < 30 cm`

Se activa la variable `peligro = 1`, se reduce la velocidad de los motores (`motorGo()`) y se configura la tira de leds en color ámbar (`on_leds()`).

Figura 72: LEDS | Ámbar

B5.Caso 3: `DISTANCIA < 15 cm`

Se paran los motores, y se activa la variable `flag0 = 1`, que permitirá entrar nuevamente en el bloque de recepción de orden para así poder controlar al robot para salir de la situación de peligro

```
// -------------PROCESO COMPRUEBA ROBOT FUERA DE PELIGRO ----------

//calculo la distancia al objeto
  DISTANCIA = ultrasonic();


  if ((DISTANCIA <= 20 && DISTANCIA >= 0) && x == 1) { //ámbar
    //Serial.println("he llegado al ámbar");
    peligro = 1;
    if (DISTANCIA <= 10 && DISTANCIA >= 0) { //si el objeto esta a menos
de 10cm, parpadeo
      //Serial.println("he llegado al rojo");
    // enciendo leds rojo
      on_leds(0, 0, 0, 0, 3); // como quiero cargar el load = 3, me da
igual lo demás

      if (flag0 == 0) {
        motorGo(0, pwm);
        flag0 = 1;
      }
    }

    //si el objeto esta a menos de 30cm, enciendo los leds de naranja, Y
reduzco la velocidad
    on_leds(0, 0, 0, 0, 2);
    motorGo(dir, 24);
 }
```

Código Arduino: 7. Bloque Chequeo Robot fuera de peligro: Caso 2 y 3

B5.Caso 4: `DISTANCIA > 30cm && peligro == 1:`

En caso de que la variable `peligro == 1` pero de alguna manera ya se haya salido de la situación de peligro y el robot deba recuperar su velocidad normal, se volverá a poner la tira de leds en verde (**`on_leds()`**) y se recuperará la velocidad normal de los motores

(`motorGo()`).

```
else {
   if( peligro == 1){
   motorGo(dir, pwm);
   on_leds(0, 0, 0, 0, mov);
   peligro = 0;
   }
}
```

Código Arduino: 8. Bloque Chequeo Robot fuera de peligro: Caso 4

Una vez llegados a este punto, habremos recorrido el bucle por completo y por tanto descrito todas las casuísticas posibles.

A continuación, se muestra un esquema del código para su mejor comprensión:

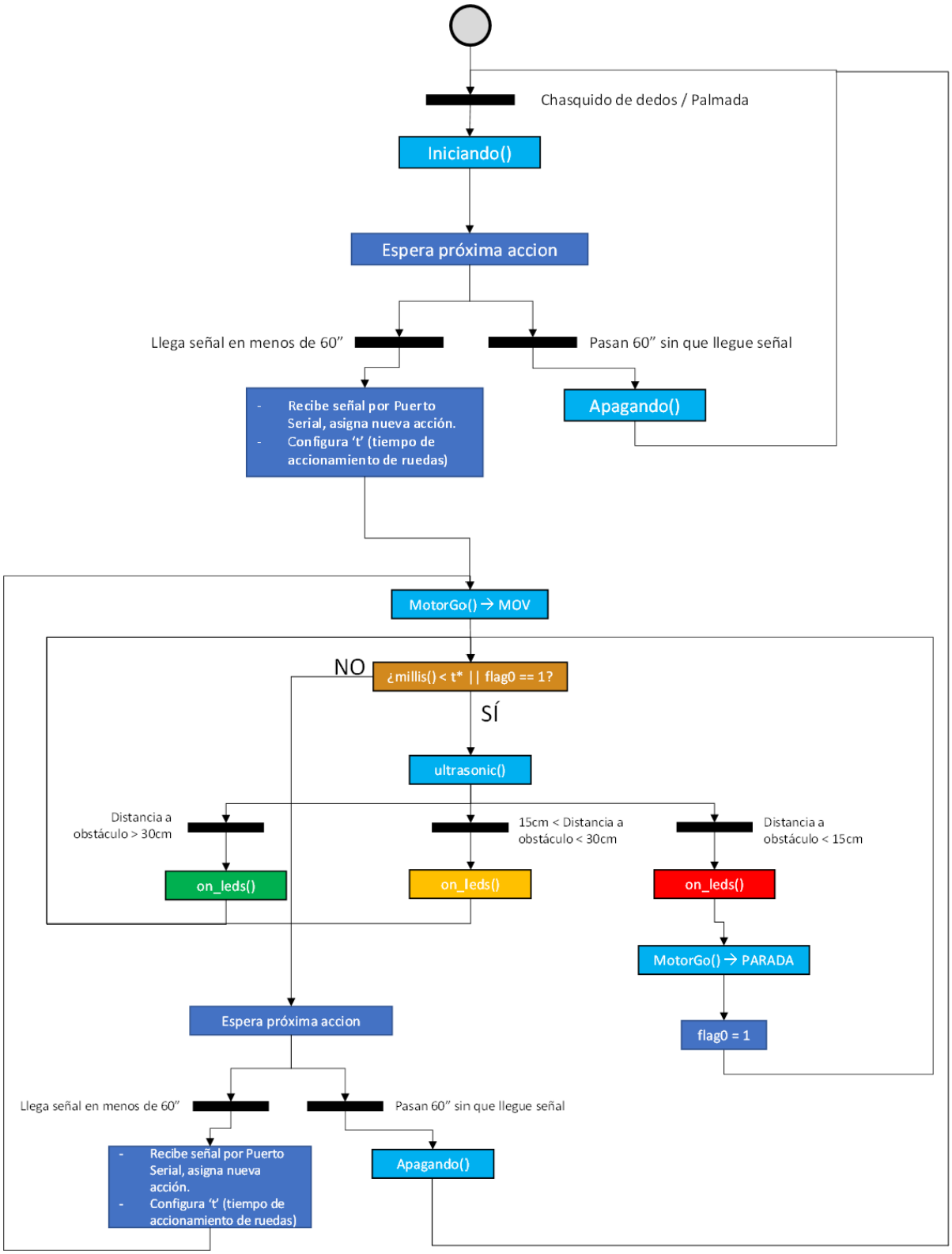

Figura 73: Diagrama Flujo Código Arduino

6.3.2 Funciones Arduino.

En este apartado se describirán las distintas funciones que han sido creadas para la elaboración del código.

6.3.2.1 Iniciando()

La función `iniciando()` se ha creado para recoger las funciones que se desean hacer cuando Currito inicia su funcionamiento:

```
 //------------- funcion iniciando -------------------------
void iniciando() {

  //encendemos leds en verde
  tira.setBrightness(60);
  for (int i = 23; i >= 0; i--) {
    tira.setPixelColor(i, 0, 255, 0);
    tira.show();
    delay(70); // los vamos encendiendo de uno en uno
  }
  playMelody(durations_on,melody_on,tam_on,700);
  //playPacMan();
 for (int i = 0; i < 50; i++) { //cuello currito
    digitalWrite(A5, HIGH);
    delayMicroseconds(PULSOMAX_cuello);
    digitalWrite(A5, LOW);
    delay(10); // esta combinación de comandos hace que el currito gire
               //el motor del cuello a los 180°

  }
}
```

Funciones Arduino: 1. Iniciando()

El primer bucle `for()` lo utilizamos para encender de manera transitoria todos los leds del robot de color verde:

Figura 74: LEDS | Iniciando LEDS: Iniciando

A continuación, se ejecuta la función `playMelody()`, esta función hará sonar una melodía de inicio, por último se realiza el alzamiento de la cabeza del robot con el movimiento del cuello. Esto se hace por medio de un bucle `for()` en el que se lleva el motor del cuello a su posición de 180º.

Con esta secuencia de acciones damos por concluido el proceso de inicio del robot.

6.3.2.2 playMelody()

Esta función la utilizaremos para reproducir una melodía a través del altavoz que Curro tiene incorporado. Se utilizarán melodías para el encendido del robot, el inicio de recepción de nueva orden, y el apagado del robot.

Antes de describir la función, se realiza una breve explicación de como hacer sonar una nota a través del altavoz:

Recordemos que el altavoz está conectado al pin 7 digital de nuestra placa de Arduino, en el código llamaremos a este pin `ALTAVOZ`, esto lo haremos en la declaración de variables al inicio del programa.

```
int ALTAVOZ = 7;
```

Para reproducir un tono en el altavoz, utilizaremos la funcion tone(), la sintaxis de la función es la siguiente: `tone(pin, frecuencia, duración)`

- '`pin`' es el número del pin digital al que está conectado el altavoz
- '`frecuencia`' es la frecuencia del todo en Hz. Por ejemplo, la nota musical 'Do' corresponde a 261,6 Hz
- '`duración`' es la duración del tono en milisegundos.

Después de la instrucción `tone()`, debemos hacer que el micro espere x ms para que la nota pueda reproducirse, si quisiéramos reproducir la nota DO durante 500ms y después la nota LA durante 500ms, debemos escribir:

```
tone(ALTAVOZ, 262, 500); // reproducir 'DO' durante 500ms
delay(500); // retrasar 500ms
tone(ALTAVOZ, 440, 500); // reproducir 'LA' durante 500ms
```

Donde 262 y 440 corresponden en Hz a las notas DO y LA respectivamente.

Para detener la reproducción de un tono podemos utilizar la función `noTone(ALTAVOZ)`

La estructura de la función es la siguiente:

```
void playMelody(int durations[],int melody[],int tam,int t)
```

Donde:

- `durations[]` es la lista de enteros que contiene la duración de cada nota musical
- `melody[]` es la lista de enteros que contiene la frecuencia en Hz de cada nota musical.
- `tam` es un entero que contiene el nº de notas que se van a reproducir.
- `t` es un entero que representa un parámetro en ms

Para el correcto funcionamiento de la función `playMelody()`, debemos:

1.- Incluir la librería pitches.h en nuestro script de Arduino, esta librería contiene una lista de notas musicales asociadas a sus frecuencias correspondientes, de esta manera podremos usar las notas musicales que queramos en vez de andar buscando la equivalencia de las frecuencias. Esta librería se incluirá con los archivos de este documento.

```
#include "pitches.h"
```

2.- Crear una lista `melody_id[]` con las notas musicales de la melodía, en orden.

3.- Crear también una lista `durations_id[]` con las duraciones de cada nota musical

4.- Crear la variable `tam_id` para cada melodía.

Nota: id representa el identificador de cada melodía:

- encendido: id = on
- apagado: id = off
- escucha nueva orden: id = listen

```
 //------------ MELODIA ------------------------------
int melody_listen[] = {
  NOTE_B4, NOTE_B5
};

int durations_listen[] = {
  16, 16
};
int melody_on[] = {
  NOTE_E5, NOTE_E5, REST, NOTE_E5, REST, NOTE_C5, NOTE_E5,
  NOTE_G5, REST
};

int durations_on[] = {
  8, 8, 8, 8, 8, 8, 8,
  4 , 4
};
```

```
int melody_off[] = {
  REST, NOTE_G5, NOTE_FS5, NOTE_F5, NOTE_DS5, NOTE_E5,
  REST, NOTE_GS4, NOTE_A4, NOTE_C4, REST, NOTE_A4, NOTE_C5, NOTE_D5,
  REST, NOTE_DS5, REST, NOTE_D5,
  NOTE_C5, REST,
};

int durations_off[] = {
  4, 8, 8, 8, 4, 8,
  8, 8, 8, 8, 8, 8, 8, 8,
  4, 4, 8, 4,
  2, 2
};

int tam_off = sizeof(durations_off) / sizeof(int);
int tam_on = sizeof(durations_on) / sizeof(int);
int tam_listen = sizeof(durations_listen) / sizeof(int);
```

Funciones Arduino: 2. playMelody(): Declaración variables

5.- La función realiza lo siguiente:

· Calcula la duración en segundos de cada nota musical, divide t ms / duración nota, para una negra (duración nota = 4) → duración en milisegundos (para t = 1000) = 1000/4 = 250

.

Reproduce el tono de la nota correspondiente y su duración calculada.

.

Para que las notas puedan distinguirse, se ha establecido un tiempo de pausa que será un 30% más de la duración de la nota

.

Detiene el tono de la nota en cuestión.

```
void playMelody(int durations[],int melody[],int tam,int t){

  for (int note = 0; note < tam; note++) {
    //to calculate the note duration, take one second divided by the note
type.
    //e.g., quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int duration = t / durations[note];
    tone(ALTAVOZ, melody[note], duration);

    //to distinguish the notes, set a minimum time between them.
    //the note's duration + 30% seems to work well:
    int pauseBetweenNotes = duration * 1.30;
    delay(pauseBetweenNotes);

    //stop the tone playing:
    noTone(ALTAVOZ);
  }
  }
```

Funciones Arduino: 3. playMelody()

Nota: está función ha sido inspirada por este repositorio de GitHub

6.3.2.3 Apagando()

Al ejecutar esta función el robot mostrará una combinación de luces, emitirá un sonido y bajará su cabeza, indicando así que ha entrado en modo reposo.

Figura 75: LEDS: Apagando

```
//---------- Función apagado --------------------------------
void apagando() {

  for (int i = 23; i >= 0; i--) {
    tira.setPixelColor(i, 0, 0, 0);
    tira.show();
  }
  for (int i = 23; i >= 0; i--) {
    tira.setPixelColor(i, 50, 0, 190);
    tira.show();
    delay(60);
  }
  playMelody(durations_off,melody_off,tam_off,700);
  for (int i = 23; i >= 0; i--) {
    tira.setPixelColor(i, 0, 0, 0);
    tira.show();
    delay(60);
  }

  for (int i = 0; i < 50; i++) { //cuello currito
    digitalWrite(A5, HIGH);
    delayMicroseconds(PULSOMIN_cuello);
    digitalWrite(A5, LOW);
    delay(10); // esta combinacion de comandos hace que el currito gire
el motor del cuello a los 180°
    //Serial.println(i);
  }
}
```

Funciones Arduino: 4. apagando()

6.3.2.4 ultrasonic()

Esta función la utilizaremos para calcular la distancia al objeto más cercano.

En primer lugar, definimos los pines de TRIGGER y ECHO para mayor comodidad:

```
int TRIGGER = 2; //pin que envia la señal
int ECHO = 3; //pin que recibe la señal
```

En la funcion void setup() deberemos definir dichos pines como salida y entrada respectivamente:

```
pinMode(TRIGGER, OUTPUT);
pinMode(ECHO, INPUT);
```

```
//----------------- función sensor ultrasónico ----------------------
float ultrasonic() {
  float DURACION;
  digitalWrite(TRIGGER, HIGH);
  delay(1);
  digitalWrite(TRIGGER, LOW); // combinacion de comando para enviar señal

  DURACION = pulseIn(ECHO, HIGH); //recibimos el tiempo que ha tardado la
onda en
                                  //rebotar y llegar de nuevo al sensor
  DISTANCIA = DURACION / 58.8; //valor calculado con la velocidad del
sonido
  return DISTANCIA;
}
```

Funciones Arduino: 5. ultrasonic()

Se envía una señal TRIGGER con un pulso alto durante 1ms, en `pulseIn(ECHO, HIGH)` se almacena el tiempo en microsegundos que ha tardado la onda en rebotar al objeto más cercano.

Para obtener la relación entre DISTANCIA y DURACION, debemos conocer que la velocidad del sonido es de 340 m/s:

$$velocidad\ de\ la\ onda = 2 * \frac{DISTANCIA}{DURACION} = 340\frac{m}{1s} * \frac{1s}{1000000\mu s} * \frac{100cm}{1m} = \frac{0.034cm}{\mu s} \rightarrow$$

$$\rightarrow DISTANCIA = \frac{DURACION}{\frac{2}{0.034}} \rightarrow DISTANCIA = \frac{DURACION}{58.8}$$

La función devuelve la **distancia** al objeto más cercano en **cm**.

6.3.2.5 on_ leds()

Esta función se ha creado con el objetivo de generalizar el uso de la tira de leds y poder crear varias combinaciones de luces predeterminadas.

Antes de describir la función, se llevará a cabo una breve explicación de cómo usar la tira de leds.

