Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBAJO FINAL DE GRADO

TÍTULO: Implementación del Circo de las Imágenes y otras Contribuciones al 'Tello Engineering Ecosystem'

TITULACIÓN: Grado en Ingeniería Aeroespacial.

AUTOR: Anna Iturralde Aguiló

DIRECTOR: Miguel Valero García

FECHA: 07 febrero 2024

TÍTULO: Implementación del Circo de las Imágenes y otras Contribuciones al 'Tello Engineering Ecosystem'

TITULACIÓN: Grado en Ingeniería Aeroespacial.

AUTOR: Anna Iturralde Aguiló

DIRECTOR: Miguel Valero García

FECHA: 07 febrero 2024

Resumen

En un mundo donde la tecnología continúa avanzando y transformando diversos sectores, la presencia de los drones se ha vuelto esencial para optimizar y revolucionar la industria. Sus características los hacen valiosos e imprescindibles para el desarrollo industrial.

A raíz de una iniciativa de la Escuela de Ingeniería de Telecomunicación y Aeroespacial de Castelldefels (EETAC), surge la creación de un ecosistema de drones que permite a estudiantes y profesores, contribuir en su desarrollo, buscando proyectar una imagen positiva de esta tecnología en la sociedad. Mediante aplicaciones tanto formativas como recreativas se pretende organizar exhibiciones y demostraciones a un público diverso.

El proyecto se enfoca en el desarrollo de un software para el control del dron DJI Tello Robomaster, con el objetivo de mejorar y expandir sus capacidades. Este enfoque se alinea con la misión de la escuela de promover la innovación y la integración de tecnologías emergentes en el ámbito educativo. Nace así la idea de este proyecto, dividido en dos secciones: la parte teórica y la parte práctica.

En la parte teórica, se describen todos los aspectos y conceptos básicos para entender con qué código se trabajará, las tecnologías utilizadas, cuáles son los objetivos, entre otros elementos fundamentales.

En la parte práctica, se destallan todas las implementaciones realizadas y los objetivos logrados, con la perspectiva de facilitar futuras contribuciones de estudiantes. Principalmente, se va a trabajar tres tipos diferentes de imágenes: fotos, panorámicas y vídeos, en la implementación de un nuevo circo.

TÍTULO: Implementación del Circo de las Imágenes y otras Contribuciones al 'Tello Engineering Ecosystem'

TITULACIÓN: Grado en Ingeniería Aeroespacial.

AUTOR: Anna Iturralde Aguiló

DIRECTOR: Miguel Valero García

FECHA: 07 febrero 2024

Overview

In a world where technology continues to advance and transform diverse sectors, the presence of drones has become essential for optimizing and revolutionizing the industry. Their characteristics make them valuable and indispensable for industrial development.

Following an initiative by the Castelldefels School of Telecommunications and Aerospace Engineering (EETAC), the creation of a drone ecosystem allows students and teachers to contribute to its development, aiming to project a positive image of this technology in society. Both educational and recreational applications seek to organize exhibitions and demonstrations to a diverse audience.

The project focuses on the development of software to control the DJI Tello Robomaster drone, with the goal of enhancing and expanding its capabilities. This approach aligns with the school's mission to promote innovation and the integration of emerging technologies in the educational field. Thus, the idea of this project was born, divided into two sections: the theoretical part and the practical part.

In the theoretical part, all the fundamental concepts are detailed to understand which code will be used, the technologies employed, and the objectives, among other elemental aspects.

In the practical part, all the implementations made and achieved objectives are highlighted, with the perspective of facilitating future contributions of students. Mainly, the focus will be on working on three different types of images: photos, panoramics and videos, in the implementation of a new circus.

Con este trabajo acabo una etapa. Una etapa dura e intensa, pero gratificante y feliz. Me llevo personas, amigos, referentes, consejos y mucho aprendizaje. Este logro no solo representa el fin de un esfuerzo académico, sino también el inicio de un camino de aprendizaje y crecimiento personal.

Agradezco profundamente a mi familia por su apoyo incondicional y motivación constante. A cada uno de mis profesores, gracias por creer en la enseñanza. A Miguel Valero, por brindarme esta oportunidad. I a tú, Adrià, gràcies per estar sempre al meu costat.

ÍNDICE

INTRODUCCIÓN

En los últimos años, la figura de los drones se ha convertido en un pilar fundamental para las industrias, proporcionando grandes ventajas para optimizar y revolucionar el sector. Los drones se están volviendo cada vez más importantes en el mundo de hoy en día.

Hasta hace poco, este tipo de tecnología era desconocida para la sociedad y estaba destinada en su mayor medida a casos militares. En sus inicios, la introducción de los drones en la sociedad no fue muy bien recibida.

Los drones, también conocidos como vehículos aéreos no tripulados, son vehículos sin tripulación capaces de mantenerse de manera estable, segura y autónoma a un nivel de vuelo controlado. Su minuciosidad y tamaño les permite ser más versátiles, maniobrables, económicos y fáciles de transportar.

La revolución de los drones está transformando diversos sectores industriales y crece exponencialmente. Cierto es que, aunque aún no ha habido una inserción masiva en la población, cada vez son más las empresas que los utilizan con fines comerciales.

Actualmente, gracias a la incorporación de GPS, cámaras y sensores, los drones se utilizan en diversas industrias. Esto incluye servicios de emergencia, equipos de rescate y búsqueda de personas, protección y seguridad para la vigilancia en multitudes, agricultura para reducir costos y pérdidas, control de cosechas, así como en el sector sanitario para el traslado de órganos, entre otros. También, se emplean en el ámbito del entretenimiento, la posibilidad de transmitir en vivo una imagen, seguir objetos o maniobrar en espacios reducidos, hacen que sean perfectos para crear experiencias únicas para el público.

Por consiguiente, nace la idea de crear un ecosistema de drones que permita desarrollar diversas aplicaciones tanto a nivel formativo como recreativo. El profesorado y alumnado de la Escuela de Ingeniería EETAC quiere proyectar una idea positiva de los drones sobre la sociedad, enseñando, a través de la diversión, múltiples funcionalidades que los drones permiten hacer y las diversas tecnologías implementadas en el ecosistema.

La escuela, comprometida por esta iniciativa, creó el ‘DroneLab’, un espacio dedicado a la investigación en el ámbito de los drones. En este espacio se llevan a cabo una amplia gama de pruebas, vuelos y entrenamientos, así como espectáculos, actividades y exhibiciones con diferentes tipos de drones dirigidos a todo tipo de espectadores.

Esta innovación atrae mucho al público que reciben con entusiasmo estas actividades, diseñadas para ofrecer conferencias y sesiones interactivas que permiten al público adentrarse con más detalle en el mundo de los drones. Además, brinda la oportunidad de presenciar demostraciones tanto en espacios

interiores, por ejemplo, dentro de una sala, así como en el exterior (DroneLab), donde pueden observar y participar en los espectáculos.

El excepcional tamaño del dron Tello le permite realizar sus exhibiciones en tamaños reducidos y limitados, garantizando seguridad a los invitados. Además, esto permite tener mayor versatilidad en sus eventos, llegar a lugares difíciles de acceder, reducir el peligro, adaptabilidad durante las demostraciones, entre otros.

Este proyecto representa una colaboración conjunta dirigida hacia el desarrollo de un *software* destinado al control del dron DJI Tello. El propósito principal es incorporar diversas mejoras a un código base existente, lo cual supone un esfuerzo de adaptación, y la implementación de nuevas funcionalidades para alcanzar los objetivos propuestos. Este enfoque busca, también, establecer una base sólida para que futuros estudiantes tengan la oportunidad de contribuir y expandir este ecosistema en constante evolución. Por ello se destaca la importancia de documentar exhaustivamente todo el trabajo realizado, brindando así a futuros estudiantes una referencia clara y completa sobre la evolución y el contexto del proyecto.

La memoria está organizada en varios capítulos divididos en tres secciones fundamentales: la teórica, la práctica y los resultados. Los primeros tres capítulos; titulados ‘Tello Robomaster TT’, ‘Tello Engineering Ecosystem’ y ‘Objetivos y plan de trabajo’, conforman la parte teórica. Los tres capítulos siguientes: ‘Primeras Contribuciones’, ‘Integración de la detección con la cámara del móvil’ y ‘Circo de las Imágenes’ forman la parte práctica. El resto de los capítulos: ‘Pruebas y validaciones’ e ‘Integración en el ecosistema’ forman la parte de resultados.

La parte teórica proporciona un detallado análisis de todos los conceptos necesarios para entender las herramientas empleadas en este proyecto. Esto abarca desde las características del dron, descritas en el primer capítulo, hasta las especificaciones del *software* base y las tecnologías empleadas en el desarrollo de este proyecto, entre otros aspectos relevantes, definidas en el segundo capítulo. En el tercer capítulo, se describe la estructura organizativa del proyecto, sus objetivos y las tareas realizadas.

En la sección práctica, se han detallado todas las implementaciones realizadas a lo largo del proyecto, incluyendo mejoras, nuevas aplicaciones y la estructura de los códigos implementados. En el cuarto capítulo se detallan los primeros cambios realizados en el código base. En el capítulo quinto, se detalla toda la implementación de la detección con la cámara del móvil. Para acabar, en el capítulo sexto, la implementación de un nuevo circo llamado ‘Circo de las Imágenes’, que trabaja con imágenes, vídeos y panorámicas.

Para finalizar, se encuentran: el capítulo séptimo en el que se validan e interpretan los resultados obtenidos; y el último capítulo en el que se documenta todo el proyecto para futuros estudiantes.

CAPÍTULO 1. TELLO ROBOMASTER TT

En este capítulo, se detallará la definición del dron Tello, sus características y elementos, algunos modos de vuelo que tiene implementados, así como la librería de Python que se utilizará para establecer una comunicación efectiva entre el dron y el *software*. Todo esto proporcionará una base sólida para conocer con qué se trabajará en este proyecto.

1.1. Introducción

El 'DJI Tello Talent Robomaster TT' es un dron desarrollado por 'DJI Education' con fines educativos y formativos. Esta herramienta tiene como objetivo introducir a los estudiantes en del mundo tecnológico, la inteligencia artificial y la robótica [1].

El 'Robomaster TT', también conocido como 'el pequeño dron con grandes sueños', es un cuadricóptero líder en la industria, que ofrece un rendimiento de vuelo potente en un tamaño compacto y utiliza algoritmos de control de vuelo de vanguardia de DJI para garantizar la seguridad y la estabilidad del vuelo [2].

Tiene las bases del 'Tello EDU', que permite el control de múltiples drones y aplicaciones de inteligencia artificial, como la percepción ambiental, el reconocimiento facial y de gestos, el seguimiento automático, entre otros. Sin embargo, el 'Tello Robomaster' presenta algunas ventajas distintivas respecto al 'Tello EDU'.

La diferencia más visual y significativa con respecto al 'Tello EDU' es el llamado *kit* de expansión, que incluye un conjunto de elementos que brindan al usuario una mayor diversidad en la implementación de la inteligencia artificial. Este *kit* está compuesto por un *display* con luces LED programables que forman una matriz de puntos, la integración de un módulo de detección de distancia, un controlador de código abierto y una placa de ampliación.

1.2. Características

El 'Robomaster Tello' se compone de dos partes: la aeronave y el *kit* expansión, en comparación con el 'Tello EDU' como se ha comentado anteriormente.

La aeronave es un 'Tello' del modelo TLW004, que pesa 87 gramos (incluyendo los protectores de las hélices). Su velocidad máxima es de 28.8 kilómetros por hora y tiene una autonomía de 13 minutos sin el *kit* de expansión montado, y de 8 minutos con el *kit* de expansión incorporado, a condiciones normales y sin vientos mayores a 15 kilómetros por hora.

Opera a una temperatura entre 0 ºC y 40 ºC, y en una frecuencia de entre 2.4 a 2.4835 GHz. Tiene un límite de vuelo de una altura no superior a los 10 metros.

La parte que conforma la aeronave está formada por los siguientes elementos:

1. **Cuatro motores:** Los motores son de metal con un voltaje operativo de 1.5-4.2 V y una velocidad de giro con carga de 205006 rpm. En la parte inferior se conectan con los soportes.

2. **Cuatro hélices:** Las hélices, del modelo 3044P, están diseñadas para rotar en diferentes direcciones. Contienen unas marcas que indican su tipo, el sentido de giro y los motores específicos en los que deben colocarse en el tren de aterrizaje.
   La nomenclatura de las hélices '3044P' hace referencia a las dimensiones de estas. El 30 es la longitud (*inch*) de la hélice, es decir, el diámetro de un disco que forma la hélice cuando gira. El 44 hace referencia al ángulo de cabeceo (*Pitch*), definido como la distancia que un puntal sería arrastrado hacia adelante a través de un sólido en una sola revolución completa.

3. **Protectores de las hélices:** Ayudan a disminuir el riesgo de daños a personas u objetos producidas por colisiones o fallos del sistema.

4. **Baterías:** Son de tipo LiPo (Polímero de Iones de Litio), con un voltaje de 3,8 V y una capacidad de 110 mmAh con protección de carga y descarga. Tienen una potencia máxima de 10 W. El dron tiene un compartimento para colocar las baterías.

5. **Cámara:** Contiene una cámara que es capaz de capturar fotos de 5MP, con un formato de vídeo MP4, y transmitir imágenes y vídeos en HD: 1280 × 720 30p.

6. **Botón de encendido:** En el lateral derecho está el botón de encender y apagar del dron.

7. **Indicador de estado de la aeronave:** Es un indicador LED que está al lado de la cámara en la parte delantera del dron, y que informa de los modos en los que se encuentra el dron. Por ejemplo, si parpadea periódicamente verde dos veces, indica que el sistema de posicionamiento visual está disponible

8. **Sistema de posicionamiento visual:** Se encuentra en la parte inferior del dron y ayuda al dron a mantenerse en su posición actual. Lo estabiliza y le permite volar con mayor precisión, minuciosidad y equilibrio. Está formado por una cámara y un módulo de detección de infrarrojos 3D, formado por sensores que capturan imágenes del entorno. El rendimiento del sistema de posicionamiento visual depende del tipo de superficie que esté sobrevolando el dron en cierto momento, ya que toma las referencias de equilibrio de la superficie volada, corrigiendo así las desviaciones y evitando posibles colisiones. El rango

que le permite trabajar de manera óptima y efectiva está entre 0.3 y 6 metros.

Por otro lado, el *kit* de expansión permite mejorar el dron 'Tello EDU' y proporciona diversas ventajas. Por ejemplo, admite una comunicación de doble banda de 2,4 GHz a 5,8 GHz, proporcionando más estabilidad y mejor adaptabilidad a su entorno. También, es más interactivo gracias a la pantalla de leds de la matriz de puntos programable y admite programación sin conexión, entre otros.

Este *kit* de expansión está formado por:

9. **Pantalla de una matriz de puntos y módulo de detección de distancia:** El *display* consta de una matriz de 8x8 de colores rojo y azul que permite mostrar ciertos mensajes amigables para el usuario. A su vez, el sensor de medición de distancia es de tipo TOF (Time of Flight) con una distancia máxima de 1.2 metros.

10. **Controlador de código abierto:** Es un dispositivo que combina un módulo *wifi* de doble frecuencia de 2,4/5,8 GHz, un módulo Bluetooth y una plataforma de código abierto Arduino. Admite trabajar con SDK (Software Development Kit), Arduino, Scratch y MicroPython. Se conecta al dron desde un puerto micro USB. El Robomaster Tello tiene el modelo RMTTOC y contiene una MCU de tipo ESP32.

11. **Placa de expansión:** Consiste en un puerto de extensión de 14 pines (dos filas de 7 pines), junto con dos posiciones reservadas para una alimentación de 3,3/5 V. Admite protocolos de programación I2C, SPI, UART y GPIO.

En la **Fig. 1.1**, recogida del manual de DJI, se pueden observar con mayor detalle las partes del dron Tello y sus posiciones [3].

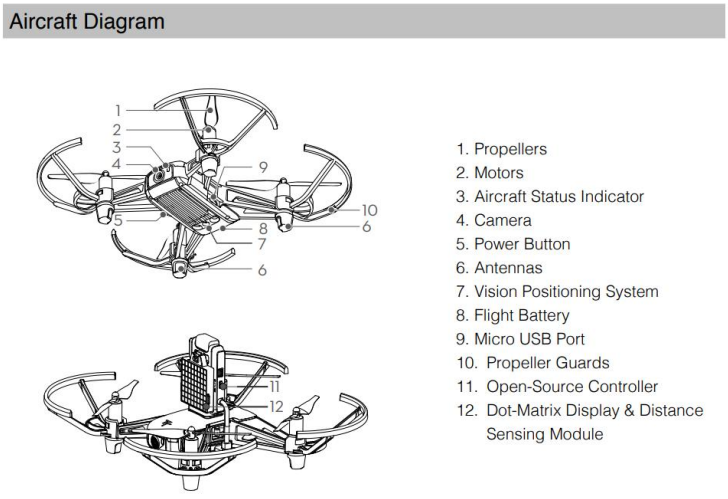

Fig. 1.1. Diagrama de elementos del dron Tello según el manual de DJI.

1.3. Modos de vuelo y limitaciones

El 'Robomaster TT' presenta diversas limitaciones en cuanto a altura, velocidad, autonomía, entre otros, interesantes a mencionar.

Como se ha mencionado previamente, el dron no debe volar por encima de los 10 metros (30 pies) para un rendimiento óptimo, aunque tiene una altitud máxima de 30 metros. Su alcance de control, o rango de vuelo, es de 100 metros, y la duración de la batería es de aproximadamente 10 a 13 minutos.

Además, tiene una velocidad máxima de 28.8 kilómetros por hora y dos tipos de velocidades de vuelo que se pueden cambiar manualmente. La configuración predeterminada incluye la velocidad lenta, que no permite al dron volar a una velocidad superior a los 10.8 kilómetros por hora ni tener un ángulo máximo de altitud de vuelo de 9 grados. Asimismo, cuenta con la velocidad rápida, que impide que el dron vuele a ángulo máximo de altitud de 25 grados ni supere la velocidad máxima de 28.8 kilómetros por hora.

A su vez, el dron Tello no está equipado con GPS, lo que limita su capacidad para realizar vuelos autónomos o modelos con trayectorias predefinidas.

Por otro lado, el 'Robomaster Tello' tiene distintos modos de vuelo que permiten a los usuarios experimentar diferentes funciones de vuelo, que se pueden activar tanto desde la aplicación como mediante botones físicos.

En primer lugar, existe el modo *Attitude* que se activa automáticamente cuando el sistema de posicionamiento de visión no está disponible. En este modo, el dron se ve afectado por el entorno y no vuela de manera estable ni puede posicionarse correctamente. En presencia de factores externos como ráfagas de viento o lluvia, puede desestabilizarse, por lo que se recomienda aterrizar lo antes posible cuando se activa este modo.

Por otro lado, existe el modo *Bounce,* en el que el dron vuela hacia arriba y hacia abajo entre los 1.6 y 3.9 pies sobre una superficie plana. Si detecta objetos debajo de él, aumentará su altitud y viceversa, permitiendo un vuelo con efecto rebote.

Además, se encuentra el modo 8D *Flips*, que permite al dron girar automáticamente en ocho direcciones distintas.

El modo *Throw and Go* permite al usuario lazar el dron al aire, y este despegar de manera segura. El Tello detecta el lanzamiento y se estabiliza, permitiendo al usuario tomar el control del dron una vez se libera.

Por último, se encuentran diferentes modos de grabación. El modo *EZ Shots* permite al dron grabar un vídeo mientras realiza una rotación de 360º. Existen también modos parecidos, como '*Using Circle mode'*, donde el dron graba un vídeo haciendo un círculo alrededor de un objeto de referencia, lo que le permite, por ejemplo, grabar de manera panorámica, y el modo *'Using Up and Away'*, que permite grabar un vídeo mientras el dron asciende y retrocede.

1.4. DJI TELLO API

El dron DJI Tello, con el que se trabajará en este proyecto, interactuará con la API DJITelloPy (siglas en inglés Application Programming Interface). Una API es un conjunto de funciones, protocolos y definiciones que posibilitan a dos componentes de software comunicarse entre sí y permite el acceso a información de dichos sistemas operativos. La API DJITello facilita a los desarrolladores controlar y programar el dron Tello para poder realizar todas las funcionalidades que se le quieran dar.

La API DJITello contienen librerías de Python (djitellopy/tellopy) que utilizan el SDK oficial de Tello y de Tello EDU, y posibilitan la comunicación con el dron, permitiendo el envío de las operaciones necesarias en cada momento y el control del Tello [4].

En este proyecto se va a utilizar la librería de Python 'djitellopy', que es una biblioteca de código abierto que permite la interacción con el Tello mediante conexión *wifi*. También, habilita la implementación de todos los comandos del dron Tello, recibir comandos del dron, controlar un enjambre de drones, recuperar una secuencia de vídeo o fotos, crear misiones para los drones, el envío de comandos al dron, etc.

Su instalación es muy sencilla y se consigue mediante el comando: '`pip install djitellopy`'. Esta librería admite muchas funcionalidades que permiten vuelos complejos y versátiles. El siguiente ejemplo, muy sencillo, proporciona una visión más clara del funcionamiento:

```
>> tello = Tello()
>> tello.connect()
>> tello.takeoff()
>> tello.land()
>> tello.end()
```

Este código permite despegar el dron. Primero se inicializa una instancia de Tello y se realiza la conexión con el dron. A continuación, se le envía el comando que se requiera en cierto momento, en este caso es despegar y aterrizar. Por último, se cierra la conexión con el dron.

Paralelamente, existen diferentes tipos de comandos, que irán apareciendo en el código y permiten mostrar por la pantalla del *display* un mensaje concreto.

El *display* está formado por una matriz 8x8, como se describió anteriormente, y contiene unos LEDs que permite jugar con las luces y los colores, y mostrar mensajes.

A continuación, se pueden observar dos ejemplos de los comandos utilizados:

```
>> arriba = "EXT mled g
000bb00000bbbb000bbbbbb0bb0bb0bbb00bb00b000bb000000bb000000
bb000"
```

```
>> ole = "EXT mled l r 2.5 OLE...."
```

El comando de 'arriba', se forma a través de una flecha azul mirando hacia arriba. Los ceros significan que no se enciende el led y las 'b' significan que los LED se encienden de color azul. El primer cero se encuentra en la fila cero y columna cero de la matriz, el segundo se encuentra en la fila cero y la columna uno, y así consecutivamente. La 'g' indica que el mensaje es fijo. Estas especificaciones consiguen mostrar la flecha azul [5].

El comando de 'ole' es un mensaje con movimiento. La 'l' indica la dirección del movimiento, en este caso izquierda (*left*). Existen también las posibilidades de moverlo hacia la derecha, arriba o abajo, siendo r,u,d las indicaciones. La 'r', indica el color rojo. A continuación, se especifica la velocidad de los fotogramas, en este caso 2.5 Hz. Por último, se indica el mensaje que aparecerá por pantalla, en este caso es 'OLE….', que irá apareciendo de derecha a izquierda por pantalla.

En la **Fig. 1.2**, se observa el *display* donde es posible mostrar estos mensajes, junto con el mensaje por defecto del Tello al crear la conexión que son las 't' de Tello, la matriz 8x8 y el juego con los colores, que se ha explicado en los párrafos previos.

Fig. 1.2. *Display* del *Kit* de Expansión del Tello predeterminado.

CAPÍTULO 2. TELLO ENGINEERING ECOSYSTEM

En esta sección se específica con qué *software* se trabajará y el conjunto de algoritmos que permitirán tener la base en la que se podrán añadir todas las contribuciones que englobarán este proyecto, así como la estructura de este código y el conjunto de tecnologías que se utilizan para poder llevar a cabo todas las funcionalidades y aplicaciones.

2.1. Introducción

El 'Tello Engineering Ecosystem' es una herramienta de *software* que permite controlar el funcionamiento de uno o varios drones Tello, proporcionando diferentes aplicaciones y actividades, como guiar al dron con poses del cuerpo, trabajar con un enjambre de drones, o jugar con la detección de colores.

El 'Tello Engineering Ecosystem*'* nace del deseo de una escuela de introducir al mundo de hoy en día una visión más innovadora y vanguardista sobre los drones. El objetivo proyectar una imagen positiva sobre los drones para poder enseñar su funcionamiento y posibles aplicaciones de manera formativa y divertida. Por eso, se creó esta plataforma para controlar drones Tello, donde tanto alumnos como profesores puedan contribuir al futuro de esta causa.

En este proyecto se trabajará con base a este código previamente creado y estructurado, al cual se le irán añadiendo nuevas funcionalidades y mejoras, y así poder lograr los objetivos de esta iniciativa.

El 'Tello Engineering Ecosystem' se puede encontrar en el siguiente repositorio de GitHub: https://github.com/dronsEETAC/TelloEngineeringEcosystem.git [6].

2.2. Estructura

El 'Tello Drone Engineering Ecosystem' consta de varios bloques que se conectan a un bróker local MQTT (Message Queuing Telemetry Transport). Se identifican tres tipos de módulos diferentes entre sí, que permiten la comunicación con el dron Tello: las aplicaciones de escritorio, las aplicaciones web App y aquellos que permiten el control de un enjambre de drones Tello.

La conexión de los elementos con el bróker se consigue mediante protocolos de publicación/suscripción MQTT, que permiten de manera eficiente la transmisión de mensajes de un sistema a otro. Este bróker se ejecuta en Internet. Todas las conexiones entre módulos y el dron se logran mediante conexiones *wifi*.

Las aplicaciones de escritorio están programadas en Python y TKinter, y son las siguientes:

- **Tello Drone Circus:** es una aplicación de escritorio que permite interaccionar tanto con el usuario como con el dron. Aquí se definen varios escenarios, como la detección de poses y caras para guiar el dron, jugar con los colores, hacer que el dron te siga, etc.
  El 'Circo de Drones' está dividido en dos circos; el 'Circo de las Poses' y el 'Circo de los 'Colores'.

- **Tello Drone Server:** es un servidor que posibilita la comunicación del dron y la aplicación web App. Al ejecutarse, se encuentran dos opciones: 'one dron' y 'play', que permiten inicializar el servidor para diferentes configuraciones de juego.

Por otro lado, se encuentra la aplicación web App, programada en Ionic y Vue:

- **Tello Web App:** es una aplicación web App que permite la interacción con el dron, guiarlo, enviarle comandos, tomar fotografías, etc. a través de la implementación de botones. Se comunica con el servidor para que le lleguen correctamente las instrucciones al dron.

Por último, existen dos módulos relacionados con el enjambre de drones Tello, llamados 'Swarm Web App' y 'Tello Swarm Server', que son la aplicación web App y su correspondiente servidor. En este proyecto no se trabajará con enjambre de drones.

La **Fig. 2.1**, facilita tener una visión óptima sobre la estructura del ecosistema, dónde se pueden ver de manera clara los diferentes módulos descritos anteriormente. El bróker marcado de color azul, el software del enjambre de drones en color verde y todo lo que engloba este proyecto de color amarillo.

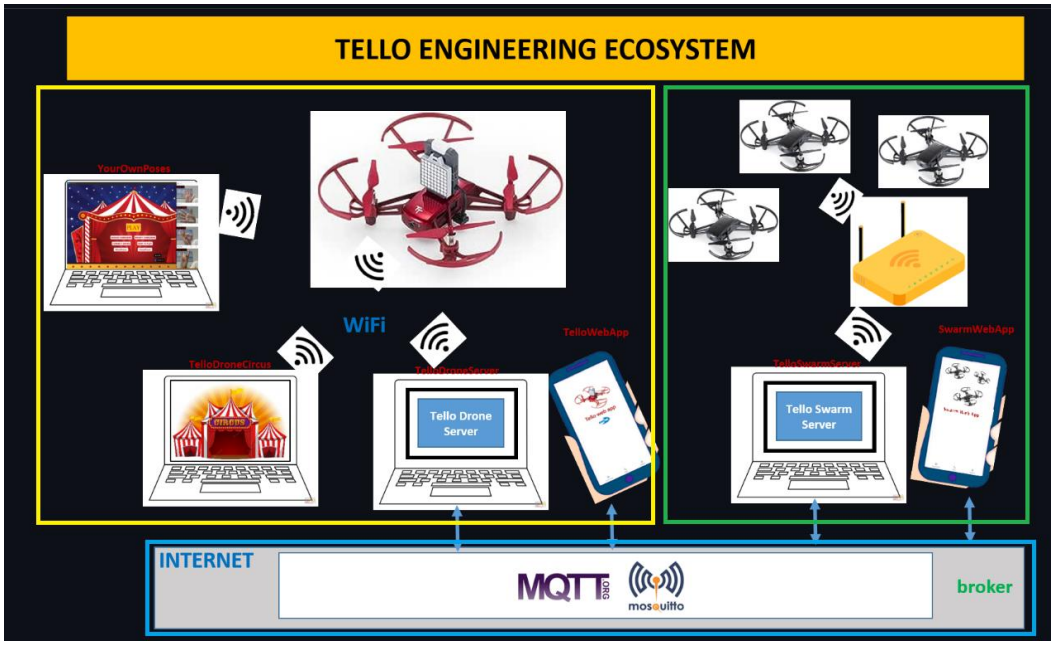

Fig. 2.1. Disposición del Tello Engineering Ecosystem

2.3. Módulos

Como se mencionó previamente, el *software* base utilizado consta de tres componentes distintos. Incluye una aplicación de escritorio llamada 'Tello Drone Circus', una aplicación web App compuesta por el 'Tello Drone Server' y 'Tello Web App', y una implementación para el enjambre de drones, que no se empleará en este proyecto en específico."

La aplicación de escritorio *'*Tello Drone Circus', programada en Python, está formada fundamentalmente por dos circos, el Circo de los Colores y el Circo de las Poses.

El panel central del 'Circo de las Poses' alberga una serie de botones que facilitan la configuración del escenario de trabajo. Entre ellos se encuentra, en primer lugar, un botón que ofrece valores predefinidos para la configuración del dron, como la anchura, altura, etc. También, hay un botón dedicado a establecer la conexión con el dron. Por último, destacan dos botones importantes en este panel: el botón 'Crea tus poses', que inicia la interacción con el dron, y 'Empezar espectáculo', para dar comienzo al espectáculo.

En la **Fig. 2.2**, se observa la disposición de los botones iniciales de configuración descritos anteriormente.

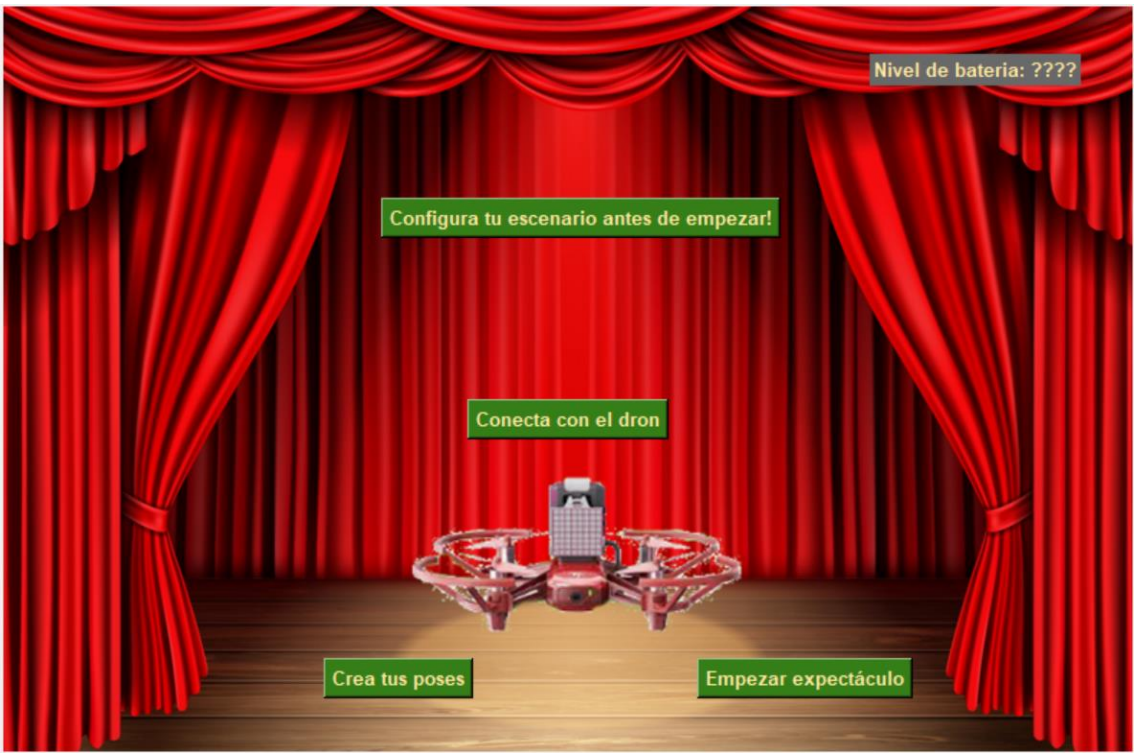

Fig. 2.2. Disposición de los elementos del panel principal del Circo de las Poses.

El botón 'Crea tus poses' permite la creación de poses de tres maneras diferentes: utilizando la mano, medio cuerpo o el cuerpo entero. En la **Fig. 2.3**, se muestra la disposición de este panel, donde se observan claramente las tres opciones mencionadas previamente en la parte superior de este.

En la parte izquierda del panel, se encuentra un conjunto de botones que facilitan la captura de estas imágenes. Primero, se puede iniciar la detección con 'Start Catching', luego es posible capturar una imagen en cualquier momento con el botón 'Catch'. Una vez se captura una imagen, esta se mostrará en los *canvas* de la parte derecha del menú de comandos. También se ofrece la posibilidad de eliminar estas imágenes capturadas de los *canvas* con la opción de 'Delete'. Al finalizar la detección y captura de imágenes, se pueden guardar las poses para utilizarlas en cualquier momento en la ejecución posterior del espectáculo. Además, existe el botón de 'Detect' que permite comprobar el correcto funcionamiento y la precisión en la detección de las poses capturadas y establecidas en los c*anvas*.

Por último, se observan tres *radiobuttons*; 'difficult', 'medium' y 'easy', que permiten gestionar el radio de detección de los puntos de interés, que se explicarán en capítulos siguientes.

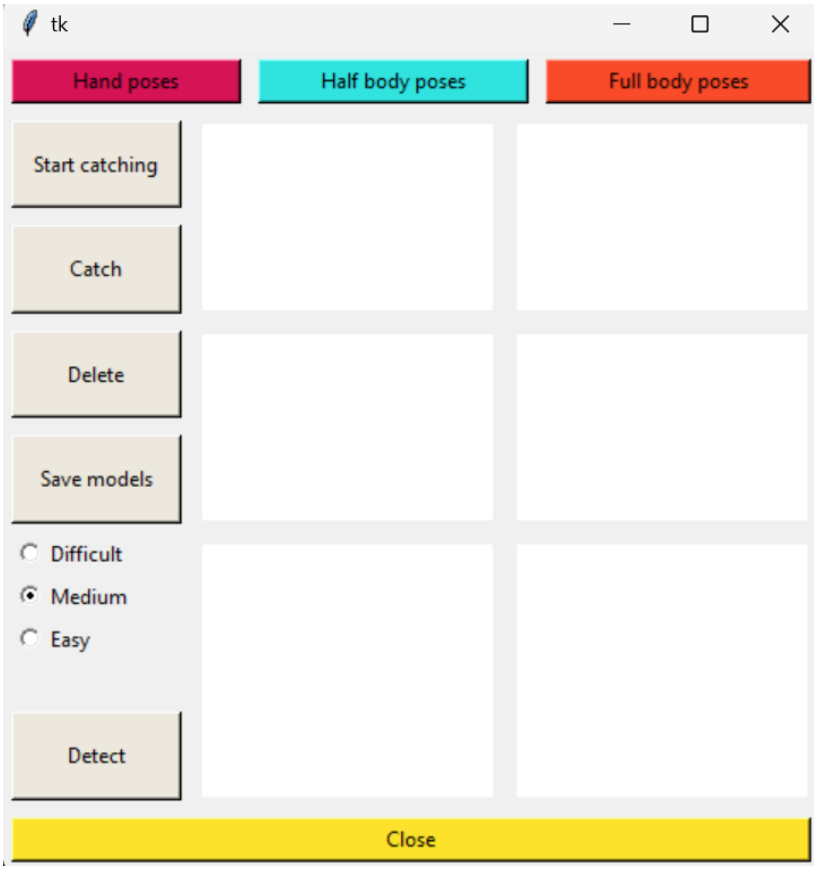

Fig. 2.3. Estructura del panel del botón 'Crea tus poses'

Por otro lado, el botón ‘Empezar espectáculo’ habilita diferentes actividades. En primer lugar, ‘Sígueme’, que permite al dron seguir al usuario. También, los botones ‘Dedos’ y ‘Poses’ que trabajan con la pose de los dedos y con las posturas corporales, respectivamente. Por último, el botón de las ‘Caras’ que trabaja con las expresiones faciales. Estas tres últimas opciones permiten guiar y controlar el dron según las poses definidas.

Más adelante, se hará referencia a estas funcionalidades y se detallará cómo se ha trabajado con ellas, las funcionalidades implementadas, etc. En este proyecto se ha utilizado con mayor énfasis los botones ‘Dedos’ y ‘Poses’.

Paralelamente, se encuentra el ‘Circo de los Colores’, que, aunque no se priorizará en este proyecto, resulta interesante mencionar. Al igual que el ‘Circo de las Poses’, este tiene un panel principal destinado a configurar el escenario de trabajo requerido. Se establece la conexión con el dron, se calibran los colores a utilizar y también dispone de la opción de ‘Empezar espectáculo’.

En la **Fig. 2.4**, se observan las funciones asociadas al botón ‘Empezar espectáculo’ en el ‘Circo de los Colores’. En primer lugar, ofrece la posibilidad de jugar al ‘Match Ball’, una especie de tenis, en el que el dron actúa como pelota y, mediante raquetas de colores, el dron es capaz de seguir los colores y participar en el juego. Luego, incluye las opciones ‘Atrapa al ladrón’ y ‘Sígueme’, en las cuales el dron es capaz de seguir un objeto de un color específico que está en movimiento.

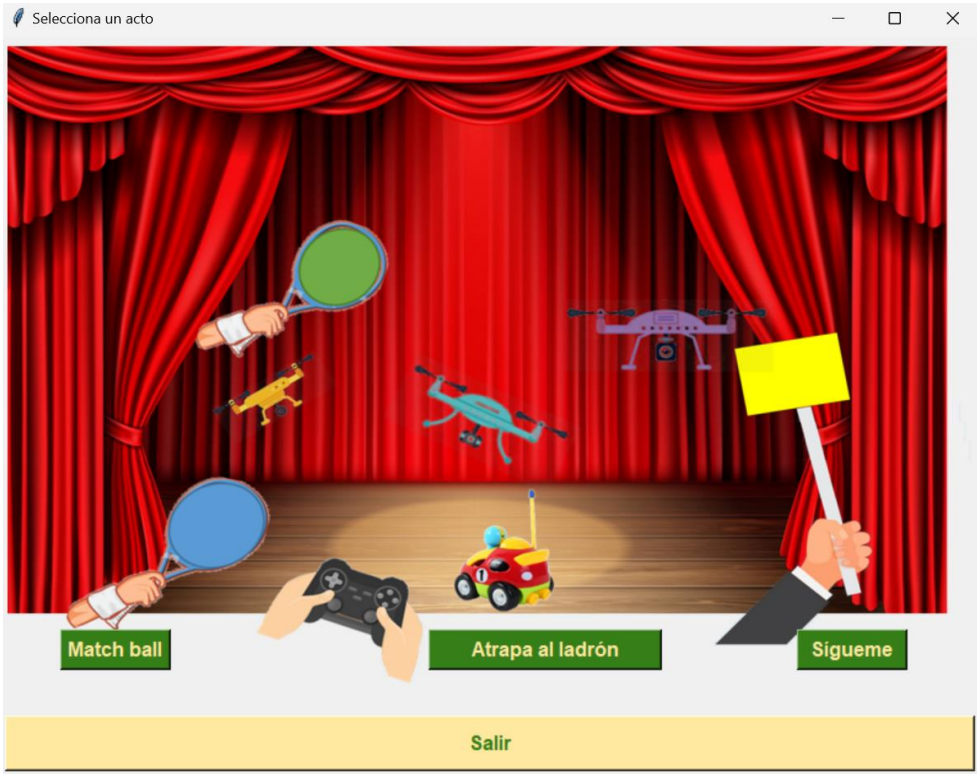

Fig. 2.4. Disposición del botón 'Empezar espectáculo' en el ‘Circo de los Colores’.

Por otra parte, la aplicación web App formada por 'Tello Drone Server' y 'Tello Web App', posibilita el control del dron desde una aplicación web App conectada al servidor ‘Tello Web Server’, el cual se encarga de la comunicación con el dron y la transmisión de los comandos necesarios. Se presentan dos opciones: ‘One Dron’ para jugar con un dron y un usuario, y ‘Play’ que permite la participación de múltiples usuarios.

Por ejemplo, si se analiza la opción de ‘One Dron’, el primer paso es establecer la comunicación con la web App y el dron a través del servidor. Una vez establecida esta comunicación, utilizando la interfaz web App y su variedad de botones, es posible controlar el dron mediante los comandos enviados, tal y como se observa en la **Fig. 2.5**.

Por otro lado, la opción de ‘Play’ sigue el mismo mecanismo que ‘One Play’ pero con la participación de más de un usuario. Se determina el número de concursantes en el servidor y luego se establece la comunicación correspondiente. Simultáneamente, la interfaz web App se habilita para cada jugador según su turno, evitando así congestiones en la transmisión de comandos.

Adicionalmente, en la interfaz web App también hay funcionalidades extra, como configurar los colores y enviar los comandos con la voz.

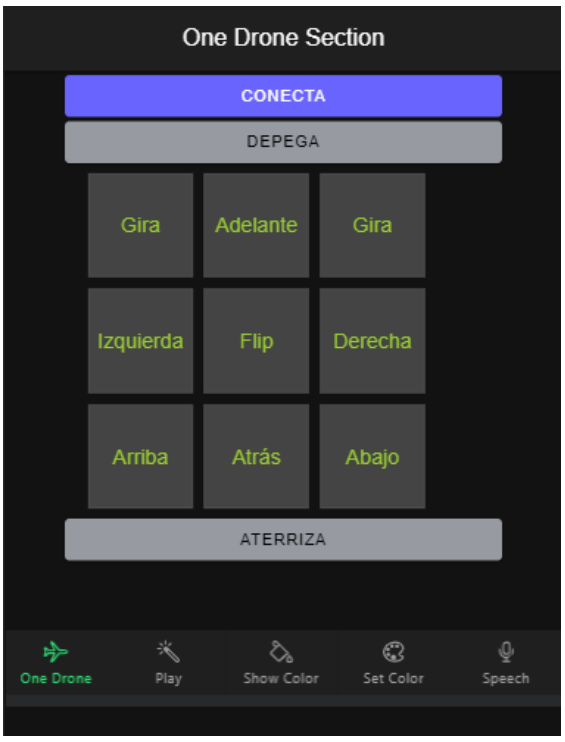

Fig. 2.5. Disposición de los comandos de la Web App

2.4. Tecnologías

En este apartado se mencionarán las tecnologías y el software utilizado en este proyecto.

2.4.1. Ionic y Vue

Ionic Framework es un SDK de *frontend* de código abierto que permite el desarrollo de aplicaciones móviles y web App. Proporciona una variedad de componentes de interfaz que lo convierte en un entorno de desarrollo muy versátil [7].

Posibilita la creación de aplicaciones multiplataforma modernas y de alta calidad, permitiendo interfaces de usuario flexibles para diferentes tipos de plataforma con una base de código compartida.

Ofrece un diseño simple, eficaz y comprensible y puede trabajar con diferentes marcos de interfaz de usuario como Angular, React y Vue, e interactuar con tecnologías web App como HTML, CSS, JS, etc.

Además, Ionic tiene la CLI (*Command-Line Interface*), que es una interfaz con un conjunto de comandos, ejecutables desde la terminal, que facilita la automatización de tareas para el desarrollo de aplicaciones, permitiendo una instalación muy sencilla.

Ionic ofrece la posibilidad de trabajar con diferentes elementos como son los ‘buttons’, ‘canvas’, ‘grids’, etc. El elemento más utilizado en las aplicaciones de este proyecto son los llamados ‘tabs’ o pestañas, que permiten crear dentro de una misma aplicación diferentes pestañas completamente independientes entre ellas y realizar funciones totalmente diferentes.

Vue, por otro lado, es un marco de JavaScript de código abierto utilizado para el desarrollo de interfaces de usuario. Se basa en HTML, CSS y JavaScript estándar. Tiene un lenguaje particular que trabaja de forma rápida, práctica y sencilla. Vue es un ecosistema que garantiza las características comunes y necesarias para el desarrollo *frontend*.

Los códigos de Vue, están formados principalmente por tres elementos: el *template* donde se escribe en HTML y se definen la estructura de la interfaz, y los elementos que habrá; el *script* donde se realizan todas las funciones de lógica y datos de la interfaz; se definen métodos, propiedades y eventos; y el *style* dónde se define en CSS los estilos y el diseño de los componentes de la interfaz [8]

En este proyecto se van a desarrollar aplicaciones web App que trabajarán con Ionic y Vue simultáneamente. Todas las características y elementos de ambos *frameworks* se combinarán y se desarrollarán en paralelo. Estos componentes

se irán viendo con mayor detalle durante el proceso de creación de las aplicaciones y se explicará cómo se ha trabajado con ellos en este proyecto.

2.4.2. Python

Python es un lenguaje de programación de alto nivel muy utilizado por su eficiencia y versatilidad. Tiene una sintaxis legible y relativamente sencilla que permite a los desarrolladores ser más rápidos y productivos.

Python tiene un lenguaje interpretado, es decir, la ejecución del código se realiza línea por línea, lo que ayuda a los desarrolladores tener una rápida detección de errores. También es un lenguaje que no requiere de una predeterminación del tipo de variables, haciendo que sea un lenguaje dinámico [9].

Además, contiene una gran cantidad de librerías/bibliotecas previamente programadas que contienen códigos reutilizables muy útiles para los desarrolladores. Por ejemplo, las más conocidas son Matplotlib, NumPy, OpenCV, Pandas, etc.

Python es comúnmente utilizado para el desarrollo de aplicaciones web App, desarrollo de *software*, la ciencia de datos, el *Machine Learning*, la automatización de scripts, aplicaciones con inteligencia artificial, etc.

El Robomaster Tello está basado en Python 3.0, que será con el que se trabajará en este proyecto.

2.4.3. MQTT

MQTT (siglas en inglés 'Message Queuing Telemetry Transport') es un protocolo de transporte de mensajería basado en estándares y reglas que se utiliza para la ligera comunicación entre diferentes tipos de equipos.

Los dispositivos portátiles, los de IoT (Internet of Things), sensores, microcontroladores, aplicaciones móviles, dispositivos de automatización del hogar, plataformas de nube, etc., generalmente necesitan transmitir y recibir datos constantemente a través de redes con recursos limitados como el ancho de banda, la frecuencia de transmisión o entornos de baja potencia.

Para conseguir una buena comunicación entre dispositivos, el uso del protocolo MQTT se ha vuelto una implementación imprescindible. Además, su eficiencia y simplicidad hace que sea muy fácil trabajar en dominios restringidos.

Este protocolo tiene diferentes componentes que lo engloban, como el cliente MQTT, que hace referencia a aquellos dispositivos que envían/reciben mensajes; el bróker MQTT, que es el servidor que actúa como intermediario

entre clientes, recibiendo mensajes y reenviándolos donde corresponda; y el tema, que es una cadena de texto con palabras clave que actúan como canal de filtro para los clientes, donde podrán publicar o suscribirse. Por último, se encuentra el QoS (siglas en inglés Quality of Service) que permite regular el nivel de calidad del servicio, que va desde el 0 (menor calidad) al 2 (calidad garantizada) [10].

El protocolo MQTT, sigue un modelo de comunicación de publicación/suscripción, en el que el cliente puede publicar un mensaje con un tema en específico y también puede suscribirse a temas en concreto que le permiten recibir mensajes publicados en ese tema. Por otro lado, también existe la posibilidad de retener un mensaje para que cuando el cliente se conecte, reciba el mensaje más reciente publicado en cierto tema.

El hecho de utilizar un patrón de publicación/suscripción permite un desacoplamiento con el receptor del mensaje, haciendo que el transporte sea más efectivo, flexible y escalable.

El protocolo MQTT tiene algunas ventajas con respecto a otros protocolos, interesantes a mencionar. Es un protocolo fiable y seguro que garantiza la seguridad de los mensajes, resultando en una buena protección de la autenticación del cliente y el bróker. Paralelamente, permite una comunicación asíncrona, con tiempos de latencia bajos y tiene una gran eficiencia de uso de ancho de banda.

En la **Fig. 2.6**, se puede observar el funcionamiento básico de publicar y suscribir mediante el ejemplo del envío del comando de 'TakeOff'.

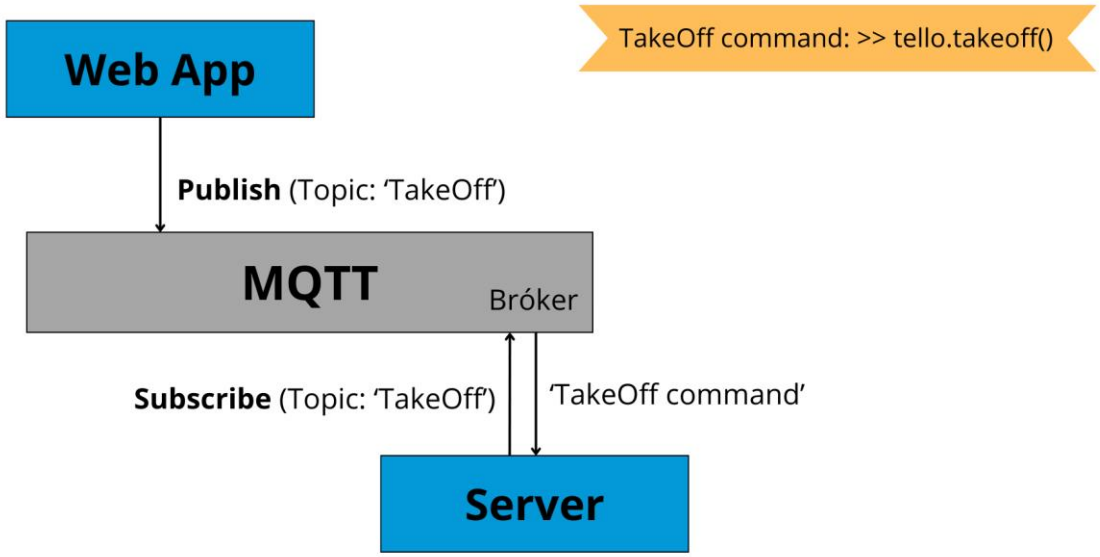

Fig. 2.6. Ejemplo del mecanismo de publicación/subscripción del protocolo MQTT

Con el ejemplo anterior, se puede ver que desde una Web App se publica un comando con el tema 'TakeOff', que hace referencia al comando de despegar el dron. Por otro lado, un servidor se suscribe al bróker mediante el tema 'TakeOff', por lo que recibirá todos los mensajes que dispositivos externos hayan publicado bajo el tema 'TakeOff'. Cuando le llegue un mensaje al servidor del tema 'TakeOff' podrá enviarle al dron la indicación de despegar.

2.4.4. Mediapipe

MediaPipe es una biblioteca o framework de código abierto desarrollada por Google que permite crear soluciones de Machine Learning e Inteligencia Artificial, y permite aplicar soluciones y servicios de aprendizaje automático con eventos en tiempo real, como la detección de objetos, el seguimiento facial, reconocimiento de partes del cuerpo, etc. [11].

A través de la API MediaPipe para Python, en este proyecto se implementan las diferentes funcionalidades de detección de poses, dedos, etc., en tiempo real.

MediaPipe, tiene la capacidad de percibir la forma y movimiento de las manos, la cara y el cuerpo de un usuario, procesando esa información y utilizándola para generar diferentes funcionalidades.

En el Tello Engineering Ecosystem, se encuentran implementadas las tres detecciones posibles; cara, manos y cuerpo. No obstante, se utilizarán en su mayor medida la detección de las manos y del cuerpo.

Para poder crear, guardar y detectar las poses, es necesario tener puntos de referencia. MediaPipe almacena las coordenadas 2D (abscisas y ordenadas) de estos puntos para luego trabajar con ellas y poder detectarlas posteriormente. Es decir, guarda las coordenadas en vectores, para luego comprobar su posición. Analiza si cierto punto se encuentra dentro de unos límites circulares con un radio determinado. De esta manera, se logra también la detección de poses iguales, pero con diferente orientación dentro del plano.

En otros términos, MediaPipe consigue detectar y reconocer diferentes partes del cuerpo, creando en esta un modelo con puntos de referencia para realizar una localización precisa de los puntos clave. Analiza tanto la postura, la forma y la orientación de la pose detectada.

Existe, 'MediaPipe Hands' que es una solución en el que identifica las manos señalizando y etiquetando 21 puntos de referencia 3D sobre ellas [12].

En la **Fig. 2.7**, se observan los 21 puntos con lo que se definen las manos.

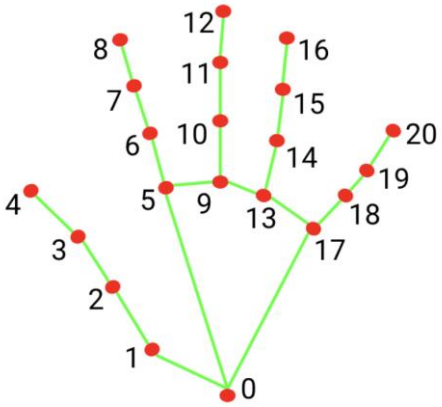

Fig. 2.7. Mediapipe Hand Land Marks.

Paralelamente, se encuentra el 'MediaPipe Body', que divide y marca el cuerpo del usuario en 33 puntos de referencia. En el 'Circo de drones', se encuentra tanto la detección de medio cuerpo como la de cuerpo entero. Por eso, cuando se trabaja con medio cuerpo, se acota este vector de 33 puntos en un vector de 14 puntos que son los que equivalen del punto 11 al 23, como se observa en la **Fig. 2.8.**

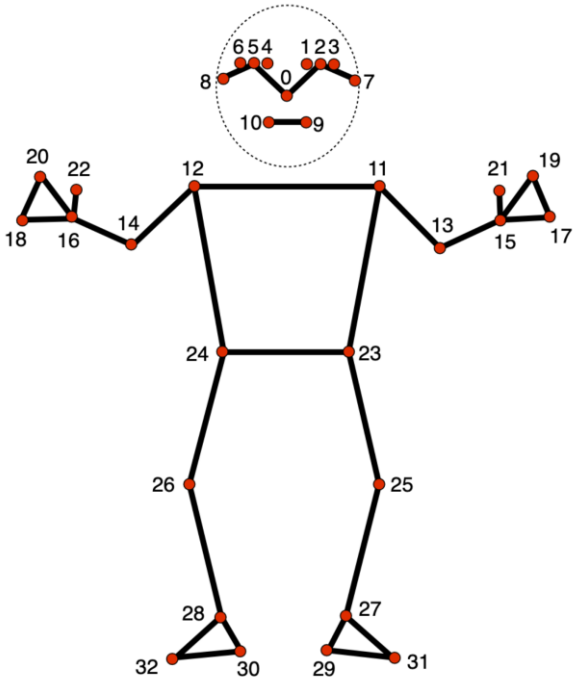

Fig. 2.8. Mediapipe Body Land Marks.

2.4.5. OpenCV

OpenCV es otra biblioteca de código abierto (siglas en inglés Open Source Computer Vision Library), que se utiliza para el procesamiento de imágenes y vídeos, como la detección de objetos, el reconocimiento de patrones, seguimiento de movimiento, la manipulación de las imágenes para la detección de los colores, etc. [13].

OpenCV es una librería multiplataforma optimizada por procesadores de Intel, que permiten realizar de manera rápida y adecuada el procesado de imágenes.

Esta herramienta se ha utilizado más en el 'Circo de los Colores', que ha permitido jugar con la detección de colores, como el cambio de la escala de colores, por ejemplo, pasando la imagen que recibe a una imagen en blanco y negro. También, permite la corrección de la saturación, contraste y color de las imágenes. Es capaz de detectar objetos de cierto color durante un vídeo o en una imagen, o contornear objetos de una imagen.

Tiene una variedad de funciones que permiten conseguir estos objetivos; a continuación, se mostrará un ejemplo de estas funcionalidades.

```
>> img = cv2.imread('fotoAvion.jpg')
>> hsv = cv2.cvtColor(img, cv2.CLOR_BGR2HSV)
```

En este ejemplo, se crea una variable 'img' que hace referencia a una imagen JPG cogida del escritorio. La función 'inread' lee la imagen y la asigna a 'img'. Seguidamente, se procesa esta imagen y se convierte del espacio BGR (Blue, Green, Red) al espacio HSV (Hue, Saturation, Value), para luego trabajar con estos valores.

Otro ejemplo interesante sería pasar una imagen a escala de grises.

```
>> img = cv2.imread('imagen.jpg', cv2.IMREAD_GRAYSCALE)
>> canny_img = cv2.Canny(img, 100, 200)
```

Donde primero se carga la imagen, cogida de una fotografía en formato JPG, y se guarda en escala de grises. Luego, mediante la función 'Canny' de OpenCV e indicando los umbrales, conseguiremos tener una imagen con los bordes de los elementos de la imagen remarcados

CAPÍTULO 3. OBJETIVOS Y PLAN DE TRABAJO

En este capítulo se van a desarrollar los objetivos de este proyecto, las tareas realizadas para alcanzarlos y cómo se ha planteado la organización para poder llevarlo a cabo. Este proyecto tuvo una duración de cinco meses aproximadamente, empezando en septiembre y finalizando en enero de 2023.

3.1. Objetivos

En líneas generales, el objetivo de este proyecto es contribuir en el desarrollo y crecimiento del Tello Engineering Ecosystem. Se pretende trabajar en la mejora y expansión de las funcionalidades del ecosistema previamente creado y desarrollado, así como la adición de nuevas herramientas, conceptos y juegos, para que en futuro estudiantes puedan seguir con dichas contribuciones. Asimismo, se busca desarrollar nuevas actividades que impulsen la innovación en las demostraciones dirigidas al público en entornos interiores.

Para conseguir este propósito se establecieron diferentes objetivos:

- Introducir mejoras en el Circo de las Poses, mediante la implementación de funcionalidades pendientes y la introducción de nuevas para características para mejorar su accesibilidad y utilidad. Por ejemplo, la implementación de la detección con las poses del cuerpo, tanto en la parte de practicar como controlar el dron.

- Integrar la capacidad de capturar imágenes utilizando un dispositivo móvil en el entorno del "Circo de las Poses". Se ha implementado una plataforma web App capaz de enviar las imágenes en tiempo real al circo, las cuales serán procesadas posteriormente.

- Establecer el "Circo de las Imágenes" como un nuevo componente del ecosistema, permitiendo el trabajo con imágenes, transmisión de vídeos en tiempo real y funcionalidades panorámicas. Esto incluye la creación de una aplicación web App complementaria que recopile y muestre los *frames* enviadas por el circo a los usuarios conectados. Este objetivo se puede dividir en diferentes puntos:

    - 1. Implementación de una nueva ventana en el ‘Circo de las Imágenes’ que captura fotos de la cámara del dron y las envía a la web App en la que estarán conectados los visitantes.

    - 2. Implementación de una nueva ventana en el ‘Circo de las Imágenes’ que captura en *streaming* las imágenes de la cámara del dron y lo muestra en la web App en tiempo real. También es capaz de enviar los archivos de vídeos creados en el circo a la web App.

  - 3. Implementación de una nueva ventana en el 'Circo de las Imágenes' que captura imágenes de la cámara del dron y las junta creando una foto panorámica que luego envía a la web App en la que estarán conectados los visitantes.

  - 4. Desarrollo de una web App dónde los usuarios conectados tendrán la posibilidad de recibir los tres tipos de *frames* (imágenes, vídeo y panorámica), y visualizar y descargar la recopilación de imágenes en galerías.

- Integración del código en el ecosistema mediante la consolidación y unificación de los diversos fragmentos de código. Para ello, se llevará a cabo la creación de repositorios individuales, cada uno incluyendo la totalidad del código junto con la información esencial. Esto permitirá a cualquier lector futuro comprender de manera integral el proceso de creación del código, así como la estructura y el funcionamiento de este.

3.2. Tareas

Para lograr los objetivos previamente mencionados, se han llevado a cabo una serie de tareas específicas:

1. El primer desafío abordado consistió en la adquisición de nuevos lenguajes de programación, con el fin de comprender sus funcionalidades y explorar sus diversas aplicaciones. Así como, adquirir conocimientos de nuevos *frameworks* ya existentes y aprender a integrarlos eficazmente en el código de trabajo.

   Se invirtió un tiempo para aprender Ionic, Vue, entre otras utilizando códigos y ejemplos más sencillos para entender estos nuevos conceptos. La comprensión profunda se facilitó mediante la consulta de videotutoriales, brindando una perspectiva visual y práctica.

2. Luego, fue esencial comprender todo el código base con el que se trabajaría, abarcando todas las funcionalidades del código, la estructura de las clases, la organización de los métodos, la relación entre clases y la metodología de juego de los circos. La práctica y la interacción con el circo fueron cruciales para avanzar con mayor rapidez en el proyecto.

3. Se iniciaron las propuestas de mejora mediante la optimización del código base, haciéndolo más robusto y fiable, se añadieron botones, *messagebox*, métodos, etc. Además, se analizaron todas las funcionalidades para determinar cuáles faltaban por implementar y se incorporaron al proyecto.

   Por ejemplo, se implementó la parte de guiar el dron con la detección de las poses del cuerpo, tanto en la parte de practicar como de vuelo.

4. Se abordó un objetivo adicional: la implementación de la posibilidad de realizar todas aquellas funcionalidades del código base con la cámara del móvil. Hasta ahora, la detección de poses con los dedos o con el cuerpo solo estaba implementado con la cámara del ordenador, en esta tarea se trabajó con la cámara del móvil de los usuarios conectados que permitió una mayor interacción con estos.

   Para ello, se desarrolló una web App con Ionic y Vue que permitió cargar las imágenes de vídeo del móvil y enviarlas a un bróker interno, para luego poder recibirlas en el circo.

5. Se implementó un nuevo circo denominado 'El circo de las Imágenes', en el que se implementan funcionalidades totalmente diferentes a los otros circos. Se abordaron tres formatos de imágenes; fotos, vídeos y panorámicas, implementando nuevas ventanas de control en el circo y una plataforma web App para recibir y gestionar todas las imágenes enviadas por el circo.

6. Se integró el código en el ecosistema. Se crearon los correspondientes repositorios para los diferentes códigos, cada uno acompañado de una explicación detallada sobre el proceso de instalación del repositorio, la estructura del código y su funcionamiento básico.

3.3. Expectativas Organizativas

En cuanto a las expectativas organizativas, se intentó definir una estructura organizativa que sirviera de guía para poder seguir durante todo este proceso. En el Diagrama de Gantt de la **Fig. 3.1** se muestran las expectativas organizativas de este proyecto.

Fig. 3.1. Diagrama de Gantt de la Planificación Inicial.

Al final de este trabajo, se realizará una reflexión sobre la organización real de este proyecto, los cambios que ha habido, y cómo se han ido alcanzando los objetivos, si se han añadido contribuciones, entre otros.

CAPÍTULO 4. PRIMERAS CONTRIBUCIONES

En este capítulo se expondrán las primeras contribuciones, realizadas sobre el código base, así como las primeras mejoras implementadas, entre otros aspectos.

4.1. Modificaciones del código principal

Como se ha mencionado anteriormente, en la primera etapa de este proyecto, se atravesó un periodo de aprendizaje del código base. Esto implicó entender la estructura, organización, funcionalidades y aplicaciones de este. Durante este proceso, se analizó el código base y se añadieron ajustes en áreas que estaban incompletas.

Un ejemplo de estas modificaciones se encuentra en el 'espectáculo' del 'Circo de las Poses'. Aquí, se exploraron diferentes formas de dirigir el dron, ya sea mediante poses de manos, cuerpo o cara. Sin embargo, no se había implementado en su totalidad el control del dron mediante poses corporales, por lo que en esta sección se detallará su implementación.

El control del dron mediante las poses tiene dos posibilidades de juego: el nivel fácil y el nivel difícil. En el nivel fácil, las poses del cuerpo ya están predeterminadas, mientras que, en el nivel difícil, las poses deben crearse y definirse previamente en la sección de 'Crear Poses', para luego ser guardarlas y utilizadas en la interacción con el dron.

Una vez elegido el nivel y definidas las poses, se presentan dos posibles actividades adicionales: la actividad de practicar las poses, que permite ensayar las posiciones preestablecidas y mejorar así el control del dron, y la fase de vuelo, en la que ya se interactúa directamente con el dispositivo.

En la **Fig. 4.1**, se pueden observar los botones con los que se interactúa tanto para practicar como para volar el dron según el caso que se quiera ejecutar. En la parte superior del panel se encuentra la elección del nivel de dificultad de la detección: 'Fácil' para el nivel fácil y 'Tus poses' para el nivel difícil, que utiliza las poses previamente creadas y guardadas.

Además, se observa el botón de 'Practica', que se irá actualizando según la fase en la se trabaje. El botón 'Práctica', como se ha mencionado anteriormente, comprueba la perfecta detección de las poses para poder asegurar un vuelo seguro en los siguientes pasos. Una vez seleccionado el botón de 'Práctica', se actualiza y aparece un botón llamado 'Quiero volar', que permite guiar el dron mientras vuela. Una vez seleccionada esta opción, aparecen dos botones rojos: 'Connect' y 'Take Off', que son condiciones que deben cumplirse antes de poder guiar el dron: establecer la conexión con el dron y despegarlo.

Luego, se despliega la opción de controlar el dron a través de las poses del cuerpo, enviándole los comandos correspondientes. Finalmente, se encuentra un botón llamado ‘Salir’ que permite cerrar esta ventana.

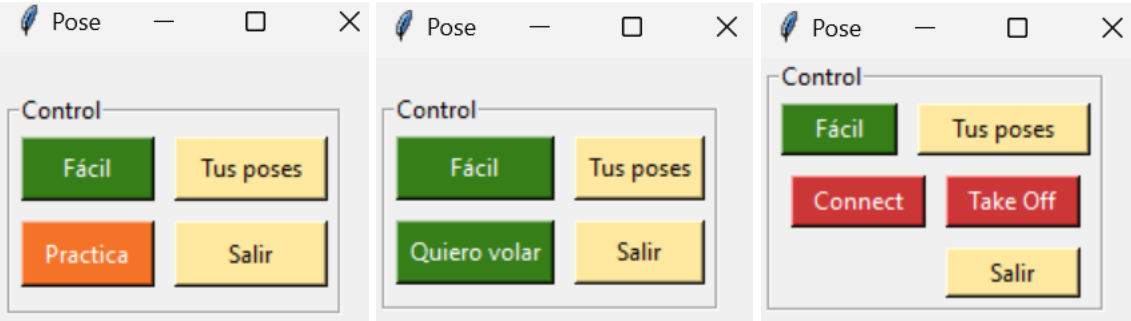

Fig. 4.1. Disposición de los botones que permiten practicar y volar el dron

En este primer reto, en el nivel fácil, se han acabado de implementar estas funcionalidades y se ha comprobado el buen funcionamiento de este, a diferencia del nivel difícil, en el que se han implementado ambos casos en su totalidad: la detección de las poses del cuerpo para practicar y el vuelo del dron, enviándole los comandos necesarios un vuelo preciso y seguro.

En la **Fig. 4.2**, se puede observar el panel de juego del nivel fácil, el cual servirá como referencia para implementar el control del dron por detección de poses del cuerpo en el nivel difícil.

A diferencia de la parte de ‘Crear poses’, donde se encuentran diferentes opciones de detección: cara, medio cuerpo y cuerpo entero, en esta área para guiar el dron con las poses del cuerpo, se trabaja únicamente con la detección de medio cuerpo, no se contemplará la posibilidad de trabajar con el cuerpo entero. La parte de detección con la cara si será posible, pero estará implementada en una sección separada.

En la **Fig. 4.2**, se observan las poses predefinidas con muñecos que ejemplifican las posibles poses con las que se podrá interactuar tanto en la parte de practicar como en la del vuelo del dron. En el nivel difícil se requiere trabajar con las fotos creadas de la sección ‘Crear Poses’, por lo que se ha implementado un código que permita guardar estas imágenes en un vector para luego ser utilizadas.

A lo largo de este proceso, se ha ido perfeccionando el código del nivel difícil añadiendo condiciones que permitan tener un código más robusto y fiable.

En la parte superior de las poses de la **Fig. 4.2**, se encuentran representados en color azul, los dibujos que aparecerán en el *display* del dron, indicando el comando que espera recibir y así el usuario podrá saber que pose debe realizar.

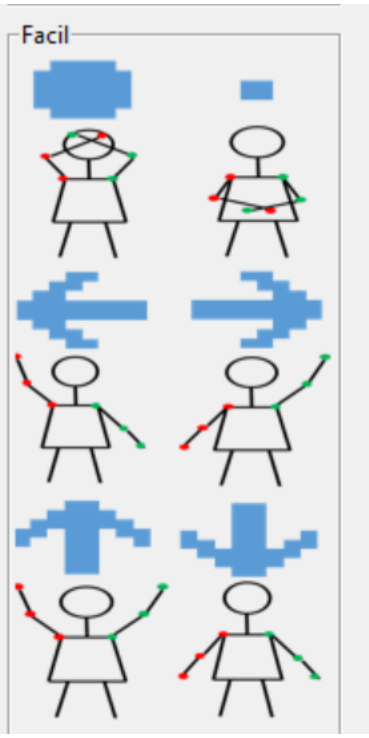

Fig. 4.2. Pestaña del nivel fácil de las Poses del cuerpo

El proceso para interactuar con las poses corporales es el siguiente: una vez el dron se encuentre en vuelo, este enviará los comandos que espera recibir en forma de dibujo, ya sean flechas, puntos, etc., de forma aleatoria. Estos comandos representarán una posición corporal determinada, tal y como se ve en la **Fig. 4.2**. Luego, el usuario deberá representar la posición que espera recibir el dron, una vez la reciba, el dron ejecutará la acción que represente esa posición. En el caso de que el dron no reciba correctamente el comando, aterrizará para así garantizar la seguridad tanto del dron como de los usuarios.

En el Anexo A, añadido al final de esta memoria, se pueden observar las funciones que dan juego a los botones 'practicar' y 'volar', que se han tenido que implementar para el correcto funcionamiento del nivel difícil.

Paralelamente, se han añadido mejoras en la sección de 'Crear Poses' y en la sección de 'Dedos' para la detección con las manos. Faltaba precisión y exactitud, por lo que se han analizado los puntos de referencia de los dedos, y en vez de trabajar sobre todos los puntos, se trabajó sobre los puntos de claves de interés y se han modificado las funciones correspondientes.

Paralelamente, se ha ido haciendo el código más robusto. Añadiendo *messageBox* tanto de errores si algo no funcionaba bien, como informativos para aclarar las dinámicas del juego y facilitar al usuario su compresión.

CAPÍTULO 5. INTEGRACIÓN DE LA DETECCIÓN CON CÁMARA DEL MÓVIL

En este capítulo se describe cómo se ha transformado el código base para poder tener las mismas funcionalidades del código principal, pero con la detección de la cámara del móvil en vez de con la del ordenador. Se ha adaptado la parte del 'Circo de las Poses' y se ha implementado una web App que permite la transmisión de las imágenes que recibe el circo.

5.1. Introducción

Para este objetivo se ha creado una nueva funcionalidad, la opción de crear poses, detectarlas y volar el dron transmitiendo las imágenes desde la cámara de un móvil. Se ha adaptado el circo de las poses, tanto en *backend* como en *frontend*, la elección del tipo de cámara con el que se desea trabajar (ordenador o móvil), la recepción de las imágenes transmitidas desde la cámara del móvil, la manipulación de estas imágenes, etc. En los siguientes apartados se explica con detalle cómo se ha implementado esto y cómo ha afectado en el código.

Por otro lado, se ha creado una web App que transmite en tiempo real las imágenes del móvil y las envía al circo dónde también se pueden visualizar en tiempo real y así trabajar con la detección de poses como hasta ahora.

Todo el código realizado para el desarrollo de la transmisión de las imágenes mediante el móvil, de la parte del circo, se encuentra disponible en el siguiente repositorio de GitHub: https://github.com/anna22itu/CircoDronesTello.git. Asimismo, la parte correspondiente a la web App se puede encontrar en el siguiente repositorio: https://github.com/anna22itu/WebAppMobileTelloDroneCircus.git.

5.2. Implementación del software con el móvil en el circo

Para poder implementar el funcionamiento de la detección con la cámara del móvil en el circo de drones, se han tenido en cuenta dos componentes. En primer lugar, se ha abordado la decisión de trabajar con la cámara del ordenador o la cámara del móvil en el circo. En segundo lugar, se ha definido cómo recibirá el circo las imágenes que le envía la cámara del móvil y cómo se trabajará con estas imágenes.

En el circo, para que el usuario pueda decidir con qué cámara desea jugar, se ