Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Control de dispositivos electromecánicos mediante una interfaz cerebro computador

Control of electromechanical devices using brain computer interface

Anabel Díaz Labrador

La Laguna, 14 de julio de 2023

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

C E R T I F I C A

Que la presente memoria titulada:

”*Control de dispositivos electromecánicos mediante una interfaz cerebro computador*”

ha sido realizada bajo su dirección por D. **Anabel Díaz Labrador**, con N.I.F. 43.492.425-T.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

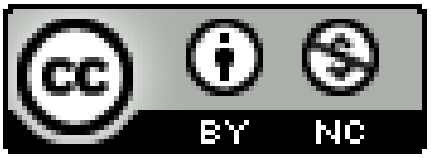
Primero y ante todo, mi más profundo agradecimiento a mi tutor Nacho por ayudarme con los múltiples problemas que me han surgido por el camino.

Muchísimas gracias a Jaime, uno de los pilares fundamentales de mi vida. No solo has sido el mejor compañero de estudios que se podría desear, sino también mi mejor amigo y compañero de vida. Con cada paso que hemos dado juntos en este viaje llamado universidad, se ha fortalecido no solo nuestra capacidad colaborativa, sino también nuestra unión emocional. Jaime, tu apoyo incondicional han sido el faro que me ha guiado en los momentos de incertidumbre. Sin duda, eres la luz en la penumbra de mis días más desafiantes y el compañero perfecto en los más brillantes.

Agradecer también a mis amigos cercanos, Saúl, Julia y Aarón, por ser un gran apoyo emocional en mis momentos grises.

Gracias a mis padres por estar siempre ahí, aguantándome, comprendiéndome y apoyándome con todas sus fuerzas en todos los sentidos. Sin su apoyo no hubiera llegado hasta aquí.

Licencia



Resumen

En este trabajo se explora la integración de una interfaz cerebro computador (BCI) para el control de dispositivos electromecánicos. El objetivo principal es explorar y evaluar las aplicaciones beneficiosas de esta interfaz, especialmente en personas con movilidad reducida. Se logró implementar con éxito la interfaz BCI en un sistema Pan-Tilt y un robot móvil. Se llevaron a cabo pruebas con participantes utilizando una interfaz gráfica específicamente desarrollada para evaluar la efectividad y la experiencia de usuario. Los resultados obtenidos indican que el BCI utilizado, NextMind, es una herramienta realmente útil para la inclusión de personas con limitaciones de movilidad, permitiéndoles lograr una mayor independencia en el control de dispositivos electromecánicos específicos, incluida la telepresencia. Además NextMind ha demostrado tener un gran desempeño en ambientes exteriores siendo especialmente útil en el ámbito de control de dispositivos electromecánicos.

Palabras clave: Interfaz cerebro computador, control de dispositivos electromecánicos, telepresencia, movilidad reducida, inclusión, experiencia de usuario

Abstract

This thesis explores the integration of a brain-computer interface (BCI) for controlling electromechanical devices. The main objective is to explore and evaluate the beneficial applications of this interface, especially for individuals with reduced mobility. The BCI interface was successfully implemented in a pan-tilt system and a mobile robot. Tests were conducted with participants using a specifically developed graphical interface to evaluate effectiveness and user experience. The obtained results indicate that the BCI used, NextMind, is a truly useful tool for the inclusion of individuals with mobility limitations, enabling them to achieve greater independence in controlling specific electromechanical devices, including telepresence. Furthermore, NextMind has shown excellent performance in outdoor environments, making it particularly useful in the field of electromechanical device control.

Keywords: *Brain-computer interface, electromechanical device control, telepresence, reduced mobility, inclusion, user experience*

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Contexto y justificación

Para comenzar a desentrañar el fascinante mundo de la robótica combinada con las interfaces cerebro computador, es importante entender el significado de esto último. Una interfaz cerebro computador (BCI, por sus siglas en inglés) establece una comunicación directa entre la actividad eléctrica del cerebro y dispositivos externos, como ordenadores o dispositivos electromecánicos. Estas interfaces se clasifican en diferentes niveles de invasividad según la proximidad de los electrodos y el tejido cerebral, pudiendo ser no invasivas, parcialmente invasivas o invasivas.

La integración de este tipo de interfaces y dispositivos robóticos presenta numerosas ventajas y aplicaciones. Una de las ventajas más destacadas es su capacidad para ayudar a personas con problemas de movilidad a realizar tareas cotidianas, como el manejo de una silla de ruedas, el control de una prótesis o la interacción con sistemas domóticos.

Este Trabajo de Fin de Grado en Ingeniería Informática, se enfoca en el estudio e integración de una interfaz cerebro computador específica, denominada NextMind, para el control de dispositivos electromecánicos.

NextMind es una startup de neurotecnología que ganó el premio a la Mejor Innovación en CES 2020. El 8 de diciembre de 2020, sacó su kit de desarrollo para su dispositivo BCI llamado por el mismo nombre [12] Posteriormente, fue adquirida por Snap Inc, una reconocida empresa de tecnología y cámaras estadounidense[7] Este acontecimiento ocasionó un gran cierre del producto, disminuyendo el nivel de soporte del kit de desarrollo.

NextMind destaca por su robustez en la detección en la actividad cerebral de la respuesta ante determinados estímulos visuales, lo cual lo convierte en una herramienta confiable de cara al reto que supone un BCI. Además, su asequible precio lo hace accesible para investigadores y desarrolladores. Asimismo, su relativa comodidad de uso garantiza una experiencia satisfactoria para los usuarios durante sesiones prolongadas de interacción con el BCI.

Dada la creciente importancia de las interfaces cerebro-computadora en numerosas aplicaciones, el objetivo principal de este trabajo es investigar y desarrollar una aplicación en el que se use NextMind para el control de dispositivos electromecánicos, con el fin de explorar su viabilidad y potencial en este área.

1.2. Objetivos

Los objetivos del presente trabajo de fin de grado son los siguientes:

1. Desarrollo de una interfaz de usuario para BCI: Esta aplicación permitirá la creación de una interfaz de usuario basada en BCI para el control de dispositivos electromecánicos, como por ejemplo un servomotor o un interruptor. La aplicación proporcionará las herramientas UI necesarias para una interacción fluida entre el usuario y los dispositivos, mediante la detección y el procesamiento de las señales cerebrales.

2. Adaptación a un caso de uso concreto: La aplicación se adaptará para el control de un dispositivo específico, como un sistema de apuntado (pan tilt) o un robot móvil. Este objetivo tomará en consideración las necesidades y requisitos del dispositivo seleccionado.

3. Evaluación de la aplicación: Se realizará una recopilación y análisis de las experiencias de los usuarios al interactuar con el BCI y los dispositivos electromecánicos controlados. Esta evaluación se realizará a través de entrevistas o cuestionarios que recogerán las opiniones de los usuarios y su percepción de la eficacia y facilidad de uso de la aplicación. Los datos recolectados serán analizados de manera cuantitativa y cualitativa para identificar áreas de mejora y de futuro desarrollo. Dentro de la aplicación habrán teclas a modo de flags/contadores para señalizar fallos y eventos ocurridos durante la prueba.

1.3. Estado del arte

El uso de los BCI para el control de dispositivos está ganando popularidad debido a su potencial para mejorar la vida diaria de las personas con movilidad limitada. También aparecen nuevas ideas para su uso habitual, como por ejemplo, siendo parte de la interfaz de usuario en sistemas de realidad virtual (VR) o realidad aumentada (AR). Este proyecto se centra en los métodos que utilizan cascos EEG de tipo malla y, particularmente, en aquellos basados en SSVEP, dada su fiabilidad y alto rendimiento. Específicamente, se utilizará el dispositivo NextMind, una BCI existente en el mercado que utiliza el enfoque SSVEP. La selección de este dispositivo se basa en su accesibilidad, robustez y comodidad.

Un estudio relevante es "*Robotic Arm with Brain*", que utiliza una BCI no invasiva basada en EEG para controlar un brazo robótico [10] Esta investigación proporciona un marco sobre cómo se pueden utilizar los BCIs para controlar dispositivos físicos de manera efectiva.

Por otra parte, el estudio "*A New SSVEP based BCI Application on the Mobile Robot in A Maze Game*", se centra en el desarrollo de un juego de laberinto controlado por una BCI basada en SSVEP con el objetivo de mejorar la calidad de vida de las personas con enfermedad de las neuronas motoras. A la interfaz cerebro-computador se le proporcionan 4 opciones de movimiento: "girar en sentido antihorario", "girar en sentido horario", "desplazarse hacia adelante" y "desplazarse hacia atrás". Para facilitar esas elecciones se utiliza un monitor LCD para mostrar iconos que parpadean a diferentes frecuencias. Esta técnica aprovecha la respuesta de las neuronas en el lóbulo occipital, encargado de procesar los estímulos visuales, que se sincronizan con la frecuencia de la luz percibida por los ojos. Los resultados demostraron que este control basado en SSVEP puede brindar entretenimiento a las personas con dicha enfermedad en el contexto del juego de laberinto.[13] La relación con el presente trabajo radica en el funcionamiento del BCI, que es similar en términos de los estímulos utilizados, así como en las opciones de movimiento proporcionadas.

Finalmente, el estudio "*A Telepresence Mobile Robot Controlled With a Noninvasive Brain–Computer Interface*" presenta un sistema de telepresencia basado en EEG que permite a los usuarios tener presencia en entornos remotos a través de un robot móvil con acceso a Internet. El sistema utiliza una BCI basada en el potencial P300 y un robot móvil con capacidades de navegación autónoma y orientación de la cámara [6] La relación con el trabajo de fin de grado se encuentra en el concepto de la telepresencia, donde se busca utilizar dispositivos controlados por una interfaz cerebro-computadora para lograr una presencia remota en entornos a través de un robot móvil. En este proyecto, se aplicarán ideas similares para permitir a los usuarios controlar objetos y tener una presencia remota mediante la interfaz cerebro-computadora.

La revisión de los estudios anteriores, especialmente aquellos relacionados con BCIs que utilizan el enfoque SSVEP, ha sido muy útil. En particular, el dispositivo NextMind, que se utiliza en este proyecto, ha proporcionado un marco útil para el diseño y la implementación del estudio.

1.4. Metodologías de investigación

Este proyecto adopta una metodología de investigación empírica, implementada a través de pruebas con usuarios. La elección de un diseño empírico se basa en la naturaleza individualizada del uso del casco BCI, que requiere una calibración única para cada individuo. Esta metodología permite analizar las variaciones individuales y determinar su efecto en la calibración y el rendimiento del BCI.

Las pruebas se realizaron con una aplicación que he desarrollado para el control de dispositivos electromecánicos, en este caso un sistema Pan-Tilt con una cámara M5 Stack Timercam.

La aplicación de Pan-Tilt está diseñada para controlar los dos grados de libertad del sistema y ofrecer la visualización en tiempo real del entorno desde la cámara que posee.

Las pruebas para el sistema Pan-Tilt se desarrollaron en un entorno que cuenta con un recorrido de códigos QR dispuestos en distintos planos.

1.4.1. Participantes y reclutamiento

Para este experimento, se reclutaron 16 participantes a través de la universidad y de amigos y conocidos. Se realizó un esfuerzo para asegurar la diversidad de la muestra, lo cual enriquece los resultados al permitir la observación de una gama más amplia de reacciones individuales al BCI. No se proporcionaron incentivos más allá de la oportunidad de experimentar con tecnología innovadora.

1.4.2. Procedimiento experimental

El experimento consta de dos partes principales: la calibración y la prueba. La calibración del BCI se realiza al principio y dura 55 segundos. Este proceso consiste en mostrar una serie de patrones o estímulos visuales a ciertas frecuencias para que el BCI pueda reconocer y calibrarse a las respuestas cerebrales individuales.

Una vez calibrado el BCI, se inicia la prueba, que tiene una duración máxima de diez minutos. Para el control del tiempo se ha incorporado un cronómetro en la aplicación. Los participantes deben recorrer un circuito de códigos QR dispuestos en distintos planos

desde el inicio hasta donde puedan llegar o hasta que lo finalicen.

Los experimentos se realizarán en tres condiciones diferentes: sin luz en interior, con luz en interior y con luz en exterior. Este enfoque permite evaluar la funcionalidad de la aplicación en diversas condiciones de iluminación, ya que este factor parece influir en la precisión de la interfaz cerebro computador.

1.4.3. Recopilación y análisis de datos

Durante las pruebas, se lleva un seguimiento de comportamientos inesperados, especialmente en caso de fallos del BCI. También se controlará el tiempo de calibración del BCI, si los participantes finalizan la prueba y, en caso de ser así, el tiempo que les lleva hacerlo.

Los datos recopilados serán analizados a través de medidas descriptivas básicas, incluyendo el promedio. Además, se buscarán posibles correlaciones entre los diferentes factores, como la edad del participante y el tiempo de calibración del BCI.

1.4.4. Consideraciones éticas

En cuanto a la ética en la investigación, se ha garantizado el consentimiento informado, la privacidad y la confidencialidad de los participantes a través del cuestionario anteriormente mencionado que se entregó al final del experimento.

1.4.5. Gestión de problemas

Se espera que existan variaciones en los resultados entre los distintos individuos. No obstante, todas las pruebas se realizan de manera uniforme para mantener la coherencia del experimento. En caso de problemas o fallos en el BCI durante la experimentación, se ha incorporado un sistema de registro de fallos en la aplicación. Este sistema permite al participante registrar un fallo en la aplicación cuando detecte que el BCI ha fallado, proporcionando un registro contable de los problemas encontrados durante la prueba.

Capítulo 2

Marco Teórico

2.1. Interfaz cerebro-computador: definición y tecnología

Los sistemas de interfaces cerebro-computadora surgieron como una respuesta a la necesidad de agregar nuevas formas de interacción más allá de las convencionales, con el objetivo de brindar soluciones diferentes y efectivas. En particular, los BCIs no invasivos son de interés debido a su accesibilidad, seguridad y facilidad de uso, sin necesidad de procedimientos quirúrgicos. La eficacia de estos sistemas se basa en gran medida en su capacidad para detectar y registrar la actividad eléctrica del cerebro, siendo el electroencefalograma (EEG) uno de los métodos más utilizados. Esto conlleva a pagar el precio de obtener señales más pobres con respecto a los métodos invasivos.

En este sentido, los sistemas BCI enfocan su atención en regiones específicas del cerebro, dependiendo de su propósito. Esta especificidad es de particular relevancia en el contexto de la investigación de BCI.

2.1.1. Metodologías de BCI basados en EEG

Hay dos metodologías que se utilizan actualmente: los impulsos cerebrales relacionados con eventos (ERPs, por sus siglas en inglés) y el electroencefalograma oscilatorio (EEG oscillation).

Los ERP son potenciales eléctricos generados por nuestro cerebro en respuesta a diversos eventos, ya sean cognitivos, sensoriales o motores. Estos ERPs actúan como señales eléctricas que nos brindan una visión interna de cómo el cerebro procesa y responde a estímulos específicos.

El segundo grupo de metodologías se basa en el análisis de la actividad oscilatoria del electroencefalograma (EEG). Estas técnicas implican la monitorización continua de las diferentes frecuencias presentes en la señal cerebral, como las ondas alfa, beta, gamma, delta y theta, que suelen representar el nivel de consciencia/estrés que posee el individuo[1]. A diferencia de los ERPs, las técnicas basadas en EEG oscilatorio no requieren un estímulo externo predefinido para llevar a cabo el análisis adecuado.

Cabe resaltar que ambas metodologías tienen sus propias ventajas y aplicaciones en el estudio de la actividad cerebral. Los ERPs resultan especialmente útiles en investigaciones cognitivas y en la evaluación de funciones sensoriales - motoras. Por otro lado, los BCIs basados en EEG oscilatorio ofrecen la posibilidad de examinar patrones de actividad cerebral en diferentes frecuencias sin necesidad de un estímulo externo controlado, lo

cual facilita su aplicación en contextos más flexibles y naturales.

Dentro de los ERPs se encuentran:

- BCIs basados en EEG P300: Estos BCIs miden el potencial evocado P300 que proporciona información valiosa sobre varios procesos cognitivos, como la memoria y la atención [3]

- BCIs basados en Potencial Evocado Visual de Estado Estacionario, conocido por sus siglas en inglés SSVEP: Este enfoque detecta las respuestas cerebrales a la estimulación visual a frecuencias específicas y ha ganado popularidad debido a su alto rendimiento y robustez [5]

Dentro de los EEG oscilatorio se encuentra:

- BCIs basados en EEG motor imagery (MI-BCI): Las MI-BCIs detectan las señales cerebrales generadas cuando el usuario imagina un movimiento específico. Han mostrado ser útiles en la medicina, particularmente en la rehabilitación motora de personas que han sufrido accidentes cerebrovasculares [2]

Dentro del marco de este Trabajo de fin de grado se usan los BCIs basados en SSVEP.

2.1.2. BCIs basados en SSVEP

Los BCI basados en SSVEP generan una respuesta utilizando frecuencias eléctricas constantes concentradas en el lóbulo occipital ocasionadas por estímulos visuales.

Cuando se encuentra un estímulo visual con una frecuencia específica y registrada por el BCI, las células de la corteza visual se sincronizan y oscilan a la misma frecuencia. Esta sincronización se refleja en la actividad eléctrica del cerebro y se puede medir, como se mencionó anteriormente, utilizando técnicas de electroencefalografía (EEG). Los BCI basados en SSVEP generalmente registran las fluctuaciones eléctricas producidas por la actividad neuronal, mediante electrodos colocados en el cuero cabelludo. Es importante a destacar es que con este método se añade ruido a los resultados y dependiendo de la persona, por lo que hay que realizar cálculos para contrarrestarlo.

Para capturar de manera eficaz la actividad cerebral, se utilizan estos electrodos y se transmiten a una base de datos. Estos electrodos se suelen agrupar en forma de casco para el usuario.

Se utilizan varios algoritmos y métodos de procesamiento de señales para extraer la información de SSVEP (Potenciales Evocados Visuales en Estado Estacionario) del ruido de fondo y de las señales cerebrales irrelevantes. Estos incluyen métodos de filtrado espacial, algoritmos de reconocimiento de frecuencia SSVEP y métodos de aprendizaje automático.

2.1.3. Diferencias entre BCIs SSVEP y NextMind

A pesar de que NextMind es bastante reservado sobre el funcionamiento interno de su dispositivo, hay varios aspectos clave que lo diferencian de los BCIs SSVEP tradicionales.

Arquitectura

NextMind emplea una arquitectura única que se centra en el lóbulo occipital, una región cerebral responsable de la interpretación de la información visual. A diferencia de otros BCIs que requieren un casco completo para cubrir múltiples regiones cerebrales, NextMind es un dispositivo compacto y liviano que se coloca en la parte posterior de la cabeza, proporcionando una mayor comodidad para el usuario.

Sistema Híbrido: Inteligencia Artificial

Una de las características más notables de NextMind es su uso de inteligencia artificial para mejorar la precisión y la eficiencia del dispositivo. NextMind implementa una red neuronal que el usuario debe entrenar inicialmente para que el dispositivo pueda ajustarse mejor a las señales cerebrales específicas del usuario. Este enfoque híbrido facilita una adaptación más personalizada y eficiente a las respuestas cerebrales individuales.

Patrón de Estímulo

NextMind emplea un patrón de estímulo novedoso para potenciar la captación de las respuestas visuales. En lugar de los patrones convencionales utilizados en los BCIs SSVEP, NextMind utiliza una malla de puntos, donde cada “neurotag” parpadea a una frecuencia específica. Esta innovadora forma de estimulación visual puede ayudar a mejorar la detección y el seguimiento de los estímulos visuales por parte del dispositivo.

Un NeuroTag generalmente consiste en un patrón visual o un objeto en una interfaz gráfica que está diseñado para provocar una respuesta neural específica cuando el usuario lo observa. Estas respuestas neuronales pueden ser registradas y decodificadas por la BCI de NextMind, lo que permite al sistema inferir la intención del usuario a través del monitoreo de las reacciones cerebrales a estos neurotags (véase en la figura 2.1)

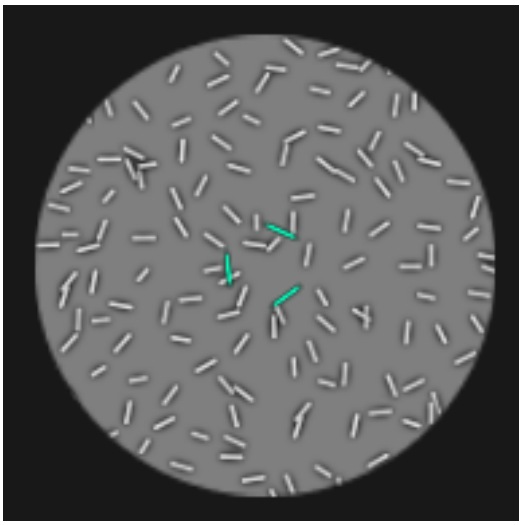


Figura 2.1: Neurotag de NextMind

2.1.4. Modulación por Ancho de Pulso (PWM)

La Modulación por Ancho de Pulso (PWM, por sus siglas en inglés) es una técnica empleada para controlar la cantidad de energía entregada a un dispositivo electrónico

mediante la manipulación del ancho de los pulsos en una señal de onda. En este proyecto, esta técnica es esencial para el control preciso del mecanismo Pan-Tilt.

El rango de ciclo de trabajo (duty cycle) para la PWM generalmente va desde 0 a 32767, esto corresponde a una resolución de 15 bits. Sin embargo, en el caso del control de un servomotor, este rango completo puede no ser aprovechado[11]

En muchos servomotores, un pulso de 1 milisegundo (ms) se traduce en un ángulo de 0 grados, mientras que un pulso de 2 ms se traduce en un ángulo de 180 grados. Estos pulsos se aplican en un período total de 20 ms, que corresponde a una frecuencia de 50 Hz, comúnmente encontrada en servomotores.

Usando la resolución máxima de 15 bits, el valor del duty cycle para un pulso de 1 ms puede ser calculado de la siguiente manera:

$$duty_{1ms} = \frac{1ms}{20ms} \times (2^{15} - 1) \approx 1638$$

Para un pulso de 2 ms, el cálculo es similar:

$$duty_{2ms} = \frac{2ms}{20ms} \times (2^{15} - 1) \approx 3277$$

Por ende, para la mayoría de los servomotores, el rango útil de los valores del ciclo de trabajo estaría aproximadamente entre 1638 y 3277. Cabe señalar que este rango puede variar en función de las especificaciones concretas de cada servomotor, por lo que es imprescindible revisar siempre la documentación correspondiente al dispositivo en cuestión. En muchas ocasiones, es útil comprobar de forma empírica cuáles son los límites reales del servomotor.

Capítulo 3

Herramientas y tecnologías

3.1. BCI NextMind

El BCI NextMind[1] como se ha mencionado anteriormente es una interfaz cerebro-computador revolucionaria. El dispositivo se presenta en forma de un casco compacto y ligero, diseñado y pensado para la comodidad y facilidad de uso del usuario.

El dispositivo NextMind se caracteriza por su diseño inalámbrico, lo que facilita la movilidad y mejora la comodidad del usuario. Este casco viene con una banda ajustable mediante una cinta de velcro, lo que permite adaptarse a diferentes tamaños y formas de cabeza, garantizando así su accesibilidad y funcionalidad para una amplia diversidad de usuarios.

NextMind se distingue por sus nueve electrodos colocados estratégicamente para capturar las señales electroencefalográficas (EEG) desde el lóbulo occipital del cerebro. Estos electrodos registran las respuestas neuronales provenientes de los “neurotags” visuales y envía la respuesta a la computadora del usuario (véase en la figura 3.1)

Figura 3.1: Electrodos del casco NextMind

Una de las características más relevantes de NextMind es su proceso de encriptación de señales EEG. Al encriptar las señales, NextMind evita que terceros no autorizados puedan interpretar o alterar estas señales. Esta estrategia de encriptación plantea desafíos adicionales en el estudio de de su tecnología, ya que limita la disponibilidad de información detallada sobre su funcionamiento interno.

[1] NextMind: `https://ar.snap.com/welcome-nextmind`

Estas señales encriptadas son transmitidas vía Bluetooth y sólo pueden ser procesadas a través de la plataforma de desarrollo Unity 3D a través del uso de la SDK proporcionado por el fabricante de NextMind.

3.2. Unity 3D

Unity3D[2] es una de las plataformas de desarrollo de videojuegos 3D más importantes del mercado. Esta herramienta polifacética no solo se utiliza en la creación de videojuegos, sino también en la producción de simulaciones, experiencias de realidad virtual (VR) y aumentada (AR), y aplicaciones interactivas para diversas industrias, que podrían ser útiles para su implementación con NextMind.

La plataforma utiliza principalmente los lenguajes de programación C# y UnityScript, una variante de JavaScript. Ofrece un entorno de desarrollo visual que permite a los creadores diseñar, personalizar y controlar los entornos virtuales y sus interacciones sin necesidad de programar cada detalle desde cero. Unity3D también incluye una extensa biblioteca de activos y herramientas predefinidas para acelerar el proceso de desarrollo.

En el contexto de este Trabajo de Fin de Grado, Unity3D adquiere una importancia especial por su integración con el dispositivo NextMind, siendo esta plataforma la única que el sistema de desarrollo de NextMind permite utilizar.

3.3. NextMind SDK

El SDK de NextMind[3] para Unity ofrece un conjunto de herramientas diseñadas para facilitar el desarrollo de juegos y aplicaciones que aprovechan la tecnología NextMind. Este SDK proporciona una API de alto nivel que abstrae a los desarrolladores del funcionamiento interno del BCI. Para el desarrollo de este trabajo, se ha hecho uso extensivo de esta SDK.

El SDK de NextMind se centra principalmente en dos componentes esenciales: el NeuroTag y el NeuroManager.

3.3.1. NeuroTags

El NeuroTag es un componente que permite hacer que cualquier objeto en una aplicación sea interactuable con el BCI. En términos prácticos, los NeuroTags son elementos visuales en la interfaz de usuario que el dispositivo NextMind puede reconocer y seguir. Actualmente, el SDK puede manejar hasta 10 NeuroTags activos simultáneamente, aunque se espera que este número aumente en futuras versiones del SDK. En el marco de este proyecto se utilizaron NeuroTags en la interfaz para que mediante la interacción con esos NeuroTags, se enviara una orden determinada a los dispositivos electromecánicos. Un ejemplo de NeuroTag se puede ver en la Figura 2.1 mostrada anteriormente.

3.3.2. NeuroManager

El NeuroManager es el componente que gestiona la comunicación entre los NeuroTags presentes en la escena y el núcleo del motor de NextMind. Su función es facilitar la

[2]Unity3D: `https://unity.com/`

[3]NextMind SDK `https://github.com/Snapchat/NextMind/l`

coordinación entre los diferentes NeuroTags y el procesamiento de las señales cerebrales captadas por el dispositivo NextMind. Este componente es esencial para cualquier proyecto que utilice NextMind.

3.3.3. Otras funcionalidades

Además de los NeuroTags y el NeuroManager, el SDK de NextMind ofrece una amplia gama de funciones adicionales que pueden ser utilizadas para personalizar las aplicaciones. Estas funciones permiten, por ejemplo, obtener información del sensor de NextMind (como el nivel de batería, el contacto, etc.), gestionar el comportamiento de escaneo de Bluetooth, simular entradas, entre otros.

El SDK también incluye varios tipos de "assets", divididos en dos categorías: los "assets" esenciales para construir una aplicación habilitada para NextMind y los "assets" de ejemplo que muestran las mejores prácticas para el uso del SDK.

El SDK también incluye varios tipos de "assets", divididos en dos categorías:

- Assets principales: son todos los archivos esenciales necesarios para crear una aplicación habilitada para NextMind. Encontrará dentro de las librerías principales que exponen las clases principales y algunos activos convenientes (prefabricados, sombreadores, componentes, herramientas, etc.).
- Assets de ejemplo: muestra varios ejemplos de cómo usar el SDK, que enseñan las mejores prácticas (por ejemplo, cómo crear su aplicación de calibración personalizada o cómo etiquetar un objeto).

Estas características adicionales fueron empleadas en el proyecto para mandar una retroalimentación sobre el estado del casco BCI e implementar una calibración personalizada dentro de la aplicación.

3.4. ROS2

El objetivo de este proyecto es controlar sistemas electromecánicos mediante una interfaz de usuario en un ordenador convencional, haciendo uso de una aplicación distribuida con elementos de cómputo en diversos puntos. Dicha configuración requiere de un medio eficiente para comunicar comandos entre estos elementos, y aquí es donde ROS2[4] cobra relevancia.

ROS2 es un conjunto de bibliotecas de software y herramientas que permite la creación de aplicaciones distribuidas, comúnmente utilizadas en el control de robots. Esta segunda generación de ROS se ha desarrollado para mantener y expandir las características más útiles de su predecesor, introduciendo al mismo tiempo mejoras significativas en áreas como el rendimiento, la seguridad y el soporte para sistemas embebidos.

La arquitectura de ROS2 es más modular y flexible, permitiendo a los desarrolladores aprovechar y combinar diferentes bibliotecas y herramientas de manera más eficiente. Además, utiliza un modelo de comunicación basado en la publicación y suscripción, lo que facilita una comunicación eficaz entre diferentes dispositivos electromecánicos.

En el contexto de este trabajo, ROS2 es particularmente relevante debido a su capacidad para integrarse con Unity3D a través de librerías como `ROS2ForUnity` o

[4]ROS2 Humble: `https://docs.ros.org/en/humble/index.html`

`Unity-Robotics-Hub`. De este modo, es posible desarrollar una aplicación en Unity3D que utilice la interacción proporcionada por un BCI para el control de dispositivos robóticos.

3.5. MicroROS

MicroROS[5] es una adaptación de ROS2 especialmente diseñada para llevar sus capacidades a sistemas embebidos. Esta herramienta, optimizada para operar con restricciones de recursos de hardware y energía, permite extender el ecosistema de ROS2 a estos sistemas. Con MicroROS, los desarrolladores pueden programar estos sistemas utilizando las mismas herramientas y procedimientos empleados para las máquinas ROS2 más potentes.

Además, MicroROS se integra de manera nativa con ROS2, permitiendo que los sistemas embebidos equipados con MicroROS interactúen con otros sistemas basados en ROS2, comportándose como nodos de ROS2 a través de la ejecución de un agente, aspecto que se explicará más adelante.

En el contexto de este proyecto, MicroROS ofrece la posibilidad de incorporar directamente ROS2 en un microcontrolador, lo que abre la puerta a la manipulación directa del funcionamiento de los sistemas robóticos a nivel de hardware. Esto proporciona una gran flexibilidad y precisión para la implementación de aplicaciones robóticas complejas.

3.6. Microcontrolador ESP32

Para el control efectivo de dispositivos en proyectos como este, la elección de un microcontrolador adecuado es de gran importancia. Es esencial contar con un componente que gestione eficazmente la comunicación entre los diferentes elementos del sistema y que posea la capacidad suficiente para realizar las tareas requeridas sin comprometer el rendimiento. En este sentido, se ha seleccionado el microcontrolador ESP32[6], desarrollado por Espressif Systems, para este proyecto.

El ESP32 es un microcontrolador de alto rendimiento que se ha ganado reconocimiento en el campo de los sistemas embebidos y el Internet de las Cosas (IoT), gracias a su robusto rendimiento, bajo consumo de energía y versatilidad.

En este proyecto, el ESP32 desempeña el papel de microcontrolador principal. Su combinación de potencia y la incorporación del marco de trabajo de MicroROS le permiten ejecutar un nodo de ROS2, convirtiéndolo en un componente clave para el logro de los objetivos del proyecto.

3.7. Entorno de desarrollo ESP-IDF

Para la programación del microcontrolador ESP32 en este proyecto, se ha elegido el entorno de desarrollo ESP-IDF (Espressif IoT Development Framework)[7]. Esta elección se sustenta en las diversas bibliotecas y herramientas que ESP-IDF proporciona, simplificando la programación de alto rendimiento para estos dispositivos. Además, una

[5] MicroROS: `https://micro.ros.org/`

[6] ESP32 `https://www.espressif.com/en/products/socs/esp32`

[7] ESP-IDF `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/`

característica determinante es su compatibilidad con MicroROS, permitiendo su integración como un componente de ESP-IDF[8].

El ESP-IDF posee un sistema de compilación basado en CMake que permite una configuración del proyecto flexible, la detección automática de dependencias y la compilación cruzada para los microcontroladores ESP32, facilitando el desarrollo del proyecto.

3.7.1. Tests: Unity - ThrowTheSwitch

Para la implementación de pruebas unitarias en el código C del proyecto del microcontrolador, se ha seleccionado Unity - ThrowTheSwitch (U-TTS)[9], un marco de pruebas para el lenguaje de programación C desarrollado por ThrowTheSwitch.org. Su integración por defecto con los proyectos ESP-IDF asegura una compatibilidad perfecta con el microcontrolador ESP32 utilizado en este proyecto.

U-TTS destaca por su diseño ligero y portátil, lo que lo hace adecuado para proyectos embebidos. Su sintaxis simple facilita su uso y, aunque está diseñado para C, U-TTS puede usarse con casi cualquier lenguaje que se compile a través de C, incluyendo C++. Además, es compatible con una variedad de entornos de compilación y plataformas.

En este proyecto, el uso de U-TTS permite realizar una verificación sistemática de la funcionalidad del código escrito, incrementando la fiabilidad del sistema en desarrollo y facilitando el mantenimiento y la detección temprana de errores.

3.8. Pan-Tilt

Para el control de la orientación de una cámara en este proyecto, se utiliza un sistema Pan-Tilt. Este mecanismo permite ajustar con precisión la dirección en la que la cámara está orientada, ampliando el campo de visión y mejorando la versatilidad del sistema. Los movimientos del sistema Pan-Tilt son controlados a través del microcontrolador ESP32, que gestiona las órdenes provenientes de la interfaz de usuario.

El Pan-Tilt utilizado está construido de forma artesanal y se encontraba en el laboratorio del departamento. Se construyó para un artículo científico llamado en inglés “A new approach in controlling the motors of a binocular camera head” [8]. Se ha optado por su uso con la intención de mantener el sistema lo más abierto y adaptable posible.

Este mecanismo está equipado con servomotores HITEC HS-645MG[10], que son reconocidos por su alta resistencia, robustez y precisión. Estos servomotores son capaces de mover y mantener de manera precisa la posición de la cámara, a pesar de las vibraciones o las cargas externas (véase la figura 3.2)

3.9. Cámara

Este proyecto integra una cámara conocida como M5 Stack Timercam [11], que destaca por su capacidad para realizar streaming de las imágenes captadas vía HTTP. Además, está equipada con un software de detección de códigos QR.

[8]Componente microROS `https://github.com/micro-ROS/micro_ros_espidf_component/`

[9]U-TTS pruebas unitarias para C: `https://www.throwtheswitch.org/unity`

[10]HITEC HS-645MG Datasheet `https://media.digikey.com/pdf/Data%20Sheets/Hi-Tech/HS-645MG.pdf`

[11]M5 Stack Timercam `https://m5stack.com/products/timercam-m5stack`

Figura 3.2: Pan-Tilt artesanal

La cámara M5 Stack Timercam es fundamental en este proyecto debido a su capacidad para proporcionar una retroalimentación visual en la interfaz de usuario. El sistema de apuntado se beneficia de un componente que reacciona si se ha logrado un apuntado correcto, permitiendo al usuario percibir de manera clara si están funcionando correctamente sus interacciones con la interfaz (véase la figura 3.3)

Figura 3.3: Cámara M5 Stack Timercam

Capítulo 4

Desarrollo del proyecto

En este capítulo se describe el desarrollo del proyecto. En primer lugar, se verá el entorno general del proyecto, explicando a grandes rasgos como funciona el proyecto en conjunto. Luego se explica la construcción del interfaz de usuario en Unity, donde interviene de forma directa el sistema de interacción cerebro - computador. A continuación se describe el desarrollo del sistema controlado, en este caso un Pan - Tilt equipado con una cámara. El control del sistema Pan-Tilt implica el desarrollo de una aplicación ESP-IDF para un ESP32 montado en una targeta de desarrollo. La aplicación de la cámara inteligente no forma parte de este proyecto de fin de grado, por lo que no se describe aquí aunque s'i se enumeran los requisitos que debe cumplir en relación a las demás partes del proyecto.

4.1. Entorno general del proyecto

Una vez instalado todo el entorno correctamente (véase , es el momento de explicar cómo se integran todos los componentes para poner en funcionamiento el proyecto.

La figura 4.1 presenta un diagrama que representa la visión general del sistema.

Es importante mencionar que todos los componentes de ROS 2 deben estar dentro de la misma red. De esta forma, Unity puede conectarse a la red en la que se está ejecutando el agente de micro-ROS. Esta red es generada por el microcontrolador ESP32, que actúa como punto de acceso.

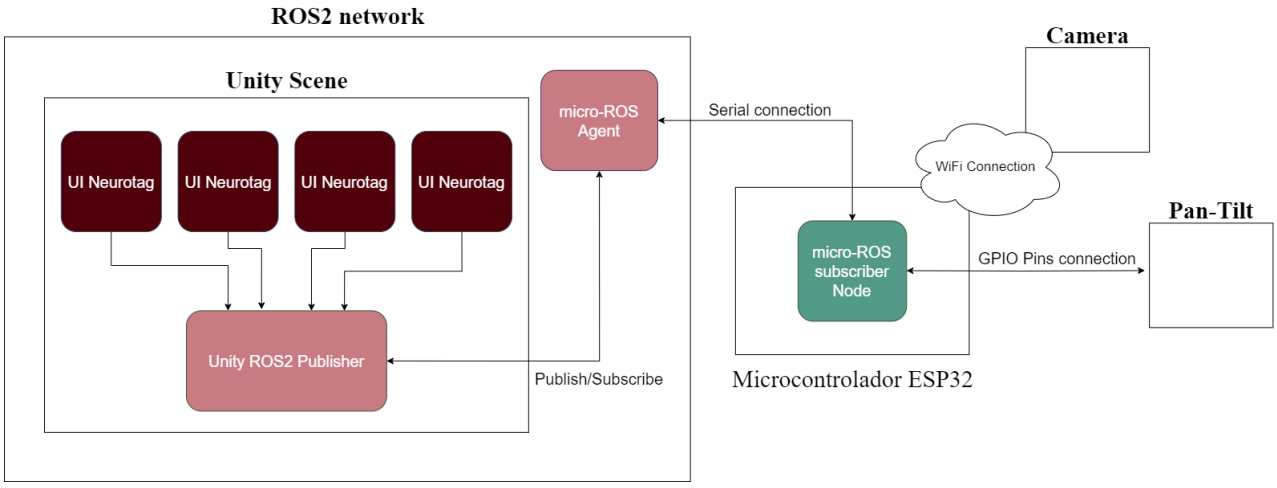


Figura 4.1: Diagrama del funcionamiento del sistema

Con ayuda de la librería Ros2ForUnity, Unity crea un nodo publicador de ROS2 que publica en el tópico “action”. Un tópico o topic en ROS es un nombre que se utiliza para identificar un canal de comunicación a través del cual los nodos pueden publicar o suscribirse a mensajes.

El nodo suscriptor en el microcontrolador ESP32 se suscribe a este mismo tópico, lo que le permite integrarse en la red de ROS2 a través del agente que se está ejecutando.

Una vez que se ha establecido la comunicación entre Unity y el ESP32, el microcontrolador recibe los mensajes que el nodo publicador transmite a través de ROS2, lo que le permite actuar en consecuencia para controlar el sistema Pan-Tilt y la cámara.

Para garantizar la correcta comunicación, se ha creado un nodo suscriptor en Unity y un nodo publicador en el ESP32 (no representados en el diagrama). Ambos nodos están adscritos al tópico “freertos_header_log”, lo que permite el control de errores. También se ha acabado usando como controlador de la cámara y para que el ESP32 envíe la IP de la cámara a Unity y que este no tenga que saberla de antemano. Gracias a esto Unity no necesita a priori saber su IP sino que será guardada automáticamente cuando la cámara se conecte.

4.2. Interfaz de Unity

En esta sección se explicará la interfaz realizada en Unity para el control de los dispositivos usando Unity y ROS2.

4.2.1. Uso de NextMind SDK

El primer objetivo de este proyecto es explorar el uso del dispositivo NextMind en una interfaz gráfica de usuario bidimensional. Esto es importante, porque los ejemplos proporcionados por NextMind están principalmente basados en objetos 3D, y existían dudas sobre el correcto funcionamiento del API proporcionada por el fabricante en otros contextos como este.

A continuación se describe la incorporación a la aplicación de diferentes elementos del API de NextMind, que conceptualmente fueron descritos en el capítulo dedicado a la tecnología empleada.

NeuroManager

Para incorporar el NeuroManager, concepto explicado en el apartado 3.3.2, se emplea un “prefab”. Los prefabs son componentes prefabricados en Unity que permiten reutilizar configuraciones de objetos. Para incorporar el NeuroManager a la escena simplemente se arrastra el prefab correspondiente.

Es importante recalcar que solo puede existir un único NeuroManager activo en un instante dado. Para garantizar que su persistencia a lo largo de las diferentes escenas y evitar su duplicación, se añadió un script llamado DontDestroyOnLoad. Este script verifica si ya existe una instancia de NeuroManager y, en caso afirmativo, destruye cualquier instancia adicional. Si no existe ninguna instancia previa, mantiene la instancia actual en todas las escenas.

El código del script DontDestroyOnLoad es el que se puede ver en el apéndice B.1.

Adición del NeuroTag

Posteriormente, se añadió un botón que contiene un componente NeuroTag, cuyo concepto se explicó en el apartado 3.3.1. Gracias a la facilidad de uso de la SDK de Unity, al insertar el componente NeuroTag en un objeto, este permite llamar a una función cada vez que el dispositivo NextMind detecta que el usuario está enfocando su atención en el NeuroTag. Este comportamiento se realiza a través de un disparador (trigger).

Se añadió ese componente arrastrando un prefab de la parte de ejemplos llamada UITag, que es un NeuroTag diseñado para meterlo dentro de un canvas dentro del botón nombrado.

Para configurarlo, se necesita un "GameObject" con el script de la función que se desea ejecutar cuando el disparador se active. Una vez que se tiene esto, se arrastra el "GameObject" al campo del disparador en la configuración del NeuroTag. Finalmente, en el menú desplegable, se selecciona el método que se desea ejecutar de la clase del script.

Este proceso es válido no solo para los NeuroTags, sino también para otras interacciones como los eventos onClick y similares. Algo relacionado también es que cuando una variable es pública en Unity, puedes asignarle directamente un objeto de la escena (por ejemplo, si es de tipo Text o Image), siempre que este objeto se encuentre dentro de un GameObject.

Se optó por un botón como soporte del NeuroTag por varias razones. En primer lugar, si el NeuroTag falla o no responde, se puede interactuar con el elemento a través del ratón, ya que se le asigna el mismo evento onClick que en el onTrigger del NeuroTag.

Animación de retroalimentación

Durante la interacción del usuario con el sistema mediante BCI es esencial que este reciba información sobre el nivel de éxito de la detección. En caso contrario, no sabría si su nivel de atención está siendo adecuado o incluso le puede servir para mejorar su propia técnica de manejo.

Para proporcionar una retroalimentación más extensa que la predeterminada (el triángulo central del NeuroTag, véase figura 2.1), se incorporó la animación que utilizan los NeuroTags en el proceso de calibración. Esta animación emplea un script denominado Flash Controller, al que hay que añadir un "Flash Animator".

A continuación, en la jerarquía del botón y a la misma altura que el UITag, se creó un GameObject llamado Background. A este se le añadió un Animator con la animación HaloFeedback, presente en el ejemplo de calibración de NextMind, y también se le incorporó un script llamado NeuroTagFeedback.

Uso de Flash Controller

Para manejar la animación de retroalimentación, se incorporó el uso del script Flash Controller proporcionado por NextMind SDK. Este script es responsable de controlar la animación del Background, lo que incluye hacer que un círculo crezca y decrezca, reflejando así el estado del estímulo: crece cuando el estímulo está activo y decrece cuando no lo está.

Primero, se debe asignar el Flash Controller al GameObject que tenga el Animator para la animación de retroalimentación. Para hacer esto, se agrega el script de Flash Controller al GameObject deseado y se configura el Flash Animator en las propiedades del script.

Luego, se debe conectar este Flash Controller al componente NeuroTag. Para hacerlo, se asigna la instancia de Flash Controller al campo correspondiente en las propiedades del NeuroTag. Esto permite que el NeuroTag controle el Flash Controller, activando la animación de crecimiento cuando se detecta un estímulo y la de decrecimiento cuando no.

Este enfoque se basó en el análisis de la estructura del prefab del CircleNeuroTag, el cual incluye un GameObject llamado Background con dos elementos principales: la animación de estímulo y la retroalimentación triangular. Por consiguiente, fue esencial aprender cómo implementar el Background y cómo asignar correctamente el Animator al UITag cuando se agrega el script de Flash Controller.

Creación del Prefab NeuroButton

Después de configurar satisfactoriamente un botón con estas características, se decidió convertirlo en un prefab y se le denominó NeuroButton. De este modo, todos los botones con NeuroTags podrían adoptar las mismas características de manera uniforme y sencilla. Para convertir un GameObject en un prefab, el proceso es tan simple como crear un directorio para los prefabs y arrastrar el GameObject hacia ese directorio. El resultado de esta acción se puede ver en la figura 4.2.

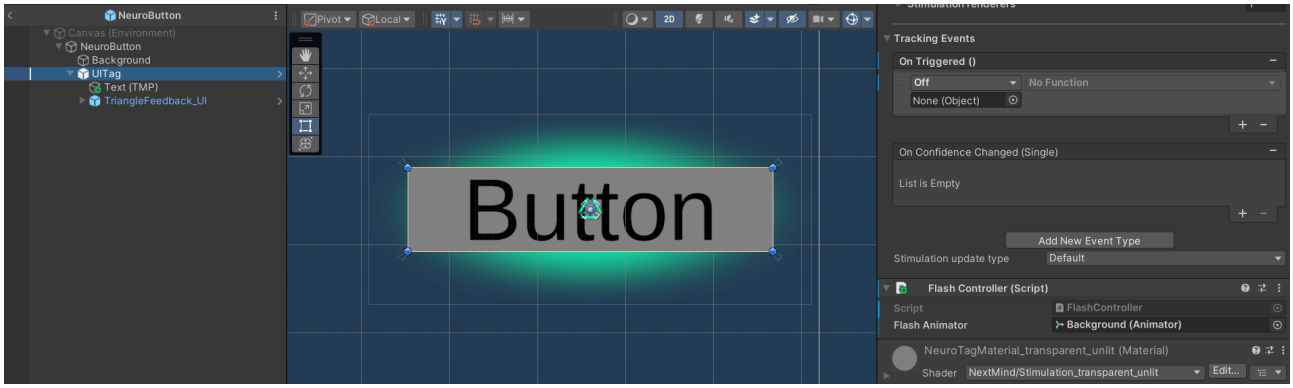


Figura 4.2: Prefab del NeuroButton

Incorporación de NeuroButtons en la escena

Una vez creado el prefab de los NeuroButtons, se incorporaron a la escena de Unity. Esto se logró simplemente arrastrando el prefab de cada NeuroButton dentro del canvas de la escena.

Cada NeuroButton se asoció con una dirección específica del pan-tilt: arriba, abajo, izquierda y derecha. Para lograr esto, se asignó a cada botón una función diferente que se activaría al enfocar el NeuroTag correspondiente. Es importante resaltar que, para proporcionar una interacción tanto por interfaz cerebral como por interfaz tradicional (mouse), se asignó la misma función tanto al evento de .ºnTrigger"del NeuroTag como al evento de .ºnClick"del botón. De este modo, se puede realizar la misma acción ya sea haciendo clic con el mouse o enfocando con la atención el NeuroTag.

Este tipo de configuración dual permite que el sistema sea operable tanto con el dispositivo NextMind como sin él, proporcionando así un grado de flexibilidad y adaptabilidad en función de las circunstancias del usuario o del ambiente de uso. Además, añade una capa de redundancia que puede resultar útil en caso de problemas técnicos con el dispositivo

NextMind o en situaciones en las que el usuario quiera interactuar con el sistema de una manera más tradicional.

Es importante señalar que se encontró un problema común cuando se trabajaba con NeuroTags dentro de un canvas en Unity: los NeuroTags pueden fallar en esta configuración. Este problema está documentado en la página de problemas conocidos y soluciones del SDK de NextMind[9].

Para solucionar este problema, se cambió el modo de renderizado (Render Mode) del canvas en el Inspector a "Screen - Space: Camera". Posteriormente, se arrastró la cámara principal (Main Camera) de la escena al campo Render Camera". Este ajuste garantiza que los NeuroTags se renderizan correctamente, permitiendo su detección y activación.

4.2.2. Integración de Ros2forUnity con NeuroButtons

Prefab de inicialización de ROS2

Teniendo la instalación realizada del asset Ros2ForUnity (véase el apéndice A.5). La integración con NeuroButtons se realizó de la siguiente manera. Se crea un prefab de inicialización de ROS2 que contiene dos scripts: un ROS2 Unity Component, siguiendo la guía oficial de Ros2ForUnity[1], y Ros2Start, un código que tiene como objetivo crear un nodo que genera un publicador para comunicarse con el Pan-Tilt y un suscriptor para recibir los mensajes del ESP32, sirviendo como canal de comunicación de log (ver apéndiceB.2) Como este es un script de inicialización, el publicador envía un mensaje a la cámara solicitando su IP si está conectada.

Es relevante destacar que Ros2Start implementa el patrón Singleton, asegurando que, una vez creada la instancia, no se destruye ni se recrea, garantizando un funcionamiento constante incluso durante los cambios de escena. Esta implementación es esencial para mantener la recepción de mensajes del suscriptor, que se discutirá en la sección 4.2.2. Igual que Ros2Start, el suscriptor también implementa el patrón Singleton para garantizar la continuidad y coherencia de la gestión de mensajes entrantes.

El patrón Singleton se implementa mediante el uso del método Awake(). Cuando Unity instancie la clase correspondiente, este método será el primero en ejecutarse. Si ya existe una instancia de la clase objetivo (significando que Instance no es nula), el objeto actual se destruirá para asegurar la unicidad de la instancia. En caso contrario, Instance se establecerá como el objeto actual, y gracias a DontDestroyOnLoad(gameObject), la persistencia de este objeto se garantiza a lo largo de los cambios de escena.

Es posible manejar el suscriptor directamente desde Ros2Start, sin embargo, para mantener la claridad y organización del código, se decidió separar esta funcionalidad en un archivo distinto.

Antes de describir el funcionamiento del código, es útil entender los desafíos que presenta la gestión de mensajes entrantes en aplicaciones en tiempo real, como Unity. En un entorno multihilo, múltiples procesos pueden intentar acceder y modificar los datos simultáneamente, potencialmente conduciendo a condiciones de carrera y otros problemas de concurrencia. Para manejar estos desafíos de manera segura y eficiente, se utilizó una cola segura para múltiples hilos (ConcurrentQueue). Esta estructura de datos permite encolar los mensajes a medida que llegan, para luego desencolarlos de manera segura para su procesamiento.

El código completo se encuentra en el apéndice B.2.

[1] Ros2ForUnityUsage: `https://github.com/RobotecAI/ros2-for-unity#usage`

Cuando se recibe un mensaje en el tópico "freertos_header_log", este se encola para su posterior procesamiento.

Finalmente, en el método Update(), los mensajes se desencolan y se invoca el evento OnMessageReceived para cada uno de ellos.

Esta implementación permite manejar los mensajes de manera segura y eficiente, mitigando los problemas potenciales de concurrencia y optimizando el uso de los recursos. Además, el uso de un patrón de delegado y evento permite una estructura de código flexible y sostenible, ya que los manejadores de eventos pueden ser fácilmente añadidos o eliminados según sea necesario.

Suscriptor

El componente Ros2SubscriberHandler (véase código en el apéndice B.3) es el responsable de gestionar y procesar los mensajes recibidos a través del sistema ROS2 en el entorno Unity. Este componente implementa el patrón Singleton, como se ha mencionado anteriormente, lo que garantiza que sólo habrá una única instancia de esta clase durante toda la ejecución del programa.

Cuando se inicializa el Ros2SubscriberHandler, el método Start() se invoca automáticamente para establecer los estados iniciales de las propiedades CameraEnabled, CameraIP y ESP32Enabled. Por defecto, asumimos que no hay ninguna cámara activa, no hay ninguna dirección IP de cámara disponible, y el ESP32 no está habilitado.

El método OnEnable() desempeña un papel esencial en el procesamiento de los mensajes recibidos. Este método se activa cuando el objeto Ros2SubscriberHandler se habilita en la escena de Unity. Dentro de este, se obtiene una referencia al componente Ros2Start y se suscribe al evento OnMessageReceived. Esta suscripción implica que cada vez que se reciba un nuevo mensaje a través de ROS2 (lo que dispara el evento OnMessageReceived), se invocará el método ProcessReceivedMessage.

El método ProcessReceivedMessage (código **??**) es el corazón de la clase Ros2SubscriberHandler. Este es el lugar donde se desglosan y analizan los mensajes entrantes. Cada vez que se recibe un mensaje, el contenido del campo Frame_id se divide en varias partes. Dependiendo de estas partes, se llevan a cabo diversas acciones, incluyendo la habilitación o deshabilitación de la cámara o del ESP32, y la actualización de la dirección IP de la cámara. El procesamiento de estos mensajes depende del formato y contenido esperado de los mismos en este sistema específico. Por ejemplo, si el mensaje indica que la estación objetivo se ha desconectado, se deshabilita la cámara y se borra la dirección IP de la misma.

Publicador

El publicador se gestiona en el script Ros2Publish (ver apéndince B.4). En él hay diferentes funciones publicas para su posterior asignación a los NeuroButtons.

Para ello, se ha creado un GameObject que incluye el script Ros2Publish con funciones para publicar, en función de lo que se necesita enviar al suscriptor del ESP32. Este script se diseñó con funciones públicas, lo que permite su asignación a eventos onClick u onTrigger.

En el código publicador que se encuentra en el apéndice B.4 y el código de la inicialización que se encuentra en el apéndice B.2, se puede observar que los tipos de mensajes del publicador y del suscriptor son del tipo Header. Header es un tipo de mensaje en

ROS2 que contiene metadatos sobre cuándo y cómo se generó la información. Este tipo de mensaje se seleccionó porque proporciona información de control adicional que resulta útil para la depuración.

El mensaje enviado siempre tiene el siguiente formato:

```
comando / datos
```

En este formato, el comando consta de 4 caracteres y los datos tienen un tamaño variable, con un máximo de 100 caracteres definido por el ESP32. Se optó por esta estructura para mejorar la eficiencia, ya que en lugar de tener muchos condicionales if/else, existen unos pocos grupos grandes de if (que corresponden a los comandos), y dentro de esos condicionales if, hay una serie de if/else dependiendo de los datos. Esta división reduce el número de condicionales if/else, mejorando el rendimiento en el peor y en el caso medio.

Para asignar las funciones públicas al onClick y al onTrigger del NeuroButton, hay que añadir a un GameObject el script de Ros2Publish. Este GameObject necesita estar separado del GameObject de inicialización y del suscriptor, porque en el momento que un solo script tenga la opción de 'Don't Destroy', el GameObject entero no se destruye y además si cambiamos de escena, al estar ya creado el Ros2Publish, los NeuroButtons y cualquier cosa que tenga onTrigger u onClick se queda en estado de 'missing'. Por lo tanto, debe construirse al mismo tiempo que el botón o cualquier otro elemento interactivo.

Una vez que se ha añadido a un GameObject que se destruya y construya según se cambia y se vuelve a la escena objetivo, se puede arrastrar ese GameObject al onClick y al onTrigger según corresponda del NeuroButton, y seleccionar la función correspondiente al acto que queremos publicar. Por ejemplo, si estamos en el UpNeuroButton, se publicará el mensaje:

```
"ACT_/Up"
```

4.2.3. Integración de la cámara

En esta parte del desarrollo se asume que la cámara incorporada al sistema debe funcionar como un servidor web capaz de realizar un streamming HTTP de imágenes, es uno de los requisitos que tiene que tener dicho dispositivo (véase la sección 3.9).

Por lo tanto, para integrar la visión de la cámara en Unity, es necesario comprender que la cámara no envía un vídeo continuo, sino que transmite imágenes MJPEG de manera sucesiva. Por tanto, será necesario procesar y separar estos paquetes.

Se procederá a detallar más adelante el proceso de separación y tratamiento de las imágenes.

Setup Inicial

Se ha creado un Prefab denominado VideoCamera que incluye en su estructura un BackgroundRawImage, un VideoRawImage y un QueryText. La intención de estos elementos es disponer de una imagen de fondo para los momentos en los que no se reciban imágenes de la cámara, un espacio para mostrar las imágenes en directo de la cámara y un área de texto para mostrar el estado de las peticiones GET a la API REST, permitiendo además la visualización de posibles errores en la transmisión.

Direcciones IP y Puertos

La cámara utiliza una dirección IP y un puerto específico para el streaming de imágenes, y la misma dirección IP con un puerto distinto para realizar peticiones GET y así poder cambiar su estado en la captura de códigos QR o en el HUD, mediante variables enviadas en la query.

El puerto 81 es utilizado para la visualización del streaming, mientras que el puerto 80 se utiliza para realizar las peticiones GET.

Interacción con la Cámara Mediante API REST

La petición GET para enviar un comando a la aplicación de la cámara se puede ver en el código 4.1.

Listing 4.1: Petición get para la cámara

```
http://IP:80/app?app=C&value=V
```

Donde IP es la dirección IP del servidor web y C y V funcionan de acuerdo a una tabla específica.

En la tabla 4.1 se pueden observar los distintos comandos (columna Ç") y sus respectivos valores (columna "V") utilizados para las peticiones GET a la cámara. Estos comandos y valores proporcionan una serie de funciones que controlan el estado y la funcionalidad de la cámara, como el reinicio del estado interno de la aplicación, la devolución del estado interno, el establecimiento de dicho estado, el control del streaming, el cambio de modo de detección de QRs, el reinicio del sistema completo y la solicitud de detección de QR. Además, algunos de estos comandos permiten conmutar entre modos de operación dependiendo del valor proporcionado.

C (comando)	V(valor)	Descripción
0	NA	Estado interno: Reinicia el estado interno de la aplicación a 0
1	NA	Estado interno: Devuelve el estado interno de la aplicación
2	s	Estado interno: Establece el estado interno de la aplicación en s
3	0	Control: detiene el streaming
3	1	Control: Cambia el modo de detección de QRs a por petición
4	NA	Reinicia el sistema completo tras una pausa
5	0	Solicita detección de QR y, si está en modo continuo, la conmuta al modo por petición

Tabla 4.1: Comandos y valores para peticiones GET a la cámara

Escena de Configuración

Para gestionar todas estas peticiones GET desde Unity, se ha creado una escena adicional denominada "Escena de Configuración"(véase la figura 4.3)

La funcionalidad de los botones se establece asignando funciones públicas a sus eventos onClick, tal como se detalló anteriormente.

El resultado final en funcionamiento se puede observar en la figura 4.6

Procesamiento de los Mensajes Recibidos

Siguiendo un orden lógico, primero se procesan los mensajes que llegan a través del suscriptor, mencionado previamente. En el método ProcessReceivedMessage, estos mensajes se procesan y se almacenan en atributos públicos.

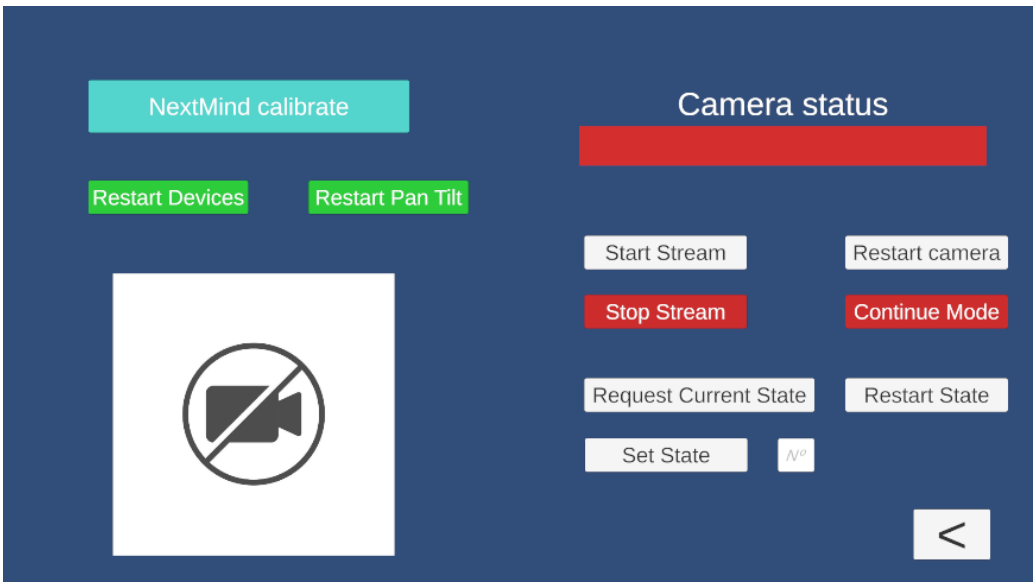


Figura 4.3: Escena de configuración

Clase CameraInfo

Para mantener una organización y legibilidad óptimas, se creó el script CameraInfo (véase el código en el apéndice B.5), que es una clase dedicada al estado de la cámara. Esta clase se encarga de recoger la dirección IP de la cámara, si está disponible, y de transformar esa IP en una URL para la captura del streaming o para preparar la URL para la API REST.

Los métodos `GetUrl` y `GetUrlApiRest` generan las URLs para el streaming de la cámara y para la API REST, respectivamente.

Clase CameraNetworkCommandController

Utilizando la clase CameraInfo, pudiendo obtener desde ella el GetUrlApiRest, se pueden hacer peticiones GET añadiéndole la query correspondiente. Eso es lo que trata de hacer este script, para lo que se incluyen métodos públicos con las ordenes descritas en la tabla 4.1. El código completo de la clase se puede ver en el apéndice B.6.

CameraNetworkCommandController: Método GetQR

Como ejemplo de las funciones de la clase CameraNetworkCommandController, el método GetQR es un método público esencial que se encarga de inicializar la solicitud de detección de códigos QR.

El método GetQR es un método público esencial en esta clase, que se encarga de inicializar la solicitud de detección de códigos QR. Primero, limpia cualquier mensaje previo en el campo responseText, que se utiliza para mostrar la respuesta de la cámara. Luego, verifica el estado de la cámara a través del método GetState del objeto cameraInfo. Si la cámara no está habilitada, se interrumpe el proceso, registrando un error en el registro de Unity y en el campo responseText. Si la cámara está habilitada, se construye un comando en formato de cadena de texto (/app?app=5&value=0") para solicitar la detección de un código QR. Posteriormente, se registra dicho comando en el registro de Unity para seguimiento y finalmente se inicia una corrutina llamada SendCommand que se encargará de enviar el comando a la cámara.

CameraNetworkCommandController: Rutina SendCommand

La rutina SendCommand lleva a cabo la solicitud GET de manera asíncrona, lo que significa que su ejecución se realiza en paralelo al hilo principal de Unity, permitiendo que otros procesos y animaciones sigan funcionando sin interrupciones. En esta rutina, se concatena el comando con la URL base de la API REST de la cámara, proporcionada por el método GetUrlApiRest del objeto cameraInfo, formando así la URL completa de la solicitud GET. Se configura un tiempo de espera de 60 segundos y se envía la petición. Una vez que se recibe la respuesta, se procesa y se muestra en el campo responseText, ofreciendo feedback sobre la operación realizada. En caso de producirse algún error durante la solicitud, se gestiona adecuadamente, proporcionando información detallada sobre la naturaleza del error.

Se modificó el tiempo de espera porque la conexión con la cámara es bastante lenta.

CameraNetworkCommandController: Método SetState

El método `SetState` tiene un comportamiento un poco diferente, ya que incluye un paso adicional antes de formar y enviar el comando. En este caso, se requiere un valor de entrada proporcionado por el usuario para cambiar el estado de la cámara. Este valor se introduce a través del campo setStateInputField. Se realiza un intento de convertir el texto introducido a un número entero utilizando int.TryParse(). Si la conversión es exitosa, es decir, el usuario ha ingresado un número entero válido, se forma el comando con la estructura /app?app=2&value="+ stateValue. Posteriormente, se registra este comando en el registro de Unity y se inicia la corutina SendCommand para enviar la petición a la cámara.

Si la conversión no es exitosa, lo que indica que el usuario ha introducido un valor no válido (no es un número entero), se registra este error en el registro de Unity y se muestra un mensaje de error en el campo responseText.

Por lo tanto, el método SetState extiende la funcionalidad de la clase CameraNetwork-CommandController, permitiendo cambiar el estado de la cámara a través de comandos de entrada del usuario, al mismo tiempo que gestiona y proporciona retroalimentación adecuada sobre cualquier error potencial en el proceso.

Inicialización del Streaming: ImageStreamUrl

Para inicializar el stream, hay un botón en la escena de configuración como se puede ver en la Figura 4.3. Ese botón tiene asociado en el evento onClick la función StartStream del script ImageStreamUrl (véase el código completo en el apéndice B.7) Este script contiene un software de código abierto diseñado para dividir imágenes MJPEG en segmentos manejables para que Unity pueda procesarlos y mostrarlos[2].

El método `StartStream()` es un método público que se utiliza para comenzar a recibir un flujo de imágenes de la cámara. Lo que se ha modificado de este software de código abierto es que primero verifica si la cámara está habilitada utilizando el objeto CameraInfo, y muestra un error si no lo está. A continuación, obtiene la URL de la cámara a través de CameraInfo y comienza a recibir el flujo.

Para hacer esto, StartStream() crea una instancia de Thread para ejecutar el método ReadMJPEGStreamWorker(). Este nuevo hilo se ejecuta en paralelo con el hilo principal

[2]Código abierto para manejar MJPEG en Unity: `https://gist.github.com/lightfromshadows/79029ca480393270009173abc7cad858`

del juego de Unity, lo que permite que la recepción y el procesamiento del stream de la cámara se realice sin bloquear la ejecución del resto del juego.

El hilo se inicia con dos argumentos: un identificador único y la URL del stream de la cámara. El identificador único se genera utilizando un número aleatorio para evitar conflictos entre múltiples instancias de ReadMJPEGStreamWorker().

Procesamiento y Segmentación de la Imagen: ImageStreamUrl

El método ReadMJPEGStreamWorker() se ejecuta en un hilo separado y se encarga de leer el stream de la cámara. Utiliza el protocolo HTTP GET para solicitar el stream de la cámara desde la URL proporcionada.

A medida que se recibe el stream de la cámara, ReadMJPEGStreamWorker() lo divide en "frames" individuales. Cada frame se representa como un array de bytes, y se identifica por una secuencia de bytes específica que marca el inicio y el final de un frame.

Estos frames se añaden a un búfer hasta que se haya recibido un frame completo. Cuando se recibe un frame completo, se envía a Unity para que lo procese y lo muestre. Esto se hace asignando el frame al atributo "nextFrame" en el método Update(). Cuando Unity está listo para procesar el siguiente frame, toma el valor de nextFrame, lo procesa y lo muestra, y luego limpia nextFrame.

Si se produce un error al leer el stream de la cámara, ReadMJPEGStreamWorker() intentará reiniciar la conexión hasta un número máximo de intentos.

Por último, el método SendFrame(byte[] bytes) toma un array de bytes que representa un "frame" completo y lo convierte en una Texture2D. Si los datos de la imagen son válidos, se aplica a un RenderTexture que puede ser mostrado en Unity. Si los datos de la imagen no son válidos, se muestra un mensaje de error.

Este Texture2D es un atributo público para poder asignarselo desde el inspector.

Visualización de Imágenes en Unity

Se dispone de un Texture2D y una imagen de fondo destinada a ser utilizada en el fondo del prefab VideoCamera, ya que son componentes necesarios del mismo. Esta relación se puede observar en la figura 4.7. Al atributo BackgroundRawImage se le asigna la imagen de fondo (que debe ser de tipo Sprite) y al atributo VideoRawImage se le asigna el Texture2D, estableciéndose así como la textura del RawImage respectivo.

Para que este Texture2D contenga imágenes, es necesario pasarlas a ImageStreamUrl, que dispone de un atributo público de tipo Texture2D preparado para este propósito. Véase la figura 4.4.

Uso del Patrón Singleton en ImageStreamUrl

Parece razonable que el ImageStreamUrl tenga que mantener su instancia y no poder ser borrada aunque cambiemos de escena para no perder la conexión con la cámara, así que le puse el patrón singleton para que solo existiera una sola instancia del mismo. Se le creó un GameObject único para que no afecte en la persistencia de otros GameObjects y que fuera legible y lógico semánticamente hablando. Ese GameObject se llama Streaming como se puede ver en la figura 4.4. Este GameObject se encuentra en la escena principal, donde se encuentran los NeuroButtons.

Gracias a esta configuración, la Textura2D que modifica ImageStreamUrl puede ser

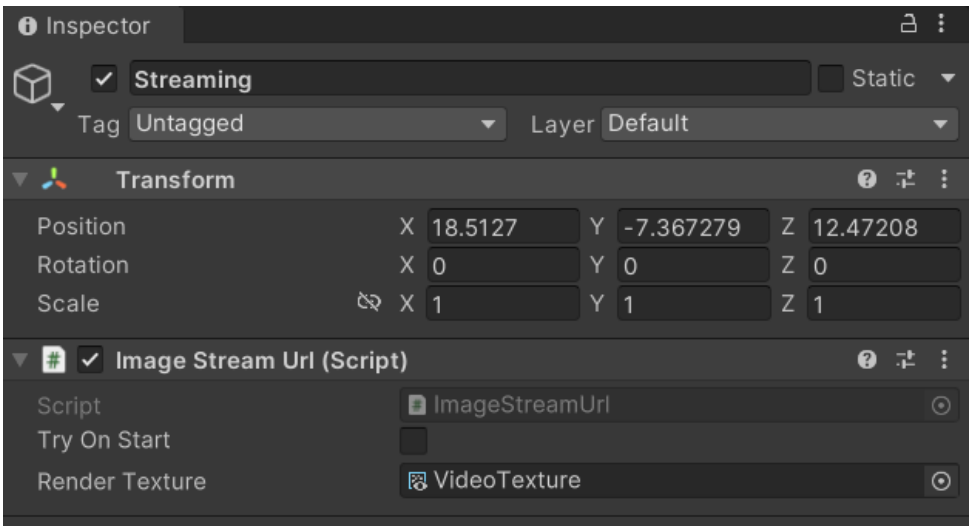


Figura 4.4: ImageStreamUrl en el GameObject Streaming

utilizada en todas las escenas con la información completamente actualizada. La implementación de este proceso se puede observar en la figura 4.5.

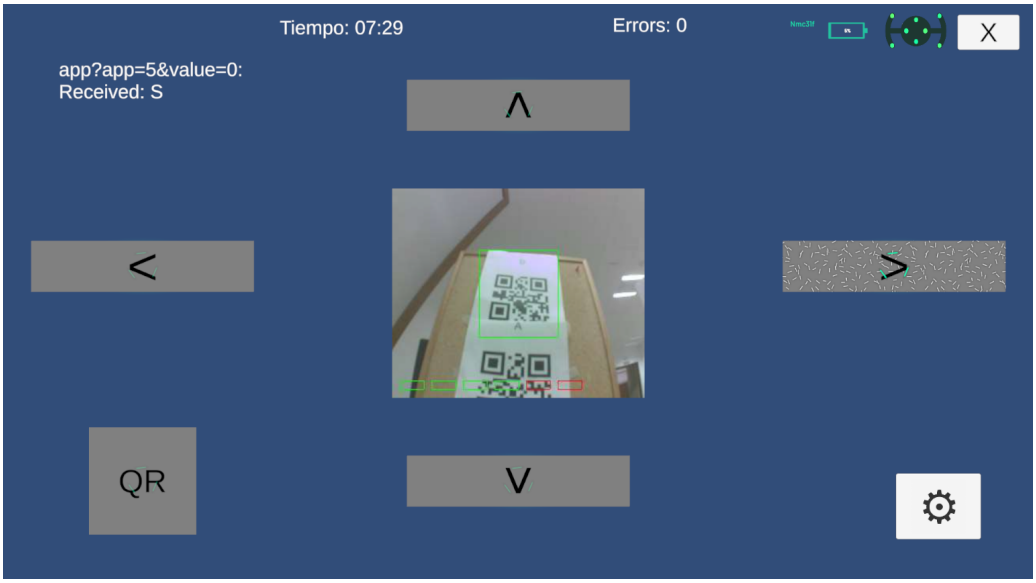


Figura 4.5: Resultado final de la escena de control del Pan-Tilt

Asignación de CameraInfo y ErrorText: ImageStreamUrlUI

Dado que el streaming es un objeto persistente, no se puede asignar directamente los valores de CameraInfo y ErrorText desde el inspector en función de la escena, ya que el streaming solo se encuentra en la escena principal y se perdería la referencia al objeto actual si lo asignamos de esa forma.

Para solucionar esto, se ha creado en ImageStreamUrl un método público llamado Initialize. Este método permite asignar los valores de CameraInfo y ErrorText, que son atributos privados, desde otro script que no sea persistente, es decir, que se pueda destruir y repetir en cada escena en la que se quiera mostrar el streaming. De esta forma, cada vez que nos movemos a una nueva escena, estos dos atributos se reasignan, y el objeto siempre sabe qué atributos escoger según la escena.

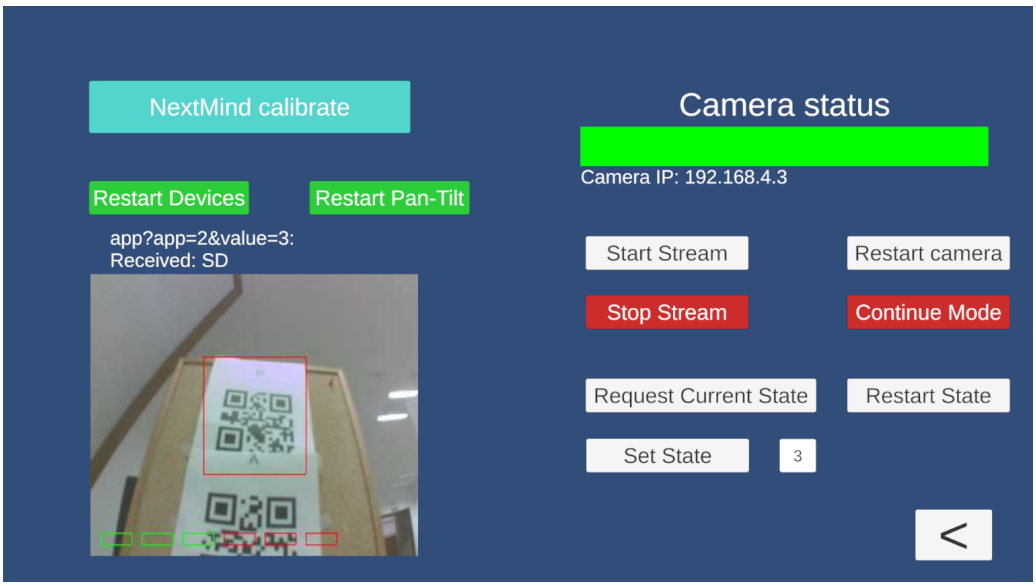


Figura 4.6: Resultado final de la escena de configuración del Pan-Tilt

Se ha creado un nuevo script llamado ImageStreamUrlUI para manejar esta funcionalidad. Este script tiene como atributos públicos una instancia de CameraInfo y un ErrorText, los cuales son asignados a través del inspector. Este script ImageStreamUrlUI se añade al prefab de VideoCamera, lo que resulta en una mejoría de la comodidad de su uso.

De esta forma, el script ImageStreamUrlUI toma la instancia de ImageStreamUrl y, mediante el uso del método público Initialize, asigna los elementos CameraInfo y ErrorText en función de la escena actual (véase figura 4.7)

Figura 4.7: ImageStreamUrlUI en el prefab VideoCamera

4.2.4. Botones de Control del Pan-Tilt

En la escena de configuración quedan dos botones, uno llamado Restart Devices y otro llamado Restart Pan-Tilt.

El botón Restart devices.ªprovecha una peculiaridad en la gestión del WiFi, que actualmente se encuentra fuera del control de micro-ROS. Por lo tanto, no puede ser desconectado internamente, lo que da lugar a un error. Este punto será detallado más adelante en el texto. Cuando ocurre dicho error, se desencadena un reinicio total de todos los dispositivos.

Para implementar esta funcionalidad, recurrimos a un procedimiento previamente

utilizado: empleamos el script Ros2Publish", instanciado en un GameObject en esta escena. Luego, arrastramos ese GameObject al evento onClick correspondiente y seleccionamos la función pública apropiada.

Para obtener una comprensión más detallada de este proceso, se puede referir a la sección 4.2.2, donde se explica su funcionamiento.

Se publica el siguiente mensaje:

```
CON_/DIS
```

En el Restart Pan-Tilt también es usando el mismo GameObject que tiene el Ros2Publish en esa escena. Esta vez se usa un comando

Se publica el siguiente mensaje:

```
ACT_/Reset
```

4.3. Proyecto ESP-IDF: Pan-Tilt

Además del repositorio de Unity, se maneja un repositorio aparte, ESP-IDF, que se emplea para la gestión del microcontrolador ESP32. Este repositorio se utiliza específicamente para controlar el mecanismo Pan-Tilt y la cámara, permitiendo un control eficiente y adaptativo de estos componentes.

4.3.1. Configuración de Comunicación: Personalización del Transporte

En este proyecto, la personalización del canal de comunicación se lleva a cabo mediante micro-ROS Transports, centrado especialmente en la UART (Universal Asynchronous Receiver-Transmitter) del ESP32. Este enfoque permite ajustar la comunicación a las necesidades específicas y características del hardware. Dado su papel crucial en la transmisión y recepción de datos, la UART es vital para la comunicación y control de otros dispositivos y sensores en el contexto del ESP32.

Para lograr esta personalización, se desarrolla un componente de transporte especializado. Este componente, basado en las funciones proporcionadas por micro-ROS Transports, gestiona la apertura, cierre, lectura y escritura de datos a través del puerto UART del ESP32.

El código para este componente se ha derivado de los ejemplos proporcionados en int32_publisher_custom_transport[3]. Sin embargo, se ha decidido convertirlo en un componente dedicado por su funcionalidad completa.

Gracias a la implementación de estas funciones, se logra establecer una comunicación personalizada utilizando la UART del ESP32 como medio de transporte. Este enfoque facilita la integración de este microcontrolador en sistemas que utilizan micro-ROS como middleware de comunicación.

El correcto despliegue de este componente requiere seguir las pautas establecidas en la sección A.3.3.

Este componente se encarga de la inicialización del ESP32 y configura su modo de transporte.

[3]int32_publisher_custom_transport: `https://github.com/micro-ROS/micro_ros_espidf_component/tree/iron/examples/int32_publisher_custom_transport`

4.3.2. Gestión del Pan-Tilt: Creación de un Componente Dedicado

Con el objetivo de controlar de manera eficaz y organizada los movimientos del sistema Pan-Tilt en el dispositivo, se desarrolló un componente específico. La estructura de este componente y la justificación de su creación se detallan en la sección A.3.3, mientras que el código está disponible en los apéndices B.8 y B.9.

El componente dedicado a la gestión del Pan-Tilt se diseñó para asegurar un control preciso de la modulación de ancho de pulso (PWM), esencial para manejar correctamente los movimientos del sistema. Este aspecto del diseño del componente se explica en profundidad en la sección 2.1.4.

El componente se compone de varias partes, que se listan a continuación:

- Constantes SERVO_MIN_DUTY, SERVO_MAX_DUTY, SERVO_MIDDLE_DUTY, SERVO_MAX_ANGLE y SERVO_MIN_ANGLE que definen los límites de los servomotores en términos de su ciclo de trabajo y los ángulos permitidos.

- Estructuras TimerState, ServoState y PanTiltState que almacenan el estado actual de los temporizadores y los servomotores.

- Funciones init_horizontal_servo, init_vertical_servo y init_pwm_timer, que inicializan los servomotores y el temporizador PWM. Incluyen además las funciones configure_timer y configure_servo, que configuran estos componentes utilizando la interfaz de control LEDC del ESP32.

- Funciones default_pan_tilt_init, pan_tilt_init y pan_tilt_deinit para manejar el estado de la plataforma de pan-tilt.

- Funciones set_horizontal_angle y set_vertical_angle, que permiten establecer los ángulos de los servomotores horizontales y verticales, respectivamente.

- Funciones update_timer_state,

  update_horizontal_servo_state y update_vertical_servo_state, que actualizan las estructuras de estado correspondientes.

- Finalmente, las funciones servo_deinit y timer_deinit, utilizadas para desactivar los servomotores y los temporizadores.

Tests del Componente Pan-Tilt

Para garantizar el correcto funcionamiento del componente Pan-Tilt, se han llevado a cabo una serie de pruebas. Estas pruebas ayudan a asegurar que los movimientos del sistema Pan-Tilt están bien controlados y que las funciones y estructuras que conforman el componente están trabajando según lo esperado.

El código de las pruebas se puede ver en el apéndice B.10.

Las pruebas realizadas son las siguientes:

- Prueba de límites máximos y mínimos de duty cycle: (TEST_CASE("Test max duty and min duty", "[pan_tilt_controller]")) Esta prueba verifica que los límites de duty cycle de los servomotores se establecen correctamente.

- Prueba de inicialización del Pan-Tilt

  (TEST_CASE("Test pan-tilt initializated", "[pan_tilt_controller]")):

  esta prueba confirma que el sistema Pan-Tilt se inicializa correctamente.

- Pruebas de ángulo horizontal y vertical

  (TEST_CASE("Test initial horizontal angle", "[pan_tilt_controller]")

  y TEST_CASE("Test initial vertical angle", "[pan_tilt_controller]")): estas pruebas verifican que los ángulos horizontales y verticales de los servomotores se establecen y se actualizan correctamente.

- Prueba de incremento de movimiento (TEST_CASE("Test increment movement", "[pan_tilt_controller]")): esta prueba verifica que el sistema Pan-Tilt puede cambiar su ángulo vertical con precisión, lo cual es importante para garantizar un movimiento suave y controlado.

Las pruebas se realizan utilizando la biblioteca Unity, que proporciona un marco para la creación de pruebas unitarias en C. Todas las pruebas pasaron con éxito, lo que indica que el componente Pan-Tilt está funcionando correctamente y puede controlar los movimientos del sistema con precisión.

4.3.3. Gestión micro-ROS

Para la gestión de micro-ROS dentro del ESP32 se ha creado una tarea específica (micro_ros_task).

La razón para dedicar una tarea a la gestión de micro-ROS es que este middleware necesita tiempo de procesador para gestionar las comunicaciones entre los diferentes nodos. Al dedicar una tarea a esta función, se garantiza que micro-ROS tiene el tiempo necesario para gestionar estas comunicaciones sin interferir con otras tareas que el microcontrolador pueda estar realizando.

Suscriptor

La función de callback del suscriptor es subscription_callback. Esta función es bastante grande porque tiene que manejar una variedad de mensajes diferentes que podrían ser enviados al nodo del suscriptor.

La función de callback comienza por extraer los datos del mensaje recibido. Si el mensaje es demasiado largo, se imprime un error y la función se detiene.

A continuación, la función de callback extrae el comando y los datos del mensaje y los publica en otro tema.

Finalmente, la función de callback ejecuta diferentes acciones dependiendo del comando y los datos recibidos. Estas acciones pueden incluir inicializar la conexión WiFi como punto de acceso, desconectar la conexión WiFi, establecer los ángulos de los servos y más.

Publicador

En este proyecto, el publicador se inicializa en la función micro_ros_task, mediante el uso de la función rclc_publisher_init_default. El publicador se configura para enviar mensajes del tipo std_msgs__msg__Header al tema /freertos_header_log".

Posteriormente, el publicador se emplea en distintas secciones del código para la transmisión de diversos mensajes. Específicamente, se utiliza en la función de devolución de llamada del suscriptor para enviar comandos y datos recibidos como registros.

Además, el publicador juega un papel importante en la gestión de conexiones WiFi. Por ejemplo, la función wifi_event_handler_sap maneja diferentes eventos que pueden suceder en la red Soft Access Point (SAP). En el evento de que una estación se conecte y su MAC coincida con la MAC objetivo (la MAC de la cámara) el publicador envía un mensaje con la dirección IP de la estación recién conectada.

En caso de desconexión de una estación, el publicador informa si la estación desconectada es la estación objetivo. Si se asigna una IP a una estación, el publicador notifica la dirección IP asignada, especificando si se ha asignado a la estación objetivo.

Finalmente, durante el proceso de inicialización de la red SAP, la función wifi_init_softap utiliza el publicador para informar de los diversos estados del proceso, incluyendo el inicio, cualquier error durante la inicialización y la finalización del proceso.

4.3.4. Manejo de la Conexión WiFi

El ESP32 se configura como un Soft Access Point (SAP) (consulte la sección 4.1), creando una red WiFi a la que se pueden conectar otros dispositivos. La finalidad de esta red es facilitar la comunicación entre la cámara y los demás nodos del sistema. El ESP32 publica información sobre los eventos de la red a través de un tema en micro-ROS, como la conexión y desconexión de estaciones, así como la asignación de direcciones IP a las mismas.

El manejador de eventos WiFi, la función `wifi_event_handler_sap`, se invoca cuando ocurren eventos específicos en la conexión WiFi. Dependiendo del tipo de evento, la función realiza acciones distintas:

- Cuando una estación se conecta al SAP, se desencadena el evento `WIFI_EVENT_AP_STACONNECTED`. Aquí, se obtiene y publica la dirección MAC de la estación conectada.
- El evento `WIFI_EVENT_AP_STADISCONNECTED` se activa cuando una estación se desconecta del SAP. En este escenario, se publica un mensaje informando de la desconexión. Si la estación desconectada corresponde a la `target_mac`, se envía un mensaje adicional notificando que la estación objetivo se ha desconectado, y se marca la variable `sap_ip_target_assigned` como false.
- Al asignarse una dirección IP a una estación conectada al SAP, se genera el evento `IP_EVENT_AP_STAIPASSIGNED`. Si la estación a la que se le ha asignado la IP coincide con la `target_mac`, se guarda la IP asignada en la variable `sap_ip_target` y se configura `sap_ip_target_assigned` como true. Luego, se publica un mensaje notificando que la estación objetivo ha obtenido una dirección IP.

Las funciones `get_camera_ip` y `get_target_ip` se utilizan para recuperar la dirección IP de la estación objetivo y verificar si a dicha estación se le ha asignado una IP, respectivamente.

Finalmente, la función `wifi_init_softap` se encarga de la inicialización del SAP. Configura el SAP con los parámetros establecidos en la estructura `wifi_config_t`, registra el controlador de eventos `wifi_event_handler_sap` para los eventos WiFi y el evento

IP_EVENT_AP_STAIPASSIGNED. Tras la configuración del SAP, se inicia el WiFi y se envía un mensaje que indica la finalización de la inicialización del SAP.

Control de la Cámara a través de WiFi

El sistema de control de la cámara debe estar diseñado para establecer una conexión automática a la red WiFi generada por el ESP32. La cámara tiene preconfigurada lacredenciales de la red WiFi del ESP32, lo que permite que se conecte de manera automática en cuanto se detecta la presencia de la red WiFi. Este enfoque asegura una conexión estable y confiable entre la cámara y el ESP32, facilitando el control y la comunicación sin necesidad de configuraciones adicionales.

4.4. Puesta en marcha del sistema

En esta sección se describirán las acciones necesarias para poner en marcha el sistema en un entorno Windows, haciendo uso del subsistema Windows para ejecutar un agente Microros conectado al dispositivo por medio de una conexión USB-Serial.

4.4.1. Actuaciones en Powershell de Windows

Para poner en marcha el proyecto primero hay que montar el setup de ROS2 en Powershell de la siguiente forma:

```
C:\dev\ros2\_humble\local\_setup.ps1
```

Y abrir el proyecto de Unity en la misma Powershell

```
unity.exe -projectPath "D:\Unity\Unity Projects\Thesis-Unity-Pan-Tilt\"
```

Luego usar el programa usbipd para atar los puertos correspondientes a WSL2 (véase sección A.3.2) En este paso hay que asegurarse de que ya estamos conectados al WiFi del ESP32 (véase la figura 4.8)

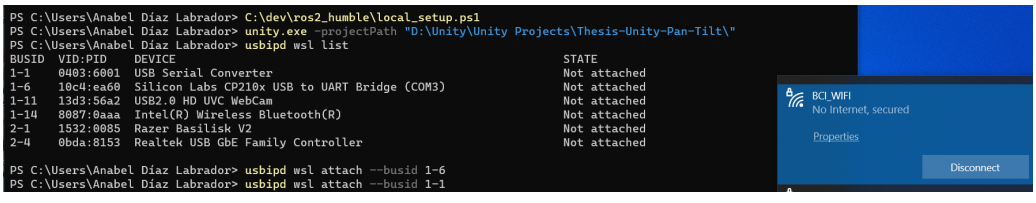


Figura 4.8: Preparación del proyecto en Windows

El comando para atar el puerto a WSL2 con usbipd es:

```
usbipd wsl attach --busid BUSID
```

Donde BUSID es el busid correspondiente.

4.4.2. Actuaciones en Ubuntu sobre WSL2

Agente de micro-ROS

En una terminal de WSL2 hay que dar permisos a los puertos, por ejemplo si hemos atado el microcontrolador y la cámara (esta última solo en caso de que sea necesario su monitorización) sería así:

```
sudo chmod 777 /dev/ttyUSB0
sudo chmod 777 /dev/ttyUSB1
```

Se realiza ahora un source install/setup.bash del repositorio del agente de microros (véase el apéndice A.4), para hacer que esta aplicación esté disponible para ROS2.

Finalmente, ejecutamos el agente de micro-ROS poniendo el puerto correspondiente:

```
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0
```

Monitorización de la cámara

En otra terminal de WSL2 se necesita establecer como entorno de desarrollo “esp-idf v4” haciendo un source al script de exportación de esta versiónde la toolchain (véase A.6.1)

```
get_idf4
```

Primer es necesario compilar el firmware e introducirlo en el microcontrolador

```
idf.py build
idf.py -p /dev/ttyUSB1 flash
```

Luego se puede monitorear de la siguiente manera:

```
idf.py -p /dev/ttyUSB1 monitor
```

4.5. Funcionamiento del prototipo

El sistema presenta una interfaz intuitiva y directa. Su funcionamiento sigue un esquema de “apunta y dispara”, donde el usuario debe centrar el código QR en un cuadrado designado como punto de mira en la interfaz de la cámara. Seis rectángulos ubicados en la parte inferior del HUD de la cámara se iluminan en verde al detectar códigos QR.

Incorpora cuatro NeuroButtons (véase sección 4.2.1) que controlan el movimiento del sistema Pan-Tilt en cuatro direcciones: arriba, abajo, izquierda y derecha. El HUD de la cámara, ubicado centralmente, alberga un cuadrado que permanece constantemente en rojo, cambiando a verde cuando se detecta exitosamente un código QR. Se puede ver prototipo entero conectado y montado en la figura 4.9.

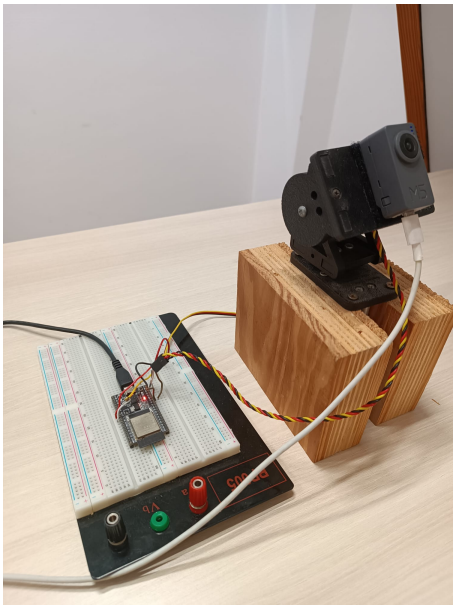


Figura 4.9: Prototipo montado y conectado

Capítulo 5

Analisis de la experimentación y resultados obtenidos

En este capítulo se abordará la evaluación del sistema desde dos perspectivas distintas.

En primer lugar, se realizará una evaluación del sistema Pan-Tilt, centrando nuestra atención en la experiencia del usuario. Se analizarán diferentes aspectos, como la facilidad de uso, la interacción y la respuesta del sistema a los comandos del usuario.

En segundo lugar, se enfoca en la flexibilidad de la interfaz, evaluando su capacidad para adaptarse a otro sistema. Específicamente, se estudiará su adaptación a un robot móvil, que incorporará un nodo de micro-ROS inalámbrico. Este nodo operará a través del protocolo UDP.

5.1. Detalles de las pruebas realizadas

Siguiendo la metodología empírica descrita en la sección 1.4, este capítulo está dedicado a proporcionar un panorama más detallado de las pruebas realizadas con los participantes. Las pruebas se llevaron a cabo en la Escuela Superior de Ingeniería y Tecnología (ESIT) de la Universidad de La Laguna, donde 16 participantes experimentaron con la aplicación de control Pan-Tilt en diversas condiciones de iluminación. Tras un periodo de calibración del BCI, los participantes afrontaron una prueba de diez minutos para recorrer un circuito de códigos QR (véase figura 5.1). Se recopilaron datos sobre fallas del BCI, tiempos de calibración y de recorrido, los cuales han resultado esenciales para el análisis en profundidad presentado en las siguientes secciones.

Hay que tener en cuenta que la muestra recogida es bastante pequeña.

La prueba implica la navegación a través de un circuito diseñado con códigos QR, utilizando un sistema Pan-Tilt equipado con una cámara. Los participantes debían completar el recorrido en un tiempo no superior a 10 minutos. La prueba se ejecuta dentro de una aplicación Unity, la cual incorpora una serie de NeuroTags para la manipulación del movimiento del sistema Pan-Tilt. El funcionamiento del prototipo se detalla en la sección 4.5 y se visualiza en la figura 4.5.

5.2. Presentación de los datos obtenidos

Los datos obtenidos de los 16 participantes se presentan en la tabla 5.1. Esta incluye variables como el tipo y la longitud del pelo, la nota de calibración, la luminosidad durante el uso del dispositivo y el tiempo total de uso de la interfaz cerebro-computadora (BCI).

Figura 5.1: Circuito de QR

Figura 5.2: Circuito de QR en el exterior

La longitud y el tipo de pelo se recogieron debido a su potencial impacto en la capacidad del dispositivo NextMind de leer correctamente las señales cerebrales. Este aspecto es esencial para comprender las limitaciones de la interfaz en poblaciones con diferentes tipos de cabello y longitud.

Se ha dado especial importancia a la luminosidad, considerando que el estudio está orientado hacia el uso del dispositivo en exteriores. La luz ambiental podría afectar la precisión de la interfaz, posiblemente debido a los reflejos en la superficie del dispositivo o la alteración de las señales que este intenta leer.

La nota de calibración, que es una medida de cuán precisamente se ajustó el dispositivo

Figura 5.3: Prototipo Unity en el exterior

a las necesidades individuales del usuario, se ha correlacionado con el número de errores cometidos durante el uso y el tiempo total de uso del BCI (Tiempo BCI). Esta relación es clave para evaluar el impacto de una calibración adecuada en el rendimiento del dispositivo.

El tiempo total de uso del BCI (Tiempo BCI) y el número de errores que el usuario cometió mientras utilizaba la interfaz se registraron para determinar la efectividad y la facilidad de uso del dispositivo.

Por último, se consideró si los usuarios pudieron completar las pruebas asignadas en el tiempo establecido. Si no se podía realizar la prueba, se les asignaba el tiempo máximo posible de 10 minutos, proporcionando una medida de la duración máxima de la prueba en las circunstancias más desfavorables.

Longitud del pelo	Tipo de pelo	Nota de Calibración	Luminosidad	Tiempo BCI	Número de errores	¿Pudo realizar la prueba?
Mediano/Corto	Rizado	2	Baja	05:50	4	Sí
Rapado/Calvo	Lacio	4	Media	06:32	5	Sí
Mediano/Corto	Rizado	3	Media	05:05	4	Sí
Rapado/Calvo	Lacio	1	Baja	10:00	-	No
Largo	Rizado	1	Baja	10:00	-	No
Rapado/Calvo	Lacio	5	Baja	03:51	0	Sí
Largo	Lacio	3	Baja	05:26	3	Sí
Mediano/Corto	Ondulado	2	Media	10:00	-	No
Rapado/Calvo	Lacio	3	Media	07:45	2	Sí
Rapado/Calvo	Lacio	2	Media	04:11	0	Sí
Largo	Rizado	4	Media	03:50	0	Sí
Mediano/Corto	Ondulado	5	Baja	02:08	0	Sí
Largo	Lacio	3	Media	03:39	1	Sí
Mediano/Corto	Lacio	3	Media	04:14	2	Sí
Mediano/Corto	Lacio	3	Media	04:58	5	Sí
Largo	Rizado	2	Media	05:19	2	Sí

Tabla 5.1: Datos obtenidos en las pruebas para los 16 participantes

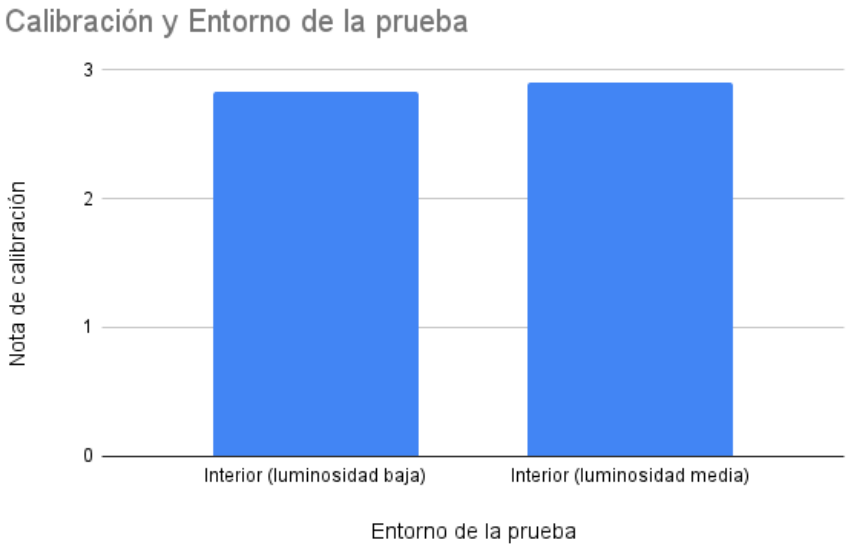


Figura 5.4: Gráfica con la calibración y el entorno de la prueba

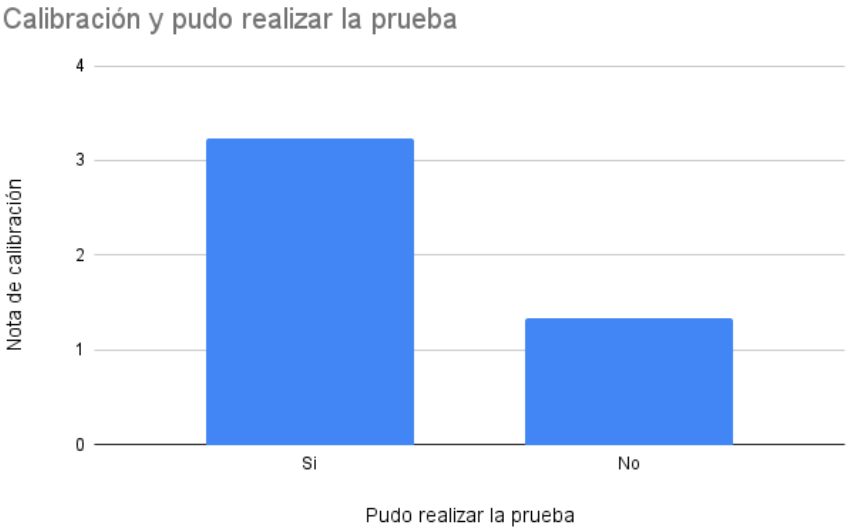


Figura 5.5: Gráfica con la calibración y si pudo realizar la prueba

5.3. Análisis de los datos

A pesar de que la muestra es relativamente pequeña, se pueden extraer conclusiones significativas.

Primero, se observó que el cabello no parece influir en la calibración ni en la destreza general del BCI.

En los entornos interiores, la variación entre las calibraciones es mínima, como se puede ver en la Figura 5.4.

Por otro lado, se realizaron pruebas en exteriores bajo las peores condiciones, en pleno sol a mediodía. Los resultados fueron sorprendentes. Si la persona es capaz de ver el estímulo visual (NeuroTags), el NextMind es capaz de reconocerlo sin demasiada dificultad. Se probó con el tutor, quien suele obtener un 5 en la calibración: obtuvo un 4 en la prueba y presentó tiempos de respuesta similares a los obtenidos en interiores.

Figura 5.6: Gráfica con el tiempo transcurrido y la calibración

Figura 5.7: Gráfica con el tiempo transcurrido y si tenía experiencia previa con el BCI

El tipo de pantalla utilizada al aire libre es un factor que influye considerablemente en los resultados. Se observaron diferencias notables al utilizar pantallas más oscuras, como la mostrada en la figura 5.2, en comparación con pantallas de brillo alto5.3. Entre las dos pantallas LCD evaluadas, la que presentaba mayor brillo incorporaba tecnología IPS, a diferencia de la otra. Además, es relevante mencionar que el brillo de las pantallas LCD puede verse afectado por la degradación con el paso del tiempo, teniendo en cuenta que ambas pantallas son LCD, una es nueva mientras que la otra tiene ya 4 años de uso, lo cual podría explicar algunas de las diferencias observadas.

La pantalla utilizada finalmente para la prueba en exterior fue un monitor SAMSUNG F24T350FHU dada por el profesorado.

Además, se encontró que cuanta más calibración se realice, más probabilidades hay de completar la prueba con éxito. Incluso con una calificación de 2, se consideró posible realizar la prueba. Solo en 1 de los 16 casos, la prueba no se pudo realizar con una calibración de 2. Ver Figura 5.5.

Se observa una correlación aparente entre el tiempo de realización de la prueba y la

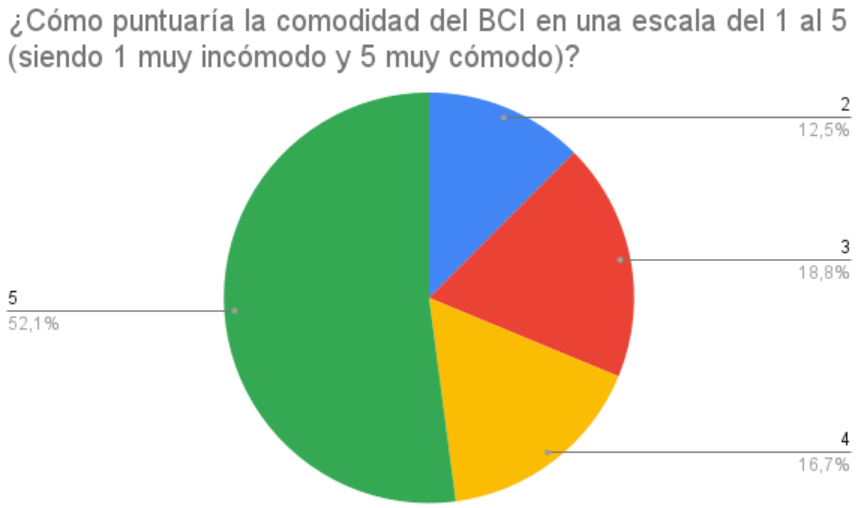


Figura 5.8: Comodidad del BCI

calificación de calibración: a mayor calificación, menor tiempo necesario para completar la prueba (Figura 5.6).

Es relevante señalar que si los participantes tienen experiencia previa con el uso de BCI, tienden a completar la prueba en menos tiempo (Figura 5.7).

Incluso durante la prueba, se observó una mejora continua en los participantes, independientemente de si su calificación de calibración inicial no era de 4 o 5.

También se consultó a los usuarios sobre su percepción de la comodidad del BCI. Como se aprecia en el gráfico 5.8, la mayoría de los usuarios, más del 50 %, calificaron su experiencia como "Muy cómoda", asignándole un 5, el puntaje máximo. En contraste, solo un pequeño grupo, el 12,5 %, le dio un 2, señalando una experiencia de comodidad reducida. Es importante resaltar que ningún usuario otorgó la puntuación mínima de 1, que representaría una experiencia "Muy incómoda".

5.4. Adaptación a un nuevo sistema

La aplicación de Unity ha sido eficientemente adaptada a un nuevo dispositivo: un robot móvil. Este proceso ha puesto de relieve la notable adaptabilidad de la aplicación. Esta capacidad quedó patente cuando logramos recrear una nueva interfaz desde cero, reutilizando los scripts y prefabs existentes, en un plazo de menos de dos horas.

El entorno donde se llevó a cabo esta prueba se puede observar en la figura 5.9.

5.5. Conclusiones

Este estudio revela la efectividad y versatilidad de NextMind en diversas condiciones y contextos. El dispositivo NextMind demuestra ser un BCI accesible para una amplia variedad de usuarios, sin ser afectado por factores como la presencia de cabello. Muestra una robustez considerable, funcionando de manera efectiva tanto en ambientes interiores como exteriores. Adicionalmente, el rendimiento del dispositivo parece ser casi inmune a las variaciones de luz ambiental, lo cual es un hallazgo que puede orientar el desarrollo futuro de dispositivos BCI. Además, la eficiencia de NextMind tiende a mejorar con una

Figura 5.9: Entorno de la prueba del robot

calibración de mayor precisión y con la experiencia previa con dispositivos BCI, lo que destaca la importancia de familiarizarse con esta tecnología. En términos de comodidad, más del 50 % de los usuarios encontraron que NextMind era "Muy cómodo", lo que sugiere una alta aceptabilidad y potencial para su uso continuado.

Para concluir, la interfaz ha demostrado una notable capacidad para adaptarse a diversos contextos y sistemas. Esta flexibilidad refuerza su utilidad, al permitir que se aplique de manera efectiva en una amplia gama de situaciones, haciendo que la interfaz sea una herramienta versátil y eficiente en el campo de la robótica.

Capítulo 6

Problemas y dificultades encontradas

6.1. Compatibilidad de versiones

Dentro del complejo escenario de trabajo que implica la utilización de múltiples herramientas, librerías y tecnologías que no han sido diseñadas a la medida para su integración, emergen frecuentes problemas de incompatibilidad de versiones. Este entorno puede producir desafíos significativos, particularmente cuando las versiones no son compatibles entre sí, generando problemas y retrasos en el desarrollo. Uno de los casos más destacados de esta situación fue el relacionado con UnityRoboticsHub, que se detalla al final de este capítulo.

6.2. Chocolatey

Durante el intento de instalación de ROS2 para Windows surgieron algunos obstáculos Tras numerosas dificultades, se identificó que la raíz del problema se encontraba en Chocolatey, la herramienta de gestión de paquetes utilizada. La dificultad surgió debido a que Chocolatey no desinstalaba correctamente los paquetes. Este inconveniente requirió una solución manual, que implicó la eliminación de los paquetes directamente en el editor de registros de Windows. Esta tarea, aunque solucionó el problema, destacó la necesidad de herramientas de gestión de paquetes más eficientes y confiables.

6.3. Configuración del WiFi: usbipd

Uno de los desafíos surgió en la configuración del WiFi utilizando usbipd. La dificultad principal se centró en la falta de conciencia de la necesidad de una red estable. El WiFi del ESP32 solo era visible mientras el agente de microros estaba en funcionamiento. Además, se observó que al conectarse al WiFi del ESP32, este se desconectaba y el WiFi se apagaba nuevamente. La causa de este problema radicaba en que la gestión del WiFi se realizaba dentro de la tarea de microros en el microcontrolador. Al cambiar de red, usbipd desconectaba los puertos, provocando problemas en Linux con la existencia de puertos simulados. Esto ocasionaba que el agente de microros dejara de funcionar y, por consiguiente, la tarea de microros dejara de ejecutarse, terminando con la gestión del WiFi.

La solución a este problema consistió en colocar la gestión del WiFi fuera de la tarea de microros. Sin embargo, esto conllevó la desventaja de que el WiFi ya no podía ser

controlado desde ROS2.

6.4. Inclusión de imágenes de la cámara en Unity

El proceso de introducir la imagen de la cámara en Unity también presentó numerosos problemas. La fuente de la dificultad residía en que se estaba transmitiendo un flujo constante de paquetes MJPEG, que son imágenes. Inicialmente, se intentó utilizar assets de videos y assets de gestión de HTTP en Unity, pero estos enfoques no dieron resultados satisfactorios. La única solución viable consistió en dividir los paquetes y procesarlos para que Unity pudiera visualizarlos a medida que llegaban. Gracias al código abierto que se encontró, fue posible implementar esta parte del proyecto.

6.5. Trabajo de cambios

En términos generales, este proyecto ha estado caracterizado por la necesidad de realizar muchos cambios y adaptaciones en función de las circunstancias que surgían. Aunque este proceso ha sido desafiante, al final se han podido superar todas las adversidades, demostrando la capacidad de adaptación y resiliencia frente a las dificultades.

6.6. Imposibilidad de usar ciertos dispositivos

Inicialmente, se contempló la posibilidad de utilizar una silla de ruedas, lo que llevó a adaptar una interfaz específica para este propósito. Sin embargo, este plan tuvo que ser desechado debido a la falta de disponibilidad de sillas de ruedas robotizadas funcionales en la universidad. Posteriormente, se intentó utilizar un robot, pero este enfoque también resultó fallido debido a problemas en su sistema electrónico.

6.7. La persistencia de datos en Unity

El manejo de la persistencia de datos en Unity presentó una serie de desafíos significativos. Uno de los principales problemas radicó en entender cómo funcionaban los cambios de escena dentro de esta plataforma. En ocasiones, estos cambios resultaban en fallos inesperados y difíciles de rastrear. Adicionalmente, hubo dificultades al tratar de mantener ciertos elementos persistentes, ya que en algunos casos se mantenían más elementos de los planificados. Estos problemas se originaron por una comprensión insuficiente de cómo funciona la persistencia en Unity y cómo se debe manejar un script dentro de un GameObject de forma correcta. A pesar de la cantidad considerable de tiempo perdido en estos asuntos, finalmente se logró obtener un resultado satisfactorio.

6.8. Experiencia con Unity-Robotics-Hub

Unity-Robotics-Hub fue la primera librería que se evaluó para ser utilizada para integrar ROS2 en Unity. Esta librería mostraba signos de robustez y de amplia utilización, lo cual indicaba que podía ser una opción viable.

No obstante, se encontraron serios problemas, ya que la librería presentaba fallos desde las versiones tempranas del código. Existe la posibilidad de que estos problemas se deban a algún tipo de incompatibilidad con la SDK de NextMind.

Se intentaron múltiples soluciones, como la modificación completa del código y el análisis de los registros de Unity. También se probaron las implementaciones de Next-Mind SDK y UnityRoboticsHub de manera independiente, observando que funcionaban correctamente por separado, pero no conjuntamente.

Es una lastima que Unity-Robotics-Hub no haya funcionado de manera óptima, ya que, a pesar de funcionar a través de un intermediario, el código resultante era más legible y la carga para Unity no era tan elevada como con la implementación actual, que debe manejar ROS2 en su totalidad, generando nodos dentro de Unity.

Capítulo 7

Conclusiones y líneas futuras

7.1. Conclusiones

En este apartado se presentan las conclusiones más importantes de este trabajo.

En primer lugar, he de destacar que se han cumplido todos los objetivos aunque no todo haya salido como se esperaba.

7.1.1. Desarrollo del prototipo

Como se destaca en la capítulo 6, la necesidad de aprender ROS desde cero representó un compromiso significativo de tiempo y esfuerzo autónomo para la comprensión y asimilación de la información. Adicionalmente, no había tenido experiencia previa programando en Unity, lo que también implicó un proceso de aprendizaje desde cero. En relación a la programación específica en el ESP32, aunque era un terreno desconocido, me beneficié de los conocimientos previos adquiridos en una asignatura del grado denominada Sistemas Empotrados, que proporcionó una base sólida para la programación en general en microcontroladores. Esto resultó especialmente útil durante la programación del pan-tilt desde el microcontrolador con respecto a la PWM, una tarea que, si bien requirió cierta investigación, resultó más sencilla gracias a las habilidades adquiridas en dicha asignatura. De todos los aspectos del proyecto, lo que presentó menos complicaciones fue la operación de los Neurotags y el sistema en sí de NextMind.

En cuanto a mi percepción personal sobre las herramientas utilizadas, ROS me pareció fascinante y es una herramienta que me gustaría seguir utilizando en el futuro. Mi experiencia con Unity, por otro lado, no fue tan positiva debido a su sistema de registro y gestión de errores, que considero complicado en cuanto a la localización de fallos. Respecto a ESP-IDF, me ha proporcionado una experiencia satisfactoria, percibiéndola como una estructura más modular que un proyecto de micro-ROS para la programación en microcontroladores Freertos.

7.1.2. Interpretación de los datos y conclusiones obtenidas

Los resultados del estudio apuntan a la eficacia y adaptabilidad de NextMind en distintos entornos y situaciones. Como BCI, se muestra accesible y resistente, sin verse influido por aspectos como la presencia de cabello. Su rendimiento eficaz, tanto en interiores como exteriores, y su resistencia a las variaciones de luz ambiental, abren nuevas posibilidades para futuros avances en dispositivos BCI. Se observó que una calibración precisa y experiencia previa con BCIs mejoran su eficiencia, subrayando la

necesidad de familiarización con la tecnología. En cuanto a la comodidad, la mayoría de los usuarios valoraron positivamente a NextMind, lo que indica su potencial para uso extendido.

7.2. Líneas futuras

7.2.1. Mejoras del prototipo

Con miras a continuar con el desarrollo y la mejora del proyecto, se han identificado diversas posibles acciones y modificaciones para mejorar su funcionamiento y expandir su utilidad.

- **Mejorar la cámara**: Se podría mejorar la resolución de la cámara para captar con mayor claridad los códigos QR. Una cámara más potente contribuiría a mejorar la efectividad y precisión en la detección de los códigos.
- **Implementar una función de deshacer estado**: Sería útil incorporar una función en el prototipo para revertir rápidamente el estado que se encuentra la cámara dentro de la deteción de QR, por ejemplo, a través de un botón o una tecla en el teclado que permita quitar un estado de manera sencilla.
- **Mejoras en la calibración y el ajuste del dispositivo**: Según las opiniones recogidas, el dispositivo en general no es difícil de poner, pero algunas personas con determinados tipos de cabello pueden tener dificultades. Por lo tanto, se podrían explorar opciones para facilitar su ajuste. Además, podría ser beneficioso tener un método alternativo para la calibración en caso de falla.
- **Incorporar un feedback acústico**: Algunos usuarios, al parecer, estaban tan enfocados en el estímulo que no notaban el feedback visual proporcionado por el NeuroTag. Para facilitar la percepción del usuario, se podría añadir un estímulo acústico adicional al visual.
- **Ampliar los controles con NeuroButtons**: Sería conveniente implementar NeuroButtons numerados del 1 al 5, donde cada número represente una cantidad específica de movimientos automáticos que puede realizar el sistema Pan-Tilt.
- **Explorar más dispositivos**: La aplicación del BCI se puede expandir a otros dispositivos, como una silla robotizada o un robot, lo que abriría nuevas posibilidades para el control a distancia de diversos aparatos mediante la interfaz cerebro-computadora.

7.2.2. Mejoras de investigación

- **Ampliar la muestra**: Una muestra más grande permitirá obtener resultados más representativos y confiables en las pruebas, pudiendo tener conclusiones más certeras.
- **Explorar el impacto del ruido ambiental**: Sería interesante, teniendo en cuenta que el presente prototipo está pensado para exteriores, estudiar la dificultad para concentrarse en ambientes ruidosos puede afectar el rendimiento del NextMind. Esto podría implicar la realización de más pruebas en estos contextos.

- **Investigar el efecto del TDAH**: El estudio de los efectos del Trastorno por Déficit de Atención e Hiperactividad (TDAH) en el uso del NextMind puede proporcionar perspectivas útiles ya que la concentración es vital en este BCI.
- **Estudiar la adaptación al dispositivo en personas con condiciones visuales**: Durante las pruebas se han encontrado ciertos patrones, aunque no suficientes, entre condiciones visuales y nota en la calibración. El estudio de cómo las personas con diferentes condiciones visuales (miopía, astigmatismo, hipermetropía, etc.) se adaptan al uso de NextMind podría ser interesante.
- **Analizar el efecto del entrenamiento prolongado**: Investigar si la eficacia del dispositivo puede mejorar con la práctica y el entrenamiento prolongado y determinar el tiempo necesario para observar mejoras significativas. Ya que durante las pruebas se vieron indicios claros de que teniendo experiencia previa en el uso de BCI se conseguía un mejor dominio del mismo.
- **Estudiar el efecto de la fatiga**: Investigar cómo la fatiga o el cansancio mental pueden afectar el rendimiento del NextMind. Esto podría ayudar a determinar cuánto tiempo puede ser usado eficazmente antes de que la fatiga comience a afectar el rendimiento.

Capítulo 8

Summary and Conclusions

8.1. Conclusions

In this section, we will discuss the conclusions.

Firstly, I have to highlight that all objectives have been met, although not everything went as expected.

8.1.1. Prototype development

As pointed out in chapter 6, the need to learn ROS from scratch represented a significant commitment of time and autonomous effort for understanding and assimilating information. Additionally, I had no previous experience programming in Unity, which also implied a learning process from scratch. Regarding specific programming on the ESP32, although it was unknown territory, I benefited from previous knowledge acquired in a degree course called Embedded Systems, which provided a solid foundation for general programming in microcontrollers. This was particularly useful during the programming of the pan-tilt from the microcontroller with respect to PWM, a task that, although it required some research, was simpler thanks to the skills acquired in that subject. Of all aspects of the project, what presented the least complications was the operation of the Neurotags and the NextMind system itself.

In terms of my personal perception of the tools used, I found ROS fascinating and it is a tool I would like to continue using in the future. My experience with Unity, on the other hand, was not as positive due to its logging and error management system, which I consider complicated to locate faults. As for ESP-IDF, it provided me with a satisfactory experience, perceiving it as a more modular structure than a micro-ROS project for Freertos microcontroller programming.

8.1.2. Interpretation of data and conclusions obtained

The results of the study point to the effectiveness and adaptability of NextMind in different environments and situations. As a BCI, it is accessible and resilient, not being influenced by aspects such as the presence of hair. Its efficient performance, both indoors and outdoors, and its resistance to variations in ambient light, open up new possibilities for future advances in BCI devices. It was observed that precise calibration and prior experience with BCIs improve its efficiency, underlining the need for familiarization with the technology. As for comfort, most users rated NextMind positively, indicating its potential for extended use.

8.2. Future lines

8.2.1. Prototype improvements

With a view to continuing the development and improvement of the project, various possible actions and modifications have been identified to improve its operation and expand its utility.

- **Improve the camera**: The camera's resolution could be improved to more clearly capture the QR codes. A more powerful camera would contribute to improving the effectiveness and precision in the detection of the codes.

- **Implement an undo state function**: It would be useful to incorporate a function in the prototype to quickly reverse the state of the camera within the QR detection, for example, through a button or a key on the keyboard that allows removing a state in a simple way.

- **Improvements in calibration and adjustment of the device**: According to the opinions collected, the device in general is not difficult to put on, but some people with certain types of hair may have difficulties. Therefore, options could be explored to facilitate its adjustment. In addition, it could be beneficial to have an alternative method for calibration in case of failure.

- **Incorporate acoustic feedback**: Some users, apparently, were so focused on the stimulus that they did not notice the visual feedback provided by the NeuroTag. To facilitate the user's perception, an additional acoustic stimulus could be added to the visual one.

- **Expand controls with NeuroButtons**: It would be convenient to implement numbered NeuroButtons from 1 to 5, where each number represents a specific amount of automatic movements that the Pan-Tilt system can perform.

- **Explore more devices**: The application of the BCI can be expanded to other devices, such as a robotic chair or a robot, which would open new possibilities for remote control of various devices through the brain-computer interface.

8.2.2. Research improvements

- **Expand the sample**: A larger sample will allow for more representative and reliable results in tests, and more accurate conclusions can be drawn.

- **Explore the impact of ambient noise**: Considering that this prototype is designed for outdoor use, it would be interesting to study how difficulty concentrating in noisy environments can affect the performance of NextMind. This could involve conducting more tests in these contexts.

- **Investigate the effect of ADHD**: Studying the effects of Attention Deficit Hyperactivity Disorder (ADHD) on the use of NextMind can provide useful insights as concentration is vital in this BCI.

- **Study the adaptation to the device in people with visual conditions**: During the tests, certain patterns were found, although not enough, between visual conditions and score in calibration. The study of how people with different visual conditions (myopia, astigmatism, hypermetropia, etc.) adapt to the use of NextMind could be interesting.

- **Analyze the effect of prolonged training**:

  Investigate if the effectiveness of the device can improve with practice and prolonged training and determine the time needed to observe significant improvements. As during the tests, clear indications were seen that having previous experience in the use of BCI resulted in better mastery of it.

- **Study the effect of fatigue**: Investigate how fatigue or mental tiredness can affect the performance of NextMind. This could help determine how long it can be effectively used before fatigue begins to affect performance.

Capítulo 9

Presupuesto

9.1. Presupuesto

Esta sección detalla el presupuesto del proyecto, contabilizando las horas de trabajo, los dispositivos necesarios y la licencia de Unity Pro.

La licencia de Unity Pro tiene un costo de 185€/mes. En el presupuesto se está contabilizando el precio de 2 meses de trabajo.

Concepto	Coste por horas	Horas	Coste total
Documentación	75	60	4500€
Horas de investigación	75	100	7500€
Horas de ingeniería	75	160	12000€
Licencia de Unity Pro	-	-	370€
NextMind Dev Kit SDK	-	-	399€
ESP32-WROOM-32D	-	-	3,65€
M5Stack Timer Camera ESP32 OV3660	-	-	19,95€
Pan-Tilt Casero: 2 unidades de HiTec HS-645MG	-	-	39,9€
	Total	320	24832,5€

Tabla 9.1: Presupuesto del proyecto

Capítulo 10

Bibliografía

[1] Abhang, P. A., Gawali, B. W., and Mehrotra, S. C. (2016). Chapter 2 - technological basics of eeg recording and operation of apparatus. In Abhang, P. A., Gawali, B. W., and Mehrotra, S. C., editors, *Introduction to EEG- and Speech-Based Emotion Recognition*, pages 19–50. Academic Press.

[2] Ang, K. K., Guan, C., Chua, K. S. G., Ang, B. T., Kuah, C., Wang, C., Phua, K. S., Chin, Z. Y., and Zhang, H. (2010). Clinical study of neurorehabilitation in stroke using eeg-based motor imagery brain-computer interface with robotic feedback. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, pages 5549–5552.

[3] Casali, R. L., Amaral, M. I., Boscariol, M., Lunardi, L. L., Guerreiro, M. M., Matas, C. G., and Colella-Santos, M. F. (2016). Comparison of auditory event-related potentials between children with benign childhood epilepsy with centrotemporal spikes and children with temporal lobe epilepsy. *Epilepsy & Behavior*, 59:111–116.

[4] eProsima (2023). Micro xrce-dds: Middleware concept. [Online; accessed 11-July-2023].

[5] Erkan, E. and Akbaba, M. (2018). A study on performance increasing in ssvep based bci application. *Engineering Science and Technology, an International Journal*, 21(3):421–427.

[6] Escolano, C., Antelis, J. M., and Minguez, J. (2012). A telepresence mobile robot controlled with a noninvasive brain–computer interface. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(3):793–804.

[7] Inc, S. (2012). Snap inc website. `https://www.snap.com/`. Retrieved June 05, 2023.

[8] Marichal, G.Ñ., Acosta, L., Toledo, J., Marichal, R. L., and Torres, S. (2004). A new approach in controlling the motors of a binocular camera head. *International Mathematical Journal*, 5-7:581–587.

[9] NextMind (2022). Known issues and workarounds. [Online; accessed 11-July-2023].

[10] Sunny, T., Aparna, T., Neethu, P., Venkateswaran, J., Vishnupriya, V., and Vyas, P. (2016). Robotic arm with brain – computer interfacing. *Procedia Technology*, 24:1089–1096. International Conference on Emerging Trends in Engineering, Science and Technology (ICETEST - 2015).

[11] Systems, E. (2016). Esp32 ledc (led controller). [Online; accessed 12-July-2023].

[12] Wire, B. (2020). Time to open your nextmind: Devkit for world's first real-time brain-sensing wearable starts shipping. https://www.businesswire.com/news/home/20201208005141/en/Time-to-Open-Your-NextMind-DevKit-for-Worlds-First-Real-Time-Brain-Sensing-Weara Retrieved June 05, 2023.

[13] Wu, C.-M., Chen, Y.-J., Zaeni, I. A. E., and Chen, S.-C. (2016). A new ssvep based bci application on the mobile robot in a maze game. In *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*, pages 550–553.

Apéndice A

Preparación del entorno de desarrollo

En este apéndice, se detallará la configuración del entorno de desarrollo utilizado para este proyecto. El ecosistema de desarrollo se compone principalmente de Unity, el middleware ROS, y la librería Ros2ForUnity. Se explicarán los pasos necesarios para instalar y ajustar cada uno de estos componentes para asegurar un funcionamiento óptimo. Por último se explicará el funcionamiento y la experiencia tenida con otra librería llamada Unity-Robotics-Hub.

A.1. NextMind SDK

A.1.1. Requerimientos del SDK

Para poder usar el SDK de NextMind es importante tener en cuenta que tiene una serie de requerimientos que se muestran a continuación:

Requisitos mínimos de hardware

- Soporte Bluetooth LE (4.0)
- Gráficos - DX9 shader model 2.5, equivalente a Intel HD 2500
- CPU - Intel i5-4590, equivalente a AMD FX 8350
- RAM - 8 GB

Compatibilidad de software

- Última versión probada: Unity - 2022LTS
- Versión oficial soportada: Unity - 2020LTS, 2019LTS
- Plataformas - Windows 10 de 64 bits, Apple macOS de 64 bits (Mojave, Catalina, Big Sur)
- Software probado y aprobado - Oculus Rift, Oculus Quest 1 y 2, HTC Vive y Pro, HoloLens 1
- Compatibilidad de software verificada - Valve Index, HoloLens 2

A.1.2. Instalación del SDK

La instalación del NextMind SDK es un proceso sencillo que se realiza a través de la plataforma Unity. En primer lugar, es necesario descargar el SDK desde el sitio web oficial de NextMind o directamente desde el Unity Asset Store. Tras la descarga, el SDK puede ser importado al proyecto de Unity utilizando el menú “Assets”.

A.1.3. Desarrollo con el SDK

Con el SDK correctamente instalado, ahora se puede comenzar a desarrollar aplicaciones que interactúan con el dispositivo NextMind. La documentación del SDK proporciona una serie de tutoriales y ejemplos que pueden ayudar a comenzar.

A.2. ROS 2

La librería Ros2ForUnity requiere la instalación de ROS 2 en Windows (Windows 10), así como en el Subsistema de Windows para Linux (WSL), específicamente Ubuntu 22.04.2 LTS. Aunque no todas las librerías exigen la instalación de ROS2 en Windows, para Ros2ForUnity es imprescindible si tenemos Unity en Windows, como es el caso.

A.2.1. Instalación de ROS 2 en Ubuntu

Se optó por la versión Humble de ROS 2 para Ubuntu, siguiendo las instrucciones proporcionadas en la documentación oficial[1]. Después de la instalación, se llevaron a cabo varias pruebas para confirmar que todo funcionaba correctamente, como se puede ver en la Figura A.1.

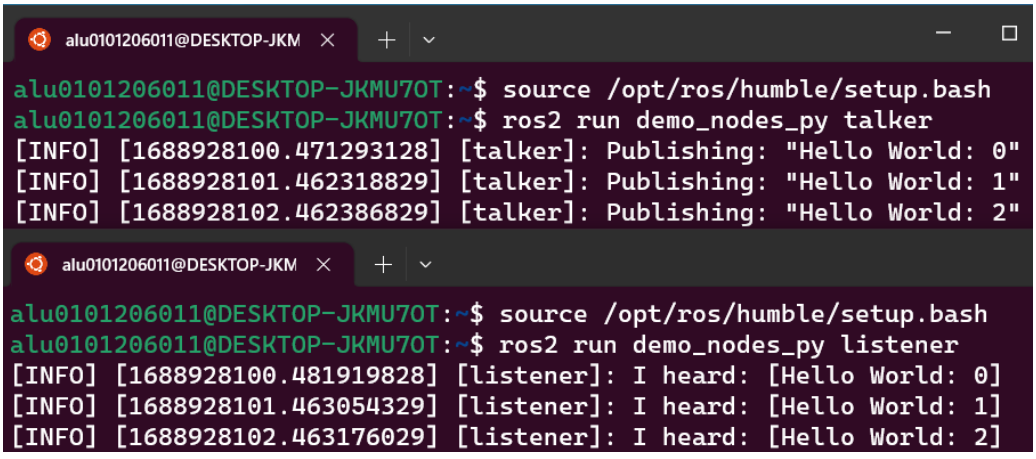


Figura A.1: Ejemplos de funcionamiento de la instalación de ROS 2 Ubuntu

[1]ROS Humble Installation Ubuntu: `https://docs.ros.org/en/humble/Installation/Alternatives/Ubuntu-Development-Setup.html`

A.2.2. Instalación de ROS 2 en Windows 10

Para instalar ROS 2 en Windows 10, se siguió la documentación oficial de ROS 2 Humble[2]. Se recomienda realizar la instalación desde la PowerShell con privilegios de administrador. Dado que el comando çall"no está disponible en PowerShell, se sustituyó por:

```
C:\dev\ros2\_humble\local\_setup.ps1
```

Es importante tratar con cuidado a Chocolatey, ya que puede tener dificultades para desinstalar completamente los paquetes que instala. Si se produce un error, será necesario eliminar los paquetes manualmente a través del editor de registros de Windows.

Finalmente, es crucial prestar atención a las versiones de .Net y al Visual Studio 2019. Visual Studio 2019, se debe asegurar que no se instalan las herramientas C++ CMake, deseleccionándolas en la lista de componentes a instalar.

A.3. ESP-IDF

Para poder instalar el ESP-IDF release 5.1 de Linux en ESP32, se necesita instalar el siguiente software:

- Conjunto de herramientas de compilación para el ESP32.
- Herramientas de construcción: CMake y Ninja, para construir una aplicación completa para ESP32.
- ESP-IDF, que contiene esencialmente la API (bibliotecas de software y código fuente) para ESP32, así como scripts para operar el conjunto de herramientas.

A.3.1. Instalación de ESP-IDF

La instalación del ESP-IDF se debe llevar a cabo siguiendo las instrucciones de su documentación oficial[3]. La versión instalada en este proyecto es la release 5.1.

Aunque la documentación del ESP-IDF proporciona instrucciones de instalación para varios sistemas operativos, en este proyecto se decidió instalarlo en Linux. Esta decisión se tomó en las etapas iniciales del proyecto, incluso antes de considerar el uso de ROS 2 en Windows. En consecuencia, toda la gestión del ESP32 se realiza desde Linux, aprovechando la instalación existente de ROS 2 en la WSL.

Además, cabe destacar que micro-ROS no proporciona instrucciones para su instalación en Windows y no se puede garantizar que el proceso de compilación funcione correctamente en ese sistema operativo. Esto también influenció la elección de utilizar Linux para el desarrollo del proyecto.

[2] ROS Humble Installation Windows: https://docs.ros.org/en/humble/Installation/Alternatives/Windows-Development-Setup.html

[3] ESP-IDF Get Started Guide: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/

A.3.2. Consideraciones con la WSL

Es importante tener en cuenta que los puertos USB están gestionados por el sistema operativo Windows. Para hacerlos accesibles desde la WSL, se utilizó el programa usbipd[4] en Windows. Primero, se listan los puertos USB conectados para obtener el BUSID con el siguiente comando:

```
usbipd wsl list
```

Luego, se ejecuta el siguiente comando para "atar" el puerto USB a la WSL:

```
usbipd wsl attach --busid (busid)
```

Los puertos en sistemas Linux se ubican en el directorio **/dev/** y suelen tener nombres del tipo `ttyUSBX`, donde X es un número entero positivo.

Es necesario otorgar permisos de lectura y escritura al puerto para su uso. Para ello, se puede utilizar el siguiente comando:

```
sudo chmod a+rw /dev/ttyUSBX
```

En este comando, es importante reemplazar `ttyUSBX` con el nombre del puerto que se desea utilizar.

A.3.3. Creación de componentes

Requisitos para la creación componentes:

1. **Modularidad:** Ser una unidad funcionalmente independiente y cohesiva.
2. **Reutilización:** Ser capaz de utilizarse en diferentes contextos y proyectos.
3. **Interfaz bien definida:** Especificar cómo interactuar con el componente.
4. **Encapsulación:** Ocultar los detalles internos de implementación y proporcionar una interfaz de alto nivel.
5. **Independencia:** No depender directamente de otros componentes.
6. **Coherencia y cohesión:** Tener una estructura y diseño coherentes, así como una funcionalidad interna relacionada.
7. **Testeabilidad:** Ser fácil de probar de forma unitaria y escribir pruebas automatizadas.
8. **Documentación:** Estar debidamente documentado con información relevante.

En el contexto de ESP-IDF, añadir un componente es tan sencillo como ponerlo en el directorio de components.

Para que se añada correctamente hay que añadir en el CMakeLists.txt de fuera lo siguiente:

[4] USBIPD-WIN project: `https://learn.microsoft.com/en-us/windows/wsl/connect-usb`

Listing A.1: CMakeLists del repositorio ESP-IDF

```
cmake_minimum_required(VERSION 3.5)

set (EXTRA_COMPONENT_DIRS "components/")

include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(pan-tilt-esp32)
```

Luego dentro de cada componente hay que tener otro CMakeLists.txt con lo siguiente:

Listing A.2: CMakeLists del componente ESP-IDF

```
idf_component_register(
    SRCS "script1.c" "script2.c"
    INCLUDE_DIRS "."
    REQUIRES <alguna dependencia>
)
```

También se le puede añadir un directorio de test para testear el componente usando U-TTS.

Dentro de ese directorio debe haber otro CMakeLists.txt:

Listing A.3: CMakeLists de los tests del componente

```
idf_component_register(SRC_DIRS "."
                       INCLUDE_DIRS "."
                       REQUIRES <nombre del componente> unity)
```

Luego al mismo nivel en el directorio debe haber un fichero con los tests incluyendo la librería de U-TTS y el nombre del componente a testear:

```
#include "unity.h"
```

A.3.4. Componente de micro-ROS

Como se mencionó anteriormente, en el marco de trabajo con ESP-IDF, MicroROS se añade como un componente[5]. Este componente ha sido validado en las versiones ESP-IDF v4.3, v4.4 y v5.0, siendo compatible con ESP32, ESP32-S2, ESP32-S3 y ESP32-C3.

Dependencias

Para el correcto funcionamiento de este componente, es necesario instalar **colcon** y otros paquetes de Python 3 dentro del entorno virtual de IDF, para la construcción de paquetes de micro-ROS:

```
. $IDF_PATH/export.sh
pip3 install catkin_pkg lark-parser empy colcon-common-extensions
```

Middlewares disponibles

El componente de micro-ROS permite trabajar con dos middlewares diferentes:

[5]Micro-ROS Component: `https://github.com/micro-ROS/micro_ros_espidf_component`

- **eProsima Micro XRCE-DDS:** Este es el middleware predeterminado de micro-ROS. XRCE-DDS es un protocolo diseñado para permitir comunicaciones eficientes y efectivas entre clientes y agentes en sistemas embebidos, siendo por tanto adecuado para entornos con recursos limitados.

- **EmbeddedRTPS:** Se trata de una implementación experimental de un middleware RTPS compatible con ROS 2. El middleware RTPS (Real-Time Publish-Subscribe) es un protocolo de capa de middleware que proporciona comunicación en tiempo real en redes de área amplia. Este middleware se utiliza en sistemas distribuidos y entornos de tiempo real.

Para seleccionar uno de estos middlewares, se debe usar el comando **idf.py menuconfig** y acceder a **micro-ROS Settings >micro-ROS middleware**.

En el presente proyecto, se optó por el uso del middleware predeterminado, eProsima Micro XRCE-DDS.

Instalación del componente

El componente de MicroROS puede incorporarse al proyecto ESP-IDF clonando directamente el repositorio correspondiente en el directorio **components**. Es importante destacar que la rama a clonar del repositorio debe coincidir con la versión de ROS2 que se está utilizando. En el contexto de este proyecto, dado que se está utilizando ROS2 Humble, se ha clonado la rama "Humble"del repositorio.

En caso de enfrentar dificultades durante el proceso de compilación, es recomendable verificar que se está operando en un entorno de shell limpio, es decir, sin que el script de configuración de ROS 2 esté en funcionamiento.

A.3.5. Guía de uso

Esta sección presenta una guía paso a paso para poner en marcha el proyecto:

- Ejecutar el comando `idf.py set-target esp32` para seleccionar el microcontrolador ESP32 como objetivo.

- Configurar el proyecto ejecutando `idf.py menuconfig`.

- Construir el proyecto con `idf.py build`.

- Conectar el dispositivo y luego cargar el proyecto en el dispositivo con el comando `idf.py -p PORT flash`, sustituyendo `PORT` por el puerto correspondiente del dispositivo.

- Monitorear la salida de la aplicación con `idf.py -p PORT monitor`.

Es importante reemplazar `PORT` con el puerto apropiado para su sistema teniendo en cuenta lo visto en el apartado A.3.2.

A.4. Agente de micro-ROS

El agente de micro-ROS es un componente fundamental del framework de micro-ROS. Sirve de intermediario entre los nodos de micro-ROS en el microcontrolador y los nodos de ROS2 en el sistema del ordenador. En este caso particular, se ejecutará micro-ROS en Linux a través de la WSL. Para lograr esto, utiliza la implementación eProsima Micro XRCE-DDS del protocolo DDS para conectar las funcionalidades de micro-ROS con las funcionalidades más extensas de ROS2 [4].

Existen varias formas de ejecutar el agente de micro-ROS, dos de las cuales se detallan a continuación:

A.4.1. Usando un repositorio de micro-ROS

Una opción para ejecutar el agente de micro-ROS es a través de un repositorio de micro-ROS. La instalación de dicho repositorio se realiza de manera sencilla siguiendo los primeros pasos del tutorial oficial de micro-ROS[6].

Para ejecutar el agente, se utiliza la utilidad *micro_ros_agent* incluida en el repositorio:

```
ros2 run micro_ros_agent micro_ros_agent [transport] [options]
```

El parámetro [transport] especifica el tipo de transporte utilizado para la comunicación (por ejemplo, udp, serial, etc.). Los [options] son argumentos adicionales específicos del tipo de transporte seleccionado.

Por ejemplo, el siguiente comando utiliza una comunicación serial a través del dispositivo `/dev/ttyUSB0`:

```
ros2 run micro_ros_agent micro_ros_agent serial /dev/ttyUSB0
```

Para ejecutar cualquier comando de ROS2, es necesario inicializar previamente el entorno de ROS2.

A.4.2. Uso de Docker para ejecutar el agente de micro-ROS

Alternativamente, es posible utilizar Docker para ejecutar el agente de micro-ROS. Inicialmente, se debe iniciar el daemon de Docker con el siguiente comando:

```
sudo dockerd
```

Posteriormente, hay que ejecutar el agente de micro-ROS utilizando el siguiente comando:

```
sudo docker run -it --rm -v /dev:/dev --privileged --net=host
microros/micro-ros-agent:humble serial --dev /dev/ttyUSB0 -v6
```

Este comando ejecuta el agente de micro-ROS dentro de un contenedor Docker, otorgándole acceso a los dispositivos del sistema anfitrión y configurando la red del contenedor para utilizar la red del sistema anfitrión. En este caso, se está utilizando el agente de micro-ROS para la versión “humble” de ROS, utilizando una conexión serial a través del dispositivo `/dev/ttyUSB0`.

[6] First micro-ROS Application on Linux: `https://micro.ros.org/docs/tutorials/core/first_application_linux/`

```
alu0101206011@DESKTOP-JKMU7OT:~/TFG-ROS2/ros2_agent$ ros2 run micro_ros_agent micro_ros_agent serial --dev /de
v/ttyUSB0
[1689192173.580115] info     | TermiosAgentLinux.cpp | init                     | running...             | fd:
 3
[1689192173.580783] info     | Root.cpp              | set_verbose_level        | logger setup           | verbos
e_level: 4
[1689192177.196111] info     | Root.cpp              | create_client            | create                 | client
_key: 0x0368435D, session_id: 0x81
[1689192177.196176] info     | SessionManager.hpp    | establish_session        | session established    | client
_key: 0x0368435D, address: 0
[1689192177.216696] info     | ProxyClient.cpp       | create_participant       | participant created    | client
_key: 0x0368435D, participant_id: 0x000(1)
[1689192177.239535] info     | ProxyClient.cpp       | create_topic             | topic created          | client
_key: 0x0368435D, topic_id: 0x000(2), participant_id: 0x000(1)
[1689192177.261681] info     | ProxyClient.cpp       | create_subscriber        | subscriber created     | client
_key: 0x0368435D, subscriber_id: 0x000(4), participant_id: 0x000(1)
[1689192177.288309] info     | ProxyClient.cpp       | create_datareader        | datareader created     | client
_key: 0x0368435D, datareader_id: 0x000(6), subscriber_id: 0x000(4)
[1689192177.383376] info     | ProxyClient.cpp       | create_topic             | topic created          | client
_key: 0x0368435D, topic_id: 0x001(2), participant_id: 0x000(1)
[1689192177.405122] info     | ProxyClient.cpp       | create_publisher         | publisher created      | client
_key: 0x0368435D, publisher_id: 0x000(3), participant_id: 0x000(1)
[1689192177.427706] info     | ProxyClient.cpp       | create_datawriter        | datawriter created     | client
_key: 0x0368435D, datawriter_id: 0x000(5), publisher_id: 0x000(3)
```

Figura A.2: Ejecución del agente de micro-ROS

A.4.3. Consideraciones adicionales

El funcionamiento de micro-ROS depende de la ejecución de un programa agente que expone los nodos de ROS2 creados en un microcontrolador de modo que sean visibles para toda la red accesible por ROS2. El agente micro-ROS puede comunicarse con el dispositivo micro-ROS de dos formas: por comunicación serial o mediante el protocolo UDP. En el primer caso, la comunicación entre el programa micro-ROS en el dispositivo y el agente no es problemática porque el programa conecta directamente con el agente con el sistema serial. Sin embargo, si se usa UDP por ejemplo en una red wifi, hay que proporcionar al programa micro-ROS, la dirección y el puerto del agente dentro de esa red. El problema radica en que el subsistema windows de linux no es accesible directamente utilizando la IP asignada por Windows a la interfaz de red virtual creada para WSL. Se puede crear una pasarela para comunicarse con una aplicación en WSL mediante la herramienta netfs, pero bajo el protocolo TCP, y no funciona en la actualidad para el protocolo UDP. La solución para no depender de un ordenador adicional bajo linux, pasaría por utilizar una máquina virtual Virtual box en Windows para ejecutar el agente, ya que este sistema sí admite la creación de un puente con el interfaz de red wifi.

A.5. Ros2ForUnity

Ros2ForUnity es una librería de Unity que permite crear nodos de ROS 2 con código C# de Unity.

A.5.1. Instalación de Ros2ForUnity

Para instalar la librería se han seguido los documentos markdown del repositorio de GitHub del proyecto [7]. En este proyecto se ha comprobado su funcionamiento hasta el commit:

[7] Ros2ForUnity Windows Installation: `https://github.com/RobotecAI/ros2-for-unity/blob/develop/README-WINDOWS.md`

```
3f548920bc4c33e178707a888d01592905bef1e9
```

Hay que prestar especial atención a los prerequisitos y a los avisos importantes que se señalizan dentro del "README-WINDOWS.md" del repositorio. Y en ejecutar la Powershell en modo administrador.

Al momento de construir el proyecto Ros2ForUnity, es necesario seleccionar la versión "overlay". Esta elección se debe a que la versión "standalone" proporciona una instancia única de ROS dentro de Unity, que no corresponde al comportamiento deseado para este proyecto.

Ahora hay dos opciones para instalar el asset en Unity. Ejecutar el comando siguiente para crear un package e instalarlo en Unity de manera habitual:

```
create\_unity\_package.ps1
```

O copiar y pegar en los assets del proyecto de Unity el directorio que se encuentra en **install/asset/**.

A.5.2. Uso de Ros2ForUnity con ROS2

Una vez instalado Ros2ForUnity en Unity, es importante recordar que se debe iniciar Unity a través de la terminal después de iniciar ROS2. Esto se debe a que Ros2ForUnity necesita acceso a la instancia de ROS2 que se ejecuta en el sistema para funcionar correctamente.

Primero, se debe iniciar ROS2. Si estás utilizando ROS2 Humble en Windows, puedes hacerlo ejecutando el siguiente comando en PowerShell:

```
C:\dev\ros2_humble\local_setup.ps1
```

Asegúrate de que tu terminal esté en el directorio que contiene tu script de configuración de ROS2 local antes de ejecutar este comando.

Después de haber iniciado ROS2, puedes abrir tu proyecto de Unity a través de la terminal utilizando el comando unity con la opción -projectPath seguido de la ruta a tu proyecto:

```
unity.exe -projectPath "D:\Unity\Unity Projects\Thesis-Unity-Pan-Tilt"
```

Con ROS2 en funcionamiento y Unity abierto de esta manera, deberías ser capaz de utilizar Ros2ForUnity para crear nodos de ROS2 con el código C# de Unity.

A.5.3. Consideraciones

Es necesario asignar a la variable de entorno ROS_IP una dirección IP que permita la comunicación entre ROS2 en la WSL y ROS2 en el sistema operativo Windows.

La necesidad de usar la dirección IP de la WSL se origina del hecho de que la comunicación entre la WSL y Windows se realiza a través de una red virtual interna. Esta dirección IP asegura que los mensajes enviados por ROS2 en la WSL sean recibidos por ROS2 en Windows y viceversa. Se puede observar en la figura A.3.

Comandos para asignar la dirección IP:

En Windows (usando PowerShell):

```
$env:ROS_IP='172.29.67.184'
```

En WSL:

```
export ROS_IP=172.29.67.184
```

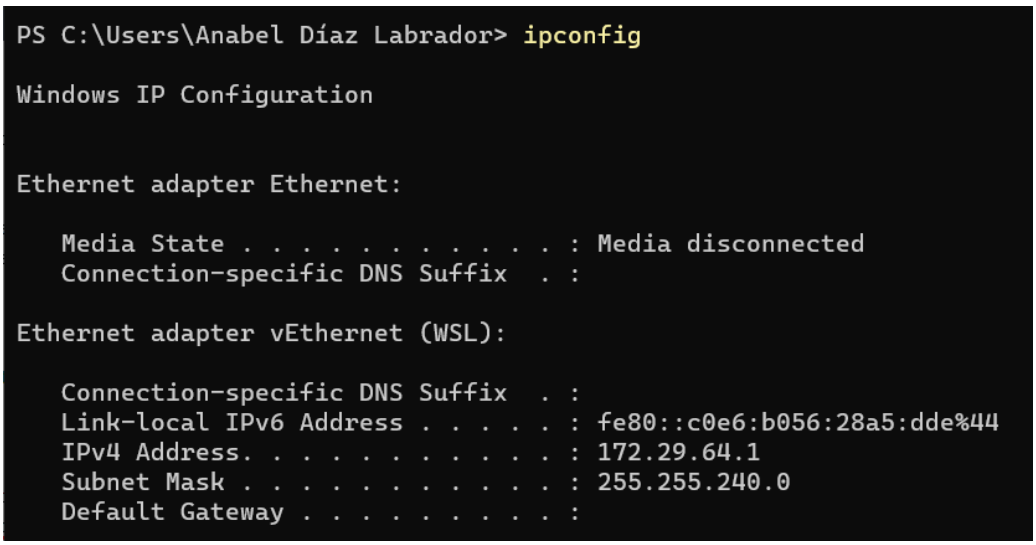


Figura A.3: Red de la WSL visualizada en PowerShell

A.6. Cámara

La cámara utilizada es una M5 Stack Timercam, como se menciona en la sección 3.9.

A.6.1. Instalación del entorno de trabajo

El software de la cámara es específico y solo es compatible con la versión ESP-IDF v4.0.1. Para garantizar una correcta instalación, es necesario clonar esta versión específica del repositorio.

```
git clone -b v4.0.1 --recursive
https://github.com/espressif/esp-idf.git esp-idf-v4.0.1
```

Posteriormente, es necesario instalar el entorno de desarrollo descargado para poder interactuar con él.

```
~/esp_idf_4.0/esp-idf-v4.0.1$ ./install.sh
```

Para poder compilar el código de la cámara o monitorizarla desde la terminal, entre otras acciones, siempre será necesario ejecutar el siguiente comando en el directorio “ /esp_idf_4.0/esp-idf-v4.0.1”:

```
source export.sh
```

Es importante mencionar que no se puede ejecutar este comando después de haber hecho un source export.sh de otro ESP-IDF.

Para facilitar el uso de estos comandos, se han creado los siguientes alias que se deben añadir al archivo .bashrc:

```
alias get_idf=’. $HOME/esp/esp-idf/export.sh’
alias get_idf4=’. $HOME/esp_idf_4.0/esp-idf-v4.0.1/export.sh’
```

Apéndice B

Códigos relevantes del proyecto

B.1. Script Don't Destroy On Load

Listing B.1: Código de Don't Destroy On Load

```
using UnityEngine;

public class DontDestroyOnLoad : MonoBehaviour {
  public static DontDestroyOnLoad Instance;

  void Awake() {
    if (Instance == null) {
      Instance = this;
      DontDestroyOnLoad(gameObject);
    } else {
      Destroy(gameObject);
    }
  }
}
```

B.2. Script Ros2Start.cs

Listing B.2: Código de Ros2Start

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ROS2;
using System.Collections.Concurrent;

public class Ros2Start : MonoBehaviour {
  public static Ros2Start Instance { get; private set; }

  private ROS2UnityComponent ros2Unity;
  private ROS2Node ros2Node;
  public IPublisher<std_msgs.msg.Header> action_pub;
  private ISubscription<std_msgs.msg.Header> esp32_sub;
```

```
  public delegate void MessageReceivedHandler(std_msgs.msg.Header msg);
  public event MessageReceivedHandler OnMessageReceived;

  // Thread-safe queue for incoming messages
  public ConcurrentQueue<std_msgs.msg.Header> messageQueue = new
      ConcurrentQueue<std_msgs.msg.Header>();

  // Start is called before the first frame update
  void Start() {
    if (Instance != null) {
      Destroy(gameObject);
      return;
    }
    Instance = this;
    DontDestroyOnLoad(gameObject);

    Debug.Log("Starting ROS2 node");

    ros2Unity = GetComponent<ROS2UnityComponent>();
    if (ros2Unity.Ok()) {
      ros2Node = ros2Unity.CreateNode("ROS2UnityPublisherSubscriberNode");
      Debug.Log("Starting ROS2 node 2");
    } else {
      Debug.Log("Ros2Unity was not created OK");
    }

    if (ros2Unity.Ok()) {
      action_pub = ros2Node.CreatePublisher<std_msgs.msg.Header>("action");
      esp32_sub = ros2Node.CreateSubscription<std_msgs.msg.Header>("
          freertos_header_log",
        msg => {
          Debug.Log("Unity listener heard: [" + msg.Frame_id + "]");
          messageQueue.Enqueue(msg);
        });
      std_msgs.msg.Header msg = new std_msgs.msg.Header();
      msg.Frame_id = "CON_/CAM";
      Debug.Log(msg.Frame_id);
      action_pub.Publish(msg);
    } else {
      Debug.Log("ROS2UnityComponent was not created OK");
    }
  }


  void Update() {
    while (messageQueue.TryDequeue(out std_msgs.msg.Header msg)) {
        OnMessageReceived?.Invoke(msg);
    }
  }
}
```

B.3. Script Ros2SubscriberHandler.cs

Listing B.3: Código de Ros2SubscriberHandler

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Ros2SubscriberHandler : MonoBehaviour
{
  public static Ros2SubscriberHandler Instance { get; private set; }

  public Ros2Start ros2Subscriber;
  public bool CameraEnabled { get; set; }
  public string CameraIP { get; set; }
  public bool ESP32Enabled { get; set; }

  private void Awake() {
    if (Instance != null) {
      Destroy(gameObject);
      return;
    }
    Instance = this;
    DontDestroyOnLoad(gameObject);
  }

  void Start() {
    CameraEnabled = false;
    CameraIP = "No camera IP";
    ESP32Enabled = false;
  }

  // Manage the event of receiving a message
  void OnEnable() {
    if (ros2Subscriber == null) {
      ros2Subscriber = GetComponent<Ros2Start>();
      if (ros2Subscriber != null) {
        ros2Subscriber.OnMessageReceived += ProcessReceivedMessage;
      }
    } else {
      ros2Subscriber.OnMessageReceived += ProcessReceivedMessage;
    }
  }

  void ProcessReceivedMessage(std_msgs.msg.Header msg) {
    string[] parts = msg.Frame_id.Split(' ');

    if (parts.Length >= 5 && parts[0] == "Target" && parts[1] == "station" &&
        parts[2] == "has" && parts[3] == "ip") {
      CameraEnabled = true;
      CameraIP = parts[4];
```

```
      Debug.Log("IP camera: " + CameraIP);
    }

    if (parts.Length >= 3 && parts[0] == "Camera" && parts[1] == "ip:") {
      CameraEnabled = true;
      CameraIP = parts[2];
      Debug.Log("IP camera: " + CameraIP);
    }

    if (parts.Length >= 3 && parts[0] == "Target" && parts[1] == "station" &&
        parts[2] == "disconnected") {
      CameraEnabled = false;
      CameraIP = "No camera IP";
      Debug.Log("Camera disabled");
    }

    if (parts.Length >= 4 && parts[0] == "Wifi" && parts[1] == "init" &&
        parts[2] == "softap" && parts[3] == "finished") {
      ESP32Enabled = true;
      Debug.Log("ESP32 enabled");
    }

    if (parts.Length >= 4 && parts[0] == "Stopping" && parts[1] == "WiFi" &&
        parts[2] == "in" && parts[3] == "base" && parts[4] == "station") {
      ESP32Enabled = false;
      Debug.Log("ESP32 disabled");
      CameraIP = "No camera IP";
      CameraEnabled = false;
    }
  }
}
```

B.4. Script Ros2Publish

Listing B.4: Código de Ros2Publish

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ROS2;

public class Ros2Publish : MonoBehaviour {
  private Ros2Start ros2Publisher;
  private IPublisher<std_msgs.msg.Header> action_pub;


  void Update() {
    if (ros2Publisher == null) {
      ros2Publisher = Ros2Start.Instance;
    }
```

```
  }


  public void OnUpNeuroButtonPress() {
    if(ros2Publisher != null && ros2Publisher.action_pub != null) {
      std_msgs.msg.Header msg = new std_msgs.msg.Header();

      msg.Frame_id = "ACT_/Up";
      ros2Publisher.action_pub.Publish(msg);
    }
  }

  // ... Omitiendo codigo por similitud
}
```

B.5. Script CameraInfo.cs

Listing B.5: Código de CameraInfo

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraInfo : MonoBehaviour {
  public bool GetState() {
    return Ros2SubscriberHandler.Instance.CameraEnabled;
  }

  public string GetIP() {
    return Ros2SubscriberHandler.Instance.CameraIP;
  }

  public string GetUrl() {
    if (GetState() == false) {
      return "No camera URL";
    }
    return "http://" + GetIP() + ":81";
  }

  public string GetUrlApiRest() {
    if (GetState() == false) {
      return "No camera URL";
    }
    return "http://" + GetIP() + ":80";
  }
}
```

B.6. Script CameraNetworkCommandController.cs

Listing B.6: Código de CameraNetworkCommandController

```
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;
using TMPro;

public class CameraNetworkCommandController : MonoBehaviour {

  public CameraInfo cameraInfo;
  public TMPro.TextMeshProUGUI responseText;
  public TMPro.TMP_InputField setStateInputField;

  public void SendRestartCommand() {
    responseText.text = "";
    if (cameraInfo.GetState() == false) {
      Debug.Log("Camera is not enabled");
      responseText.text = "Command error: Camera is not enabled";
      return;
    }
    string command = "/app?app=0&value=0";
    Debug.Log("Sending command: " + command);
    StartCoroutine(SendCommand(command));
  }


  public void RequestCurrentState() {
    responseText.text = "";
    if (cameraInfo.GetState() == false) {
      Debug.Log("Camera is not enabled");
      responseText.text = "Command error: Camera is not enabled";
      return;
    }
    string command = "/app?app=1&value=0";
    Debug.Log("Sending command: " + command);
    StartCoroutine(SendCommand(command));
  }


  public void SetState() {
    responseText.text = "";
    if (cameraInfo.GetState() == false) {
      Debug.Log("Camera is not enabled");
      responseText.text = "Command error: Camera is not enabled";
      return;
    }
    int stateValue;
    if(int.TryParse(setStateInputField.text, out stateValue)) {
```

```
        Debug.Log(cameraInfo);
        string command = "/app?app=2&value=" + stateValue;
        Debug.Log("Sending command: " + command);
        StartCoroutine(SendCommand(command));
    } else {
        Debug.Log("Input value is not a valid integer");
        responseText.text = "Command error: Input value is not a valid integer
            ";
    }
}


public void StopStream() {
    responseText.text = "";
    if (cameraInfo.GetState() == false) {
      Debug.Log("Camera is not enabled");
      responseText.text = "Command error: Camera is not enabled";
      return;
    }
    string command = "/app?app=3&value=0";
    Debug.Log("Sending command: " + command);
    StartCoroutine(SendCommand(command));
}


public void ContinueMode() {
    responseText.text = "";
    if (cameraInfo.GetState() == false) {
      Debug.Log("Camera is not enabled");
      responseText.text = "Command error: Camera is not enabled";
      return;
    }
    string command = "/app?app=3&value=1";
    Debug.Log("Sending command: " + command);
    StartCoroutine(SendCommand(command));
}


public void RestartCamera() {
    responseText.text = "";
    if (cameraInfo.GetState() == false) {
      Debug.Log("Camera is not enabled");
      responseText.text = "Command error: Camera is not enabled";
      return;
    }
    string command = "/app?app=4&value=0";
    Debug.Log("Sending command: " + command);
    StartCoroutine(SendCommand(command));
}
```

```
  public void GetQR() {
    responseText.text = "";
    if (cameraInfo.GetState() == false) {
      Debug.Log("Camera is not enabled");
      responseText.text = "Command error: Camera is not enabled";
      return;
    }
    string command = "/app?app=5&value=0";
    Debug.Log("Sending command: " + command);
    StartCoroutine(SendCommand(command));
  }


  private IEnumerator SendCommand(string command) {
    string fullUrl = cameraInfo.GetUrlApiRest() + command;
    Debug.Log(fullUrl);

    using (UnityWebRequest webRequest = UnityWebRequest.Get(fullUrl)) {
      webRequest.timeout = 60;

      // Request and wait for the desired page.
      yield return webRequest.SendWebRequest();

      string[] pages = fullUrl.Split('/');
      int page = pages.Length - 1;

      switch (webRequest.result) {
        case UnityWebRequest.Result.ConnectionError:
        case UnityWebRequest.Result.DataProcessingError:
          responseText.text = pages[page] + ": Error: " + webRequest.error;
          break;
        case UnityWebRequest.Result.ProtocolError:
          responseText.text = pages[page] + ": HTTP Error: " + webRequest.
              error;
          break;
        case UnityWebRequest.Result.Success:
          responseText.text = pages[page] + ":\nReceived: " + webRequest.
              downloadHandler.text;
          break;
      }
    }
  }
}
```

B.7. Software libre modificado: ImageStreamUrl.cs

Listing B.7: Código de ImageStreamUrl

```
using System.IO;
using System.Threading;
```

```
using System.Collections;
using System.Collections.Generic;
using System.Net;
using UnityEngine;

public class ImageStreamUrl : MonoBehaviour {
  public static ImageStreamUrl Instance { get; private set; }

  private CameraInfo cameraInfo;
  private TMPro.TextMeshProUGUI errorText;

  [SerializeField] bool tryOnStart = false;

  [SerializeField] RenderTexture renderTexture;

  int MAX_RETRIES = 3;
  int retryCount = 0;

  byte[] nextFrame = null;

  Thread worker;
  int threadID = 0;

  static System.Random randu;
  List<BufferedStream> trackedBuffers = new List<BufferedStream>();

  private void Awake() {
    if (Instance != null) {
      Destroy(gameObject);
      return;
    }
    Instance = this;
    DontDestroyOnLoad(gameObject);
  }

  public void Initialize(CameraInfo camInfo, TMPro.TextMeshProUGUI errText) {
    cameraInfo = camInfo;
    errorText = errText;
  }

  void Start() {
    randu = new System.Random(Random.Range(0, 65536));
    if (tryOnStart) {
      StartStream();
      Debug.Log("Stream initialized");
    }
  }


  private void Update() {
    if (nextFrame != null) {
```

```
    SendFrame(nextFrame);
    nextFrame = null;
  }
}


private void OnDestroy() {
  foreach (var b in trackedBuffers) {
    if (b != null) {
      b.Close();
    }
  }
}


public void StartStream() {
  if (!cameraInfo.GetState()) {
    errorText.text = "Camera is not enabled";
    return;
  }
  string url = cameraInfo.GetUrl();
  retryCount = 0;
  StopAllCoroutines();
  foreach (var b in trackedBuffers) {
    b.Close();
  }
  worker = new Thread(() => ReadMJPEGStreamWorker(threadID = randu.Next
     (65536), url));
  worker.Start();
}


void ReadMJPEGStreamWorker(int id, string url) {
  var webRequest = WebRequest.Create(url);
  webRequest.Method = "GET";
  List<byte> frameBuffer = new List<byte>();

  int lastByte = 0x00;
  bool addToBuffer = false;

  BufferedStream buffer = null;
  try {
    Stream stream = webRequest.GetResponse().GetResponseStream();
    buffer = new BufferedStream(stream);
    trackedBuffers.Add(buffer);
  }
  catch (System.Exception ex) {
    errorText.text = "Error: " + ex.Message;
    Debug.LogError(ex);
  }
  int newByte;
```

```
while (buffer != null) {
  if (threadID != id) {
    return;
  }
  if (!buffer.CanRead) {
    Debug.LogError("Can't read buffer!");
    break;
  }

  newByte = -1;

  try {
    newByte = buffer.ReadByte();
  }
  catch {
    errorText.text = "Error: " + "Stream read error";
    break;
  }

  if (newByte < 0) {
    continue;
  }

  if (addToBuffer) {
    frameBuffer.Add((byte)newByte);
  }

  if (lastByte == 0xFF) {
    if (!addToBuffer) {
      if (IsStartOfImage(newByte)) {
        addToBuffer = true;
        frameBuffer.Add((byte)lastByte);
        frameBuffer.Add((byte)newByte);
      }
    } else {
      if (newByte == 0xD9) {
        frameBuffer.Add((byte)newByte);
        addToBuffer = false;
        nextFrame = frameBuffer.ToArray();
        frameBuffer.Clear();
      }
    }
  }
  lastByte = newByte;
}

if (retryCount < MAX_RETRIES) {
  retryCount++;
  Debug.LogFormat("[{0}] Retrying Connection {1}...", id, retryCount);
  foreach (var b in trackedBuffers)
    b.Dispose();
```

```
    trackedBuffers.Clear();
    worker = new Thread(() => ReadMJPEGStreamWorker(threadID = randu.Next
        (65536), url));
    worker.Start();
  }
}


bool IsStartOfImage(int command) {
  switch (command) {
    case 0x8D:
      Debug.Log("Command SOI");
      return true;
    case 0xC0:
      Debug.Log("Command SOF0");
      return true;
    case 0xC2:
      Debug.Log("Command SOF2");
      return true;
    case 0xC4:
      Debug.Log("Command DHT");
      break;
    case 0xD8:
      return true;
    case 0xDD:
      Debug.Log("Command DRI");
      break;
    case 0xDA:
      Debug.Log("Command SOS");
      break;
    case 0xFE:
      Debug.Log("Command COM");
      break;
    case 0xD9:
      Debug.Log("Command EOI");
      break;
  }
  return false;
}


void SendFrame(byte[] bytes) {
  Texture2D texture2D = new Texture2D(2, 2);
  texture2D.LoadImage(bytes);
  if (texture2D.width == 2) {
    errorText.text = "Error: " + "Bad image data";
    return;
  }
  Graphics.Blit(texture2D, renderTexture);
  Destroy(texture2D);
}
```

```
}
```

B.8. pan_tilt_controller (header)

```
#pragma once

#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
#include <time.h>
#include <string.h>

#ifdef ESP_PLATFORM
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/ledc.h"
#endif

#define SERVO_MIN_DUTY  ((uint16_t)((0.56666 / 20.0) * (32768.0 - 1.0)))  //
    duty cycle for min grade
#define SERVO_MAX_DUTY  ((uint16_t)((2.24444 / 20.0) * (32768.0 - 1.0)))  //
    duty cycle for max grade  (2^15) duty resolution
#define SERVO_MIDDLE_DUTY  ((uint16_t)((1.36666 / 20.0) * (32768.0 - 1.0)))
    // duty cycle for middle grade  (2^15) duty resolution
#define SERVO_MAX_ANGLE 180
#define SERVO_MIN_ANGLE 0

typedef struct {
  uint8_t is_initialized;
  uint8_t freq_hz;
  uint8_t duty_resolution;
} TimerState;

typedef struct {
  uint8_t is_initialized;
  uint8_t timer;
  uint8_t channel;
  uint8_t gpio_num;
  uint16_t angle;
  uint32_t duty;
} ServoState;

typedef struct {
  ServoState horizontal_servo;
  ServoState vertical_servo;
} PanTiltState;

extern TimerState timers[4];
extern PanTiltState pan_tilt_state;
```

```
void default_pan_tilt_init(void);
void pan_tilt_init(uint8_t timer, uint8_t freq_hz, uint8_t channel_horizontal,
    uint8_t channel_vetical, uint8_t gpio_horizontal, uint8_t gpio_vertical);
void pan_tilt_deinit(void);

void set_horizontal_angle(int16_t angle);
void set_vertical_angle(int16_t angle);
```

B.9. pan_tilt_controller

```
#include "pan_tilt_controller.h"

TimerState timers[4] = {0};
PanTiltState pan_tilt_state = {0};

void init_horizontal_servo(uint8_t channel_selected, uint8_t timer_selected,
   uint8_t gpio_num);
void init_vertical_servo(uint8_t channel_selected, uint8_t timer_selected,
   uint8_t gpio_num);
void init_pwm_timer(uint8_t timer_selected, uint8_t timer_freq_hz);
void configure_timer(uint8_t timer_selected, uint8_t timer_freq_hz);
void configure_servo(uint8_t channel_selected, uint8_t timer_selected, uint8_t
    gpio_num);
void update_timer_state(uint8_t timer_selected, uint8_t timer_freq_hz);
void update_horizontal_servo_state(uint8_t channel_selected, uint8_t
   timer_selected, uint8_t gpio_num);
void update_vertical_servo_state(uint8_t channel_selected, uint8_t
   timer_selected, uint8_t gpio_num);
void servo_deinit(uint8_t channel_selected);
void timer_deinit(uint8_t timer_selected);

void default_pan_tilt_init() {
  pan_tilt_init(0, 50, 0, 1, 17, 16);
  set_horizontal_angle(90);
  set_vertical_angle(0);
}

void pan_tilt_init(uint8_t timer, uint8_t freq_hz, uint8_t channel_horizontal,
    uint8_t channel_vetical, uint8_t gpio_horizontal, uint8_t gpio_vertical) {
  init_pwm_timer(timer, freq_hz);
  init_horizontal_servo(channel_horizontal, timer, gpio_horizontal);
  init_vertical_servo(channel_vetical, timer, gpio_vertical);
}

// Pan tilt desinstalation
void pan_tilt_deinit(void) {
  servo_deinit(pan_tilt_state.horizontal_servo.channel);
  servo_deinit(pan_tilt_state.vertical_servo.channel);
```

```
  timer_deinit(pan_tilt_state.horizontal_servo.timer);
  memset(&pan_tilt_state, 0, sizeof(pan_tilt_state));
}

void set_horizontal_angle(int16_t angle) {
  if (angle < SERVO_MIN_ANGLE) {
    angle = SERVO_MIN_ANGLE;
    return;
  } else if (angle > SERVO_MAX_ANGLE) {
    angle = SERVO_MAX_ANGLE;
    return;
  } else if (!pan_tilt_state.horizontal_servo.is_initialized) {
    printf("Error: the horizontal servomotor isn't initialized.\n");
    return;
  }

  uint32_t duty = ((angle * (SERVO_MAX_DUTY - SERVO_MIN_DUTY)) /
      SERVO_MAX_ANGLE) + SERVO_MIN_DUTY;
  uint8_t channel = pan_tilt_state.horizontal_servo.channel;

  pan_tilt_state.horizontal_servo.angle = angle;
  pan_tilt_state.horizontal_servo.duty = duty;

  ledc_set_duty(LEDC_HIGH_SPEED_MODE, channel, duty);
  ledc_update_duty(LEDC_HIGH_SPEED_MODE, channel);
}

void set_vertical_angle(int16_t angle) {
  if (angle < SERVO_MIN_ANGLE) {
    angle = SERVO_MIN_ANGLE;
    return;
  } else if (angle > SERVO_MAX_ANGLE - 30) {
    angle = SERVO_MAX_ANGLE;
    return;
  } else if (!pan_tilt_state.vertical_servo.is_initialized) {
    printf("Error: the vertical servomotor isn't initialized.\n");
    return;
  }

  uint32_t duty = ((angle * (SERVO_MAX_DUTY - SERVO_MIN_DUTY)) /
      SERVO_MAX_ANGLE) + SERVO_MIN_DUTY;
  uint8_t channel = pan_tilt_state.vertical_servo.channel;

  if (duty < SERVO_MIN_DUTY || duty > SERVO_MAX_DUTY) {
    printf("Error: the duty should be between %u and %u.\n", SERVO_MIN_DUTY,
        SERVO_MAX_DUTY);
    return;
  }

  pan_tilt_state.vertical_servo.angle = angle;
  pan_tilt_state.vertical_servo.duty = duty;
```

```
  ledc_set_duty(LEDC_HIGH_SPEED_MODE, channel, duty);
  ledc_update_duty(LEDC_HIGH_SPEED_MODE, channel);
}

void init_horizontal_servo(uint8_t channel_selected, uint8_t timer_selected,
    uint8_t gpio_num) {
  if (channel_selected > 7) {
    printf("Error: The channel selected must be between 0 and 7.\n");
    return;
  }

  if (timers[timer_selected].is_initialized) {
    configure_servo(channel_selected, timer_selected, gpio_num);
    update_horizontal_servo_state(channel_selected, timer_selected, gpio_num);
  }
}

void init_vertical_servo(uint8_t channel_selected, uint8_t timer_selected,
    uint8_t gpio_num) {
  if (channel_selected > 7) {
    printf("Error: The channel selected must be between 0 and 7.\n");
    return;
  }

  if (timers[timer_selected].is_initialized) {
    configure_servo(channel_selected, timer_selected, gpio_num);
    update_vertical_servo_state(channel_selected, timer_selected, gpio_num);
  }
}

void init_pwm_timer(uint8_t timer_selected, uint8_t timer_freq_hz) {
  if (timer_selected > 3) {
    printf("Error: The timer selected must be between 0 and 3.\n");
    return;
  }
  if (!timers[timer_selected].is_initialized) {
    configure_timer(timer_selected, timer_freq_hz);
    update_timer_state(timer_selected, timer_freq_hz);
  } else {
    printf("Error: The timer %d is already initialized at a frecuency of %d Hz
        .\n", timer_selected, timers[timer_selected].freq_hz);
    return;
  }
}

void configure_timer(uint8_t timer_selected, uint8_t timer_freq_hz) {
  ledc_timer_config_t timer_conf;
  timer_conf.duty_resolution = LEDC_TIMER_15_BIT; // resolution of PWM duty.
      Max duty is 32767
  timer_conf.freq_hz = timer_freq_hz;
```

```
  timer_conf.speed_mode = LEDC_HIGH_SPEED_MODE;
  timer_conf.timer_num = timer_selected;
  ledc_timer_config(&timer_conf);
}


void configure_servo(uint8_t channel_selected, uint8_t timer_selected, uint8_t
    gpio_num) {
  ledc_channel_config_t ledc_conf;
  ledc_conf.channel = channel_selected;
  ledc_conf.duty = SERVO_MIDDLE_DUTY;
  ledc_conf.gpio_num = gpio_num;
  ledc_conf.intr_type = LEDC_INTR_DISABLE;
  ledc_conf.speed_mode = LEDC_HIGH_SPEED_MODE;
  ledc_conf.timer_sel = timer_selected;
  ledc_channel_config(&ledc_conf);
}


void servo_deinit(uint8_t channel_selected) {
  ledc_stop(LEDC_HIGH_SPEED_MODE, channel_selected, 0);
}

void timer_deinit(uint8_t timer_selected) {
  ledc_timer_rst(LEDC_HIGH_SPEED_MODE, timer_selected);
}


void update_timer_state(uint8_t timer_selected, uint8_t timer_freq_hz) {
  timers[timer_selected].is_initialized = 1;
  timers[timer_selected].freq_hz = timer_freq_hz;
}


void update_horizontal_servo_state(uint8_t channel_selected, uint8_t
   timer_selected, uint8_t gpio_num) {
  pan_tilt_state.horizontal_servo.is_initialized = 1;
  pan_tilt_state.horizontal_servo.timer = timer_selected;
  pan_tilt_state.horizontal_servo.channel = channel_selected;
  pan_tilt_state.horizontal_servo.gpio_num = gpio_num;
  pan_tilt_state.horizontal_servo.angle = (SERVO_MAX_ANGLE + SERVO_MIN_ANGLE)
     / 2;
  pan_tilt_state.horizontal_servo.duty = SERVO_MIDDLE_DUTY;
}


void update_vertical_servo_state(uint8_t channel_selected, uint8_t
   timer_selected, uint8_t gpio_num) {
  pan_tilt_state.vertical_servo.is_initialized = 1;
  pan_tilt_state.vertical_servo.timer = timer_selected;
  pan_tilt_state.vertical_servo.channel = channel_selected;
```

```
  pan_tilt_state.vertical_servo.gpio_num = gpio_num;
  pan_tilt_state.vertical_servo.angle = (SERVO_MAX_ANGLE + SERVO_MIN_ANGLE) /
      2;
  pan_tilt_state.vertical_servo.duty = SERVO_MIDDLE_DUTY;
}
```

B.10. Tests del Pan-Tilt

```
#include "unity.h"

#include "pan_tilt_controller.h"

void setUp(void) {
  pan_tilt_init(0, 50, 0, 1, 17, 16);
}

void tearDown(void) {
  pan_tilt_deinit();
}

TEST_CASE("Test max duty and min duty", "[pan_tilt_controller]") {
  TEST_ASSERT_EQUAL(1638, SERVO_MIN_DUTY);
  TEST_ASSERT_EQUAL(3276, SERVO_MAX_DUTY);
}

TEST_CASE("Test pan-tilt initializated", "[pan_tilt_controller]") {
  int8_t timer_selected_state = timers[0].is_initialized;
  int8_t horizontal_servo_state = pan_tilt_state.horizontal_servo.
      is_initialized;
  int8_t vertical_servo_state = pan_tilt_state.vertical_servo.is_initialized;


  TEST_ASSERT_EQUAL(1, timer_selected_state * horizontal_servo_state *
      vertical_servo_state);
}

TEST_CASE("Test initial horizontal angle", "[pan_tilt_controller]") {
  set_horizontal_angle(0);

  int16_t horizontal_angle_state = pan_tilt_state.horizontal_servo.angle;
  int32_t horizontal_duty_state = pan_tilt_state.horizontal_servo.duty;

  TEST_ASSERT_EQUAL(0, horizontal_angle_state);
  TEST_ASSERT_EQUAL(SERVO_MIN_DUTY, horizontal_duty_state);
}

TEST_CASE("Test horizontal angle", "[pan_tilt_controller]") {

  set_horizontal_angle(90);
```

```
  int16_t horizontal_angle_state = pan_tilt_state.horizontal_servo.angle;
  int32_t horizontal_duty_state = pan_tilt_state.horizontal_servo.duty;

  TEST_ASSERT_EQUAL(90, horizontal_angle_state);
  TEST_ASSERT_EQUAL(SERVO_MIN_DUTY + 819, horizontal_duty_state);
}

TEST_CASE("Test initial vertical angle", "[pan_tilt_controller]") {

  set_vertical_angle(0);

  int16_t vertical_angle_state = pan_tilt_state.vertical_servo.angle;
  int32_t vertical_duty_state = pan_tilt_state.vertical_servo.duty;

  TEST_ASSERT_EQUAL(0, vertical_angle_state);
  TEST_ASSERT_EQUAL(SERVO_MIN_DUTY, vertical_duty_state);
}

TEST_CASE("Test vertical angle", "[pan_tilt_controller]") {

  set_vertical_angle(15);

  int16_t vertical_angle_state = pan_tilt_state.vertical_servo.angle;
  int32_t vertical_duty_state = pan_tilt_state.vertical_servo.duty;

  TEST_ASSERT_EQUAL(15, vertical_angle_state);
  TEST_ASSERT_EQUAL(SERVO_MIN_DUTY + 136, vertical_duty_state);
}

TEST_CASE("Test increment movement", "[pan_tilt_controller]") {
  pan_tilt_deinit();
  default_pan_tilt_init();
  set_vertical_angle(pan_tilt_state.vertical_servo.angle - 10); // Up

  int16_t vertical_angle_state = pan_tilt_state.vertical_servo.angle;

  TEST_ASSERT_EQUAL(80, vertical_angle_state);
}
```