Se debe incluir la librería 'Adafruit NeoPixel.h' y declarar la tira de leds, en mi caso he utilizado la variable 'tira':

```
#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel tira = Adafruit_NeoPixel(24, 9, NEO_GRB + NEO_KHZ800);
```

Al trabajar con la tira de leds, utilizaremos principalmente tres funciones:

- **tira.begin()**, debemos llamarla en la función void setup() de nuestro script. Esta función encenderá la tira de leds.
- **tira.setBrightness(brillo)**, la utilizaremos para configurar el brillo de los leds, la llamaremos una vez en la función `void setup()` y en aquellas ocasiones en las que querramos cambiar el brillo de los leds.
- **tira.setPixelColor(led,R,G,B),** llamaremos a esta función cada vez que queramos fijar un color a un led, los leds se numeran del 0 al 23 (hay 24 leds).
- **tira.show(),** sirve para encender los leds.

Si quisiéramos encender el led 5 en rojo durante 50 ms:

```
tira.setPixelColor(5,255,0,0);
tira.show();
delay(50);
```

Figura 76: LEDS | Pixel 5

```
tira.setPixelColor(5,0,0,0);
tira.show;
```

Si quisiéramos encender todos los leds a la vez de color rojo:

```
for (int i=0; i<24; i++){
tira.setPixelColor(i,255,0,0);
}
tira.show;
```

Figura 77: LEDS | Rojo

Si quisiéramos ir encendiendo los leds de verde de manera transitoria recorriendo toda la tira:

```
for (int i=0; i<24; i++){
tira.setPixelColor(i,255,0,0);
delay(80);
}
tira.show;
```

Figura 78: LEDS | Transitorio Verde

Estos son algunos ejemplos de combinaciones de luces, sabiendo estos conceptos básicos, y haciendo uso la función delay() y los bucles for(), existen infinidad de posibilidades.

A continuación, muestro cómo están numerados los leds para el posible interés de aquel usuario que desee probar distintas combinaciones:

Figura 79: LEDS | Numeración píxeles

Una vez explicado el funcionamiento básico de la tira de leds, se muestra a continuación el manual de la función **on_leds()**, donde se explica para qué sirve cada argumento y cuantas combinaciones posibles hay:

```
//---------------- funcion enciende tira leds ----------------------
// Manual de la función:
//  - r, g, b : indican los colores rojo, verde y azul respectivamente
//  - blnk: indica la frecuencia de parpadeo (en ms) blnk = 0 ==> sin
parpadeo
//  - load: permite cargar algunas secuencias ya grabadas
//          load = 0: para una nueva configuracion
//          load = 1: configuracion ultrasónico verde
//          load = 2: configuracion ultrasónico ámbar
//          load = 3: configuracion ultrasónico rojo (parpadeo)
//          load = 4: esperando accion
//          load = 5: giro derecha
//          load = 6: giro izquierda
//          otro load: error

void on_leds(int r, int g, int b, int blnk, int load)
```

Funciones Arduino: 6. Manual función on_leds()

Nota: al cargar alguna de las secuencias grabadas, no importa el valor que tomen el resto de los argumentos.

La función desarrollada es la siguiente:

```
void on_leds(int r, int g, int b, int blnk, int load) {

  switch (load) {

    case 0:
      if (blnk == 0) {
        for (int i = 23; i >= 0; i--) {
          tira.setPixelColor(i, r, g, b);
          tira.show();
        }
      }

      else {
        for (int i = 23; i >= 0; i--) {
          tira.setPixelColor(i, r, g, b);
          tira.show();
        }

        delay(abs(blnk));
        for (int i = 23; i >= 0; i--) {
          tira.setPixelColor(i, 0, 0, 0);
          tira.show();
        }
        delay(abs(blnk));

      }

      break;

    case 1: //Verde

      for (int i = 23; i >= 0; i--) {
        tira.setPixelColor(i, 0, 255, 0);
        tira.show();
      }
      break;

    case 2: //Ambar
      for (int i = 23; i >= 0; i--) {
        tira.setPixelColor(i, 255, 40, 0);
        tira.show();
      }
      break;

    case 3: //rojo
      for (int i = 23; i >= 0; i--) {
        tira.setPixelColor(i, 255, 0, 0);
        tira.show();

      }

      delay(100);
      for (int i = 23; i >= 0; i--) {
        tira.setPixelColor(i, 0, 0, 0);
        tira.show();

      }
      delay(100);
      break;
```

```
  case 4: //Esperanto accion modo 'normal'
    for (int i = 0; i < 24; i++) {
      tira.setPixelColor(i%24,255,0,0);
      tira.setPixelColor((i+1)%24,255,40,0);
      tira.setPixelColor((i+2)%24,0,255,0);
      tira.setPixelColor((i+3)%24,20,0,255);
      tira.setPixelColor((i+4)%24,87,0,100);
  if (option == 1){
      tira.setPixelColor((i+12)%24,255,0,0);
      tira.setPixelColor((i+1+12)%24,255,40,0);
      tira.setPixelColor((i+2+12)%24,0,255,0);
      tira.setPixelColor((i+3+12)%24,20,0,255);
      tira.setPixelColor((i+4+12)%24,87,0,100);
    }

      tira.show();
      delay(50);

      for (int j = 0; j<24; j++){
        tira.setPixelColor(j, 0, 0, 0);
        }
     tira.show();
    }
    break;

  case 5: // giro derecha
    //encendemos los leds de la mitad derecha
    for (int i = 16; i < 24; i++) {
      tira.setPixelColor(i, 0, 255, 0);
      tira.show();
    }


    for (int i = 0; i < 4; i++) {
      tira.setPixelColor(i, 0, 255, 0);
      tira.show();
    }
    break;

  case 6: // giro izq
    //encendemos los leds de la mitad izquierda
    for (int i = 4; i < 16; i++) {
      tira.setPixelColor(i, 0, 255, 0);
      tira.show();
    }
    break;


  case 7: //esperando accion tras choque 'modo peligro'
          for (int i = 0; i < 24; i++) {
      tira.setPixelColor(i%24,255,0,0);
      tira.setPixelColor((i+1)%24,255,0,0);
      tira.setPixelColor((i+2)%24,255,0,0);
      tira.setPixelColor((i+3)%24,255,0,0);
      tira.setPixelColor((i+4)%24,255,0,0);

          if (option == 1){
      tira.setPixelColor((i+12)%24,255,0,0);
      tira.setPixelColor((i+1+12)%24,255,0,0);
      tira.setPixelColor((i+2+12)%24,255,0,0);
      tira.setPixelColor((i+3+12)%24,255,0,0);
      tira.setPixelColor((i+4+12)%24,255,0,0);
```

```
      }
        tira.show();
        delay(35);

        for (int j = 0; j<24; j++){
          tira.setPixelColor(j, 0, 0, 0);
          }
        tira.show();

      }
      break;

    default: //error
      for (int i = 23; i >= 0; i--) {
        tira.setPixelColor(i, 130, 0, 150);
        tira.show();
      }

      delay(80);
      for (int i = 23; i >= 0; i--) {
        tira.setPixelColor(i, 0, 0, 0);
        tira.show();
      }
      delay(80);


  }
}
```

Funciones Arduino: 7. on_leds()

6.3.2.6 toma_foto()

Esta función se encargará de simular el flash de la fotografía que tomará la PiCamera cuando al robot le llegue la orden 'Z' que equivale a tomar una foto.

La función realiza una cuenta atrás acompañada de unos flashes de luz blanca y unos pitidos emitidos por el altavoz. Para sincronizar la cuenta atrás que se realiza en Arduino, con la verdadera cuenta atrás de la PiCamera, el código se queda colgado esperando la entrada por el puerto serial de la letra 'X', esta señal será enviada por la RaspBerry.

Figura 80: Cuenta atrás - toma_foto()

Figura 81: Flash - toma_foto()

```
void toma_foto(){

       t_foto=millis();
       t_melody = millis();
       on_leds(0,0,0,0,8);
       delay(50);
       tone(ALTAVOZ,NOTE_A5,100);
       on_leds(0,0,0,0,9);
       delay(50);
       tone(ALTAVOZ,NOTE_A5,100);
       on_leds(0,0,0,0,8);
       delay(50);
       tone(ALTAVOZ,NOTE_A5,100);
       on_leds(0,0,0,0,9);


         tone(ALTAVOZ,NOTE_A5,400);    // inhabilita el motor
         Serial.println("FOTO");
         on_leds(255,255,255,0,0);
         while (Serial.available() != 0) {
           delay(6);
           Serial.read();

         }
         while(Serial.available() == 0){

           delay(10);
         }
         Serial.println(letra);
         letra = Serial.read();
         Serial.println(letra);

         if (letra == 'X'){
           noTone(ALTAVOZ);
           on_leds(0,0,0,0,0);
           foto = 0;
           }

}
```

Funciones Arduino: 8. toma_foto()

6.3.2.7 motorGo().

Esta función será la encargada de dar movimiento a los motores de las ruedas de nuestro animatrón, la función tiene la siguiente estructura:

```
void motorGo(int direct, int pwm)
```

- `direct`: dirección que queremos que tome el robot: 1 (avanza), 2 (retrocede), 3 (giro derecha), 4 (giro izquierda), 0 (robot parado).
- `pwm`: introducimos un entero con la velocidad a la que queremos que se mueva el robot, siendo 0 la mínima velocidad y 100 la máxima.

En la declaración de variables declaramos:

```
#define avanza 1 //direcciones
#define retrocede 2
#define dch 3
#define izq 4
int pwm = 30;
```

Esto lo hacemos para mayor facilidad en la comprensión del código y las funciones.

Antes de mostrar la función, se va a describir de manera breve como se deben manejar los motores.

```
// MOTOR1 = DERECHO; MOTOR2 = IZQUIERDO
#define MOTOR1_DIRECCION_PIN  12
#define MOTOR2_DIRECCION_PIN  13

#define MOTOR1_VELOC_PIN  10
#define MOTOR2_VELOC_PIN  11
```

La variable de dirección configurará el avance o retroceso de la rueda.

La variable de velocidad configurará la señal pwm de la velocidad de la rueda.

Para mover el motor hacia adelante, ejecutamos las siguientes líneas de código:

```
digitalWrite(MOTOR1_DIRECCION_PIN, HIGH);
digitalWrite(MOTOR2_DIRECCION_PIN, HIGH);
analogWrite(MOTOR1_VELOC_PIN, pwm);
analogWrite(MOTOR2_VELOC_PIN, pwm);
```

Siendo pwm = 30 (p.e)

Una vez explicadas las órdenes básicas para mover las ruedas, se muestra a continuación el código de la función `motorGo()`:

```
void motorGo(int direct, int pwm) {
  pwm = 255 / 100 * (pwm);
  if (direct != 0)
  {
    mov = 1;
    if (direct == avanza)
    {
      digitalWrite(MOTOR1_DIRECCION_PIN, HIGH);
      digitalWrite(MOTOR2_DIRECCION_PIN, HIGH);
    }

    else if (direct == retrocede)
    {
      digitalWrite(MOTOR1_DIRECCION_PIN, LOW);
      digitalWrite(MOTOR2_DIRECCION_PIN, LOW);

    }

    else if (direct == dch)
    {
      mov = 6;
      digitalWrite(MOTOR1_DIRECCION_PIN, LOW);
      digitalWrite(MOTOR2_DIRECCION_PIN, HIGH);

    }

    else if (direct == izq)
    {
      mov = 5;
      digitalWrite(MOTOR1_DIRECCION_PIN, HIGH);
      digitalWrite(MOTOR2_DIRECCION_PIN, LOW);

    }

    analogWrite(MOTOR1_VELOC_PIN, pwm);
    analogWrite(MOTOR2_VELOC_PIN, pwm);

  }

  else if (direct == 0) {
    mov = 0;
    analogWrite(MOTOR1_VELOC_PIN, 0);
    analogWrite(MOTOR2_VELOC_PIN, 0);
  }
}
```

Funciones Arduino: 9. motorGo()

Nota: La variable `mov` se utiliza para llevar el control de qué dirección se ha tomado, servirá para saber qué combinación de luces debe configurarse.

Una vez explicada esta función, habremos hecho un recorrido por todas las funciones de fuente propia que han sido necesarias para la realización de este proyecto.

6.4 Programación RB_py_arduino.py

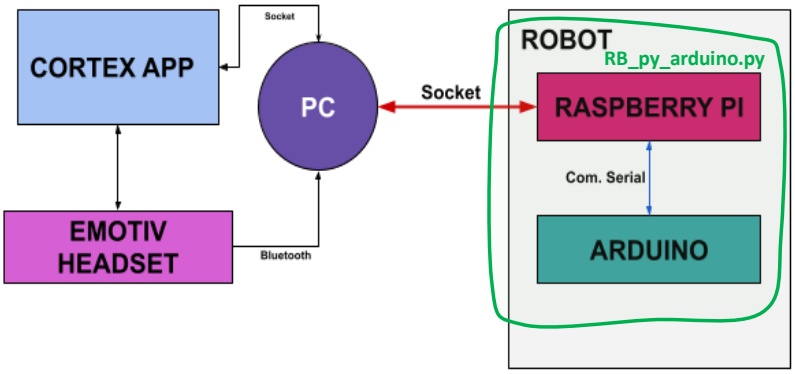

Figura 82: Diagrama Flujo de Conexiones | RB_py_arduino.py

En este apartado, se explicará el funcionamiento del fichero RB_py_arduino.py, que tendrá como función principal hacer de **conector entre el PC y la placa de Arduino**, y **enviar mensajes de texto e imágenes al usuario vía Telegram**. Los mensajes serán de avisos e informes de estados y las imágenes serán las que se tomen con una de las funcionalidades del robot que es "Tomar una Foto".

Este fichero se ejecutará en la Raspberry Pi 4, que recordemos trabaja con Raspbian y permite programación en Python, además, como se ha explicado en capítulos anteriores, gracias a la capacidad de la Raspberry de conexión a internet, podremos comunicar el PC con esta vía sockets, a través de la red a la que estemos conectados.

Por otro lado, la comunicación Raspberry – Arduino se realizará a través de un puerto serial.

6.4.1 Estructura del código

El código se puede estructurar en varias partes y estados:

1. Librerías, inicialización de variables y definición de funciones.

2. Bucle Principal:

- Estado 0: Estableciendo conexión con el Socket y con Arduino
- Estado 1: Conexión establecida → Sincronización con Arduino

3. Bucle Secundario

- Estado 2: Curro disponible para nueva orden → Actualizar situación al servidor.
- Estado 3: Nueva orden antes de 65 segundos → Enviar orden a Arduino.
- Estado 4: Envío Orden de Apagado | Sin mensajes en 65 segundos
- Estado 5: Proceso de Apagado / Reinicio

6.4.2 Desarrollo del código

1. Librerías, inicialización de variables y definición de funciones.

Se añaden las librerías que van a ser utilizadas:

Librería	Funcionalidad
`from picamera import PiCamera`	Uso de la PiCamera
`import serial`	Comunicación Serial Raspberry- Arduino UNO
`import socket`	Creación y uso de sockets para la comunicación PC - Raspberry
`import time`	Uso y manejo del tiempo
`import select`	La utilizaremos para detectar cuando llega un mensaje a través de un socket, permitiéndonos además finalizar la espera de un mensaje pasado un tiempo
`import requests`	Uso API Telegram para envío de texto e imágenes.

Tabla 12: Librerías RB_py_arduino.py

Se inicializan variables como la dirección del servidor y del puerto, variables que utilizaremos más adelantes y que creamos como tuplas vacías, y se crean variables que funcionarán de máquina de estados para nuestro código…

Se realiza también la creación del socket cliente, y se configuran algunos parámetros para el correcto funcionamiento del código.

```
serverName = '192.168.0.23' #nombre del servidor
serverPort = 4444
bucle = 'TRUE'
hora = []
url = []

clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
clientSocket.setblocking(0)
inicio = 0
sincro = 0
msg = 0
sincronizacion = 0

#creamos 3 tuplas vacías, de lectura, escritura y excepción, las variables
que estén dentro de cada tupla,
# causarán una interrupción/aviso en futuras funciones señalizando un
recibimiento de mensaje, o envío del mismo, en su mayoría.

read_fds, write_fds, except_fds = [], [], []
read_fds.append(clientSocket) #habilitamos sensibilizacion a señales lectura
```

Código RaspBerry: 1. Inicialización

Para el envío de mensajes e imágenes por Telegram, se han creado 2 funciones, 'bot_send_text()' y 'bot_send_photo()', las cuáles se definen en esta primera parte del código también.

Estas funciones contienen en su interior las claves para enviar mensajes a un Bot de Telegram que hemos creado. Las funciones contienen en su interior la sintaxis necesaria para realizar el envío de mensajes de texto y de imágenes. Para la sintaxis se han utilizado los métodos **.get** y **.post** de la librería **requests.**

La función bot_send_text() recibe como argumento el mensaje que queremos enviar.

La función bot_send_photo() recibe como argumento la dirección de la imagen que queremos enviar y el texto que queremos que acompañe a la imagen,

A continuación, se adjunta el código correspondiente a la definición de ambas funciones

```
def bot_send_text(bot_message):

    bot_token = '5882923540:AAG97MNZvATCoiQJ9H0QHMqOgRGEoTjH3oA'
    bot_chatID = '829337896'

    send_text = 'https://api.telegram.org/bot' + bot_token +
'/sendMessage?chat_id=' + bot_chatID + '&parse_mode=Markdown&text=' +
bot_message

    response = requests.get(send_text)

    return response
```

Código RaspBerry: 2. Función 'bot_send_text()'

```
def bot_send_photo(image_path, image_caption):
    bot_token = '5882923540:AAG97MNZvATCoiQJ9H0QHMqOgRGEoTjH3oA'
    bot_chatID = '829337896'

    response = requests.post('https://api.telegram.org/bot' + bot_token +
'/sendPhoto',
                files={'photo': (image_path, open(image_path, 'rb'))},
                data={'chat_id': bot_chatID, 'caption': image_caption})
    return response
```

Código RaspBerry: 3. Función 'bot_send_photo()'

2. Bucle Principal:

Una vez que se han creado las variables más elementales, necesarias para la ejecución del código, el programa entra en un bucle infinito.

Estado 0: Estableciendo conexión con el Socket y con Arduino

Se crea el puerto de conexión entre Raspberry y Arduino.

En este estado el programa entra en un bucle dado que la variable 'sincronizacion' está a 0, esta

variable se pondrá a 1 cuando Cliente y Servidor (Raspberry y PC) hayan establecido conexión y se hayan sincronizado.

Para ello, el Cliente enviará cada 10 segundos un mensaje al Servidor, si en esos 10 segundos llega un mensaje de confirmación del Servidor, 'sincronizacion' se pondrá a 1, en caso contrario, seguirá enviando mensajes hasta que se establezca la conexión.

Cuando el cliente envía el mensaje de llamada "call", se ejecuta la función select()

Se utiliza la función select() para esperar a que el socket del cliente esté listo para leer (ready_to_read), escribir (ready_to_write) o manejar excepciones (ready_to_except) durante un máximo de 10 segundos (10). Esta función bloquea la ejecución del programa hasta que se produce uno de estos eventos o hasta que se agota el tiempo límite.

Si salta la variable ready_to_read, significa que ha llegado un mensaje y por tanto podemos pasar a leerlo. Esta ha sido la solución que he encontrado para poder esperar un mensaje durante un tiempo determinado. Estado 0 → Estado 1

```
while True:
# Estado 0: Estableciendo Conexión con Socket y con Arduino
    while sincronizacion == 0:
        serialport = serial.Serial('/dev/ttyUSB0',9600)
        msg = 'call' # el cliente envia al servidor un mensaje que dice:
'call'
        clientSocket.sendto(msg.encode(),(serverName,serverPort))
        print('mensaje -call- enviado')
        ready_to_read, ready_to_write, ready_to_except =
select.select(read_fds,write_fds,except_fds,10)

        if ready_to_read: # ha ocurrido un evento de tipo lectura en menos de
3 segundos
            msg = clientSocket.recv(2048)
            msg = msg.decode()
            if msg == 'msg recibido':
                sincronizacion = 1
                print('El servidor ha recibido el mensaje, conexion
establecida...')
        elif ready_to_except:
            print('Se ha enviado una señal al servidor sin respuesta...')
```

Código RaspBerry: 4. Estado 0: Estableciendo conexión con Socket y con Arduino

3. Bucle Secundario: en él se desarrolla el funcionamiento del código, solo se saldrá de este bucle si se pierde la conexión entre sockets

Estado 1: Conexión establecida → Sincronización con Arduino

Con la variable sincronización activada, damos por concluida la sincronización Cliente-Servidor.

EL programa entra en este estado con la variable **inicio == 0**, esto significa que el robot no está en funcionamiento normal todavía. Esta variable servirá para ejecutar ciertas líneas de código dependiendo del valor que tome. AL entrar en este estado inicio == 0 → Muestra mensaje de inicio, luego se pone a 1 para no volver a mostrar dicho mensaje.

En este estado, el programa lee por el puerto serial los mensajes que va mostrando el Serial de Arduino.

Por su parte, Arduino enviará un mensaje que diga “INTRODUZCA NUEVA ORDEN” cuando el robot esté listo para recibir una nueva orden. De ese mensaje, nosotros captamos las primeras 5 letras:

```
message = serialport.readline(5)
```

Por lo que cuando se reciba por el puerto serial la cadena “INTRO”, sabremos que el robot está listo para recibir una nueva orden. Estado 1 → Estado 2

Estado 2: Curro disponible para nueva orden → Actualizar situación al servidor.

A continuación, el programa debe enviar un mensaje a través del socket al Servidor para indicarle de esta nueva situación. Una vez hecho esto, quedará esperando **recibir una nueva orden para el robot.**

```
# Estado 1: Conexión establecida → Sincronización con Arduino
       if inicio == 0: #para mostrar mensaje la primera vez que entra
            inicio = 1
            nfoto = 0
            time_ejec = time.time()
            print('\n Esperando que Curro despierte...')

        message = serialport.readline(5)
        message = message.decode("utf-8")


# Estado 2: Curro disponible para nueva orden → Actualizar situación al
servidor.
        if message == "INTRO":
            print('he recibido INTRO')
    #         ready_to_read, ready_to_write, ready_to_except =
select.select(read_fds,write_fds,except_fds)
    #         if ready_to_write:
            ready = 'TRUE'
            clientSocket.sendto(ready.encode(),(serverName,serverPort))
            #test_bot = bot_send_text('Hola, soy Currito, el robot
animatrónico de Ángel, acabo de despertar...')
            print("\n Esperando mensaje por socket...")
            okay = select.select([clientSocket],[],[],65)
```

Código RaspBerry: 5. Estados 1 y 2: Conexión establecida, Sincronización con Arduino, Robot esperando orden, Actualizar situación al Servidor.

Una vez más, utilizamos la función select() para esperar la llegada de un mensaje por el socket, esta vez el tiempo de espera es de 65 segundos, si el servidor (PC) no envía una nueva orden tras 65 segundos, se iniciará el proceso de apagado del robot. Estado 2 → Estado 4

En caso de que recibamos la orden de manera normal, antes de 65 segundos, pasamos al estado 3. Estado 2 → Estado 3

Estado 3: Nueva orden antes de 65 segundos → Enviar orden a Arduino.

En este estado recibimos el carácter que contiene la orden que queremos enviar al robot: 'A', 'Z'…

La decodificamos y la volvemos a codificar en "utf-8" para enviarla por el puerto serial a Arduino (que desde que envió la señal de que estaba listo para recibir, se ha quedado esperando una nueva orden).

```
action1 = action.encode("utf-8")
serialport.write(action1)
```

Entre las acciones que pueden llegar encontramos 2 tipos:

- No requieren ninguna funcionalidad de la RaspBerry.

Estas son las acciones avanzar, retroceder, giro derecha, giro izquierda y apagado.
Estas acciones dependen únicamente de actuadores que manejados por Arduino.

- Requieren funcionalidad de la RaspBerry:

Otra de las acciones que hemos incluido en el robot es la de hacer una fotografía, esta acción, que se ejecuta con la letra 'Z', involucra tanto al robot, que inicia una combinación de luces simulando el efecto de la cuenta atrás y del flash, como a la PiCamera instalada en la RaspBerry, que es el elemento que realiza la foto propiamente dicha.

Sincronizando estos dos procesos (cuenta atrás + flash y toma de foto) podemos crear un efecto muy similar al que tendría una cámara de fotos.

La diferencia con las otras acciones es que esta requiere una sincronización de Arduino y RaspBerry.

El procedimiento de toma de foto se explica a continuación en los comentarios del código. Este se muestra en una zona resaltada en gris en el código que se muestra a continuación

```
#Estado 3: Funcionamiento Normal | Recibo mensaje antes de 65 segundos
      if okay[0]:#significa que he recibido algun dato en menos de 90 seg,
okay[0] almacena una lista con el descriptor de archivo 'clientSocket' si está
listo para lectura
          action, serverAddress = clientSocket.recvfrom(2048)
          print("\n Mensaje recibido...")
          action = action.decode()
          print(action)
          if action == 'NoTrue':
             print('\n entro aqui y no deberia')
          else:
             print(action)
             print('X')
             action1 = action.encode("utf-8")
             serialport.write(action1)
```

```
             if action == 'Z':
#se quiere tomar una foto, se le ha comunicado al robot y este comienza con la
cuenta atrás
                print('\n he llegado a la accion == z')
                camera = PiCamera() # iniciamos la cámara
# se inicia el preview
```

```
                camera.start_preview(fullscreen=False, window=(30,30,320,240))
                  while message != "FOTO":
# entramos en un while en el que leemos los mensajes mostrados por el puerto
serial de Arduino, cuando llegue el mensaje "FOTO", coincidirá con el final de
la cuenta atrás y se dará paso a la captura de la foto.
                     message = serialport.readline(4)
                     message = message.decode("utf-8")
                  if message == "FOTO":
# llega el mensaje "FOTO" desde Arduino, se toma la captura
                     camera.capture('/home/pi/imagen'+str(nfoto)+'.jpg')
                     t = time.localtime()
# se guarda la hora a la que se hizo la foto
                     print(nfoto)
                     print(f"{t.tm_hour:02}:{t.tm_min:02}:{t.tm_sec:02}")
                    hora.append(f"{t.tm_hour:02}:{t.tm_min:02}:{t.tm_sec:02}")
                     print(hora[nfoto])
                     url.append('/home/pi/imagen'+str(nfoto)+'.jpg')
                     print(url[nfoto])

# se guarda en tuplas la url de la imagen y a la hora a la que se tomó,
posteriormente se enviarán por Telegram

                     nfoto = nfoto +  1
                     camera.stop_preview()
                     camera.close()
                     print('\nFoto realizada con éxito')
# se envía señal al robot de que la foto ya se ha tomado
                     action = 'X'
                     action1 = action.encode("utf-8")
                     serialport.write(action1)
```

Código RaspBerry: 6. Estado 3: Funcionamiento Normal

Una vez terminado el proceso de envío de orden, sea con foto o no, el programa vuelve al Estado 1 y espera a recibir nuevamente una señal del robot indicando que está listo para recibir una nueva orden. Estado 3 → Estado 1.

Estado 4: Sin nueva orden en 65 segundos → Envío señal de Apagado

En este estado nos encontraremos cuando no haya llegado una orden en 65 segundos, entonces imprimiremos un mensaje por pantalla (aunque este no se verá en el funcionamiento final pues la RaspBerry irá dentro del robot) y asignamos la variable inicio a 2, **inicio = 2.**

```
# Estado 4 (I) : Envío Orden de Apagado |  Sin mensajes en 65 segundos
            else:
                print("\n Han pasado 65 segundos sin recibir ninguna orden")
                inicio = 2
```

Código RaspBerry: 7. Estado 4 (I): Envío señal de Apagado

Esto permitirá ejecutar unas líneas del bucle que hasta ahora no se habían ejecutado. Estas líneas corresponden al proceso de apagado, que coinciden con la continuación del estado 4.

```
# Estado 4 (II): Proceso de Apagado/ Reinicio
        if inicio == 2:
            action = 'P'
            action1 = action.encode("utf-8")
            serialport.write(action1)
            serialport.close()
            for i in range(len(url)):
                print(str(i))
                test_bot = bot_send_photo(str(url[i]),'Foto capturada a las'+
str(hora[i]))
            hora.clear()
            url.clear()
            msg = 'Curro Apagado'
            clientSocket.sendto(msg.encode(),(serverName,serverPort))
            test_bot = bot_send_text('Adiós, hasta la próxima...')
            sincronizacion = 0
            inicio = 0
            break
```

Código RaspBerry: 8. Estado 4 (II): Proceso de Apagado/Reinicio

En este bloque enviamos a través del puerto serial la orden de parada (letra 'P', el robot comenzará a ejecutar el proceso de apagado: combinación de luces, movimiento de cuello, melodía…), cerramos el puerto serial, y enviamos las fotos que hayamos tomado durante la sesión y los mensajes de despedida al usuario a través de Telegram con las funciones bot_send_photo() y bot_send_text().

Por último, reseteamos las señales '**sincronización**' e '**inicio**' a 0 y ejecutamos un **break**, de esta manera salimos del bucle secundario (el que llevaba a cabo el proceso normal), deberemos entonces volver a establecer la conexión con el socket y con Arduino. Estado 4 → Estado 0

Nota: En el programa que se esté ejecutando en el PC (servidor) hemos fijado que cuando han pasado 65 segundos sin enviar una orden, debemos reiniciar el programa, de esta manera el servidor inicia también la conexión de nuevo.

Se muestra a continuación un diagrama de flujo que resumen el funcionamiento del código y muestra los distintos estados por los que transcurre el programa:

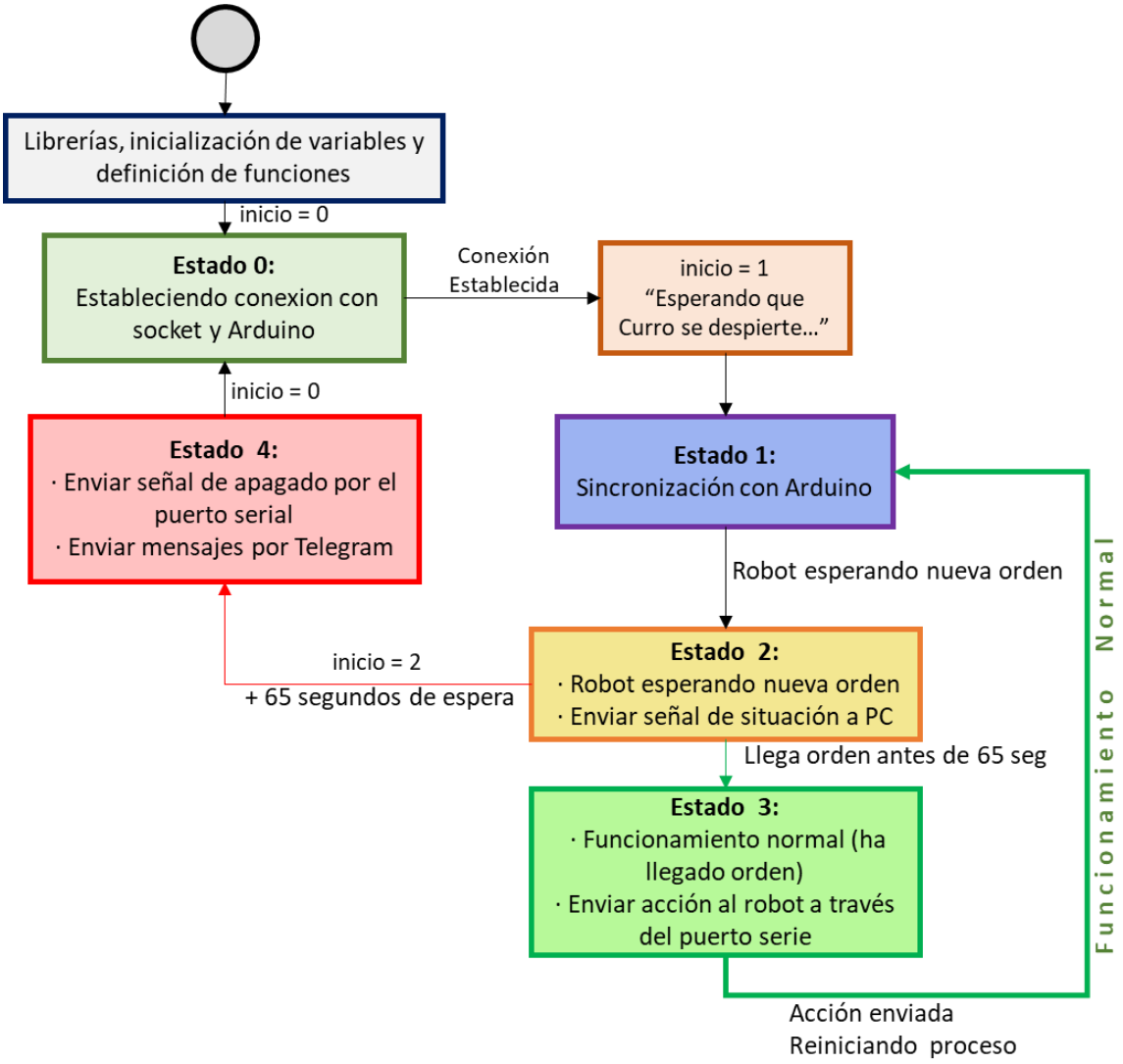

Figura 83: Diagrama Flujo Código Raspberry Pi

Una vez explicado este programa, habremos concluido con la explicación de todos los programas que se ejecutan durante el funcionamiento de nuestro sistema. Al final del documento, en los anexos, se adjuntarán todos los códigos al completo para que aquel que esté interesado en verlo completo pueda hacerlo.

7 CONCLUSIONES

"Un final es sólo un inicio disfrazado."

-Craig Lounsbrough-

En este capítulo, se desarrollarán las conclusiones obtenidas del proyecto, se analizarán los resultados comparándolos con las expectativas que se tenían, se presentarán los distintos inconvenientes que se han ido encontrando a lo largo del proyecto y se presentarán distintas vías de investigación para futuros proyectos relacionados con el tema que nos ocupa.

7.1 Expectativas → Problemas encontrados → Resultados

Al inicio del proyecto, se presentaban distintos objetivos:

- Elaborar un programa de control de movimiento del robot desde Arduino
- Elaborar un programa de procesamiento de órdenes Emotiv
- Entrenar con la diadema para conseguir envío deliberado de órdenes.
- Realizar la conexión Robot – Emotiv de manera inalámbrica

En resumidas palabras, conseguir un robot teledirigido con la mente

A continuación, se discutirán los éxitos obtenidos frente a los objetivos presentados, así como se presentarán nuevas funcionalidades que se han incluido con vistas a mejorar el potencial del funcionamiento del proyecto.

7.1.1 Elaborar un programa de control de movimiento desde Arduino:

El primer objetivo estaba claro, consistía en programar internamente al robot para dotarlo de movimiento que posteriormente debería ejecutar a través de órdenes enviadas no desde teclado, sino desde la diadema Emotiv. Las primeras etapas del proyecto se centraron en este punto, entender el funcionamiento de los distintos componentes del robot, y elaborar un programa cíclico medianamente sólido que pudiera responder a distintos estímulos.

El programa base consiste en un "despertar" del robot, una recepción de comandos (avance, retroceso, giros, paro, toma foto) con la ejecución de estos, un sistema de control de choque mediante el sensor ultrasónico, con reducción de velocidad frente a objeto cercano, parada inminente frente a obstáculo frontal y avisos mediante combinación de luces y sonidos. En el "programa" inicial no se incluía la acción de tomar una fotografía, finalmente se ha incluido un comando de toma de foto que podrá activarse como el resto de acciones.

Durante esta etapa, se tuvo que lidiar con algunos problemas de hardware (cambio de piezas, reajustes de conexión…), también se ha tenido que contar con la limitación de las prestaciones físicas que aporta el robot, la rueda loca que incorpora en la parte trasera de su estructura, limita la precisión de los movimientos, al no ser movimientos muy largos, no se consigue a veces direccionar correctamente la rueda loca, y el movimiento del robot se ve condicionado por la posición en la que esta estuviera antes de iniciar la acción.

Para solventar esto de la mejor manera, se han llevado a cabo numerosas pruebas de calibración, se ha aumentado la velocidad de giro (de manera que las ruedas tengan más par) y reducido los tiempos de ejecución para que los movimientos permitan un reajuste de dirección y el robot ante una orden de avance, no avance muchos metros sin recibir otra orden.

7.1.2 Elaborar un programa de envío de órdenes de Emotiv:

El segundo paso del proyecto era conocer cómo funcionaba la tecnología Emotiv y cómo se realizaba el procesamiento de los datos. Esta etapa conllevó largas semanas de trabajo pues las funcionalidades de app de estas tecnologías están menos documentadas y dedicadas a un público más específico y técnico. Aun así, se encontró información muy valiosa en guías de la propia empresa Emotiv y en algunos proyectos que también se han realizado con la tecnología EEG de Emotiv.

Tal y cómo se ha explicado a lo largo del proyecto, la estrategia utilizada ha sido modificar un script ya creado para ajustarlo a las necesidades que nuestro proyecto tenía. El principal hándicap que se tuvo fue desgranar un código para averiguar qué partes se podían modificar y qué partes formaban parte de la estructura necesaria para la conexión software – diadema.

7.1.3 Entrenar con la diadema para conseguir envío deliberado de órdenes:

Otro de los puntos más importantes fue el de entrenar los comandos que se pretendían enviar al robot. El primer y principal problema con el que se ha tenido que lidiar durante todo el proceso ha sido la puesta en marcha de la diadema, realizar el encendido, la lubricación de los sensores, y conseguir una buena conectividad ha sido una tarea realmente intensa.

Conseguir una buena conectividad es tan difícil como importante, casi siempre hay algún sensor que no tiene una buena conectividad y esto limita el entrenamiento de las órdenes, dado que la empresa recomienda una conectividad de más del 90%.

Una vez lograda la puesta en marcha de la diadema en cada día de prueba, se procede con el entrenamiento de las órdenes:

- El entrenamiento de comandos **mentales** es una tarea que requiere mucha concentración y un entorno sin apenas estímulos, dado que el cerebro debe estar concentrado en el entrenamiento. A lo largo de todo el proyecto, y cómo se ha explicado en capítulos anteriores, se han probado distintas técnicas de entrenamiento. Los resultados son fascinantes, pero limitados. Los comandos que se han conseguido entrenar son escasos en números, por lo que un control del robot basado en el envío de órdenes por comandos mentales carece de sentido, quizás en futuros proyectos se podrían asociar ciertas acciones a algunas emociones, pero de manera puntual.
- Por otra parte, el entrenamiento de comandos por medio de **expresiones faciales** ha sido nuestra principal vía de trabajo, cuando inicialmente estaba en un segundo plano.

  Se cuenta con varias ventajas frente al entrenamiento mental:

  - La conectividad de los sensores es importante, pero no tanto como en el caso anterior, las órdenes en este caso están asociadas a estímulos musculares que podríamos decir son más robustos.
  - El número de órdenes que se pueden llegar a mandar son mayores, de hasta 5

o 6 comandos diferentes.

- La concentración que se requiere es menor, una vez que se ha entrenado, a la hora de ejecutar las órdenes, el sujeto puede estar hablando, escuchando, andando… mientras realiza el control del robot.

7.1.4 Realizar la conexión Robot – Emotiv de manera inalámbrica:

Al inicio del proyecto, se pretendía realizar la conexión Robot – Emotiv por medio del software de ROS (Robot Operating System), un framework para el desarrollo software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo, por lo que a priori permitiría esa fusión Arduino – Emotiv. La idea era que la RaspBerry instalada en el robot trabajara con el sistema operativo de ROS y recibiera directamente los datos de la diadema, para que pudiera enviárselos así a la placa de Arduino.

Durante varias semanas estuve aprendiendo cómo funcionaba el software e investigando acerca de cómo iba a ser la estrategia de ejecución.

Finalmente esas semanas de trabajo no fueron fructíferas, se encontraron muchos inconvenientes tanto de instalación como de comunicación con el programa en sí, además de contar con muy poca información de Emotiv trabajando en ROS.

La conexión Robot – Emotiv se realizó finalmente sin ROS, se llegó a la conclusión de que no era necesario, el objetivo principal era lograr una comunicación inalámbrica entre el robot y el usuario.

- En el PC se tenía el programa que procesaba los datos de la diadema y podía convertirlos en datos que el robot pudiera recibir
- En el robot se contaba con la placa de Arduino que podía recibir los datos por el puerto serie.
- La solución fue crear un tercer nodo, la Raspberry Pi, que haría de puente entre el PC y Arduino. La Raspberry Pi cuenta con conexión a internet, por tanto recibe los datos enviados por el PC y se comunica con él a través de sockets, por otra parte, se comunica con Arduino a través del puerto serie, esto es posible ya que ambas placas van incorporadas en el robot.

  El ”precio” que se ha tenido que pagar para lograr la conexión inalámbrica ha sido la necesidad de tener un ordenador operando mientras se controla el robot. Se ha considerado que es un problema menor y que no limita las funcionalidades del proyecto.

En la ejecución de este tercer programa “puente” se ha tenido que lidiar principalmente con problemas de sincronización entre todos los programas para que así pudiera funcionar como uno solo. Para ello se han llevado a cabo distintas técnicas de sincronización, consiguiendo así un sistema medianamente robusto capaz de responder ante problemas de conexión y fallos de sincronización.

Como resumen final de este apartado, se ha conseguido un robot dotado de movimientos básicos de avance, giro y retroceso, sistema de seguridad antichoque, cámara de fotos, y comunicación con el usuario vía Telegram controlado por medio de expresiones faciales, como la sonrisa, el asombro, el guiño de ojos, la mirada… de manera inalámbrica.

7.2 Posibles mejoras y futuras vías de investigación

Se presentan a continuación algunas sugerencias para la mejora del funcionamiento del robot, así como algunas ideas que complementarían y mejorarían el proyecto realizado.

Si se pretende que el robot pueda ejecutar movimientos precisos, convendría mejorar el agarre de sus ruedas, la distribución del peso, y encontrar una mejor solución para la rueda trasera. Se podría también incluir sensores ultrasónicos en la parte trasera, de esta manera podría tener un mejor conocimiento del entorno y que no estuviera limitado únicamente a ir de frente.

Como posibles mejoras, sería interesante lograr un control del robot que combinara órdenes por expresiones faciales y mentales de manera simultánea, de esta manera se aumentaría el número de ordenes que se puedan enviar. Otra de las posibles mejoras sería rediseñar el sistema de control para que realizara el muestreo de recepción de orden a una frecuencia mayor, de este modo no se apreciaría el tiempo de recepción de orden y la respuesta sería más fluida, para ello se propone además incluir algún tipo de botón de confirmación para que el sujeto decida cuando quiere enviar órdenes o cuando no quiere hacerlo por si tuviera que estornudar (por ejemplo). Actualmente el tiempo de ciclo es de aproximadamente 1 segundo.

Se propone también trabajar con el siguiente modelo de diadema Emotiv, que no requiere de lubricación sensorial y permite una mayor robustez en cuanto a conectividad, además de aportar mayor comodidad al usuario.

A modo de conclusión, agradecer la oportunidad de haber trabajado en este proyecto que me ha permitido desarrollar mis habilidades en programación, gestión y solución de problemas. He podido conocer con profundidad el control mediante EEG, una tecnología fascinante y con mucho potencial en la cuál espero trabajar en un futuro, superando así los grandes resultados obtenidos hasta ahora.

7.3 Enlace a vídeo demostrativo

A continuación, adjunto el enlace a un pequeño vídeo que demuestra el funcionamiento del Robot Curro: https://www.youtube.com/watch?v=PKaFVvPZU-Q

REFERENCIAS

[1] Gentiletti, G. G., Tabernig, C. B., & Acevedo, R. C. (2007, Junio). *Interfaz Cerebro - Computadora: Estado del arte y desarrollo en Argentina*. Revista Argentina de Bioingeniería, 13(1), 22-24. https:/www.researchgate.net/profile/Ruben-Acevedo-4/publication/262568028_Interfaz_Cerebro_-_Computadora_Estado_del_arte_y_desarrollo_en_Argentina/links/00b495382773756c22000000/Interfaz-Cerebro-Computador

[2] *The Introductory Guide to EEG (Electroencephalograpgy)*, Emotiv. https://www.emotiv.com/eeg-guide/

[3] *Interfaces Cerebro-Computador. (2018)*. Bitbrain. https://www.bitbrain.com/es/aplicaciones/interfaces-cerebro-computador

[4] EPOC X User Manual. (n.d.). *Package contents - EPOC X User Manual*. https://emotiv.gitbook.io/epoc-x-user-manual/getting-started/package-contents

[5] EMOTIV Home. https://emotiv.gitbook.io/emotiv-home/

[6] Haldón, J. (2021). *Puesta en marcha y programación de un robot animatrónico*. [Trabajo de Fin de Grado Inédito]. Universidad de Sevilla

[7] American Electroencephalographic Society. *"Guideline thirteen: Guidelines for standard electrode position nomenclature"*. Journal of Clinical Neurophysiology, pp. 25:111-113. 1994.

[8] Control y Robótica con EduBásica. (n.d.). http://platea.pntic.mec.es/~mhidalgo/edubasica/01arduino/arduino01.html

[9] Arduino sensor shield v5.0 (ref: 0203) – electronperdido.com. (n.d.). Electron Perdido. https://electronperdido.com/shop/modulos-expansion/shields-modulos-expansion/arduino-sensor-shield-v5-0/

[10] Anillo NeoPixel, 24 LED's – WS2812 5050 RGB. (n.d.). SANDOROBOTICS. https://sandorobotics.com/producto/com-12665/

[11] HXT12K Metal Gear Servo 10kg / 0.16sec / 55g. (n.d.). Hobbyking. https://hobbyking.com/es_es/hxt-10kg-servo-metal-gear-10kg-0-16sec-55g.html

[12] Baterías LiPo, características y cuidados! - DynamoElectronics. (n.d.). Dynamo Electronics. https://dynamoelectronics.com/baterias-lipo-caracteristicas-y-cuidados/

[13] Playing popular songs with Arduino and a buzzer. (Agosto, 2022). HiBit. https://www.hibit.dev/posts/62/playing-popular-songs-with-arduino-and-a-buzzer

[14] Carbonell, L. (3 de Abril, 2019). *Un bot en Python para Telegram (y en una sola línea)*. Atareao. https://atareao.es/tutorial/crea-tu-propio-bot-para-telegram/bot-en-python-para-telegram/

[15] *What are the detections based on? How were the algorithms created?* (31 de Mayo, 2019). EMOTIV. https://www.emotiv.com/knowledge-base/what-are-the-detections-based-on-how-were-the-algorithms-created/

[16] Qué son las redes neuronales y sus funciones. (22 de Octubre, 2019). ATRIA Innovation. https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/

[17] Saha, S. (15 de Diciembre, 2018) *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Saturn Cloud. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[18] *Raspberry PI 4.* (16 de Julio, 2019). MSRobotics. http://msrobotics.net/index.php/clases-de-raspberry-pi/247-raspberry-pi-4

[19] Santana vega, C. (12 de Noviembre, 2020). *Redes Neuronales Convolucionales. ¿Cómo funcionan?* Youtube. *https://www.youtube.com/watch?v=V8j1oENVz00&t=300s*

[20] Bagnato, J. I. (2018, November 29). *Convolutional Neural Networks: La Teoría explicada en Español*. Aprende Machine Learning. https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

[21] Illustration of Max Pooling and Average Pooling Figure 2 above shows an... | Download Scientific Diagram. (n.d.). ResearchGate. https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451

[22] Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou, *"A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,"* in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 12, pp. 6999-7019, Dec. 2022, doi: 10.1109/TNNLS.2021.3084827. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9451544&isnumber=9966944

[23] Germain, S. (n.d.*). CNN-RNA Convolucional*. Numerentur.org. https://numerentur.org/convolucionales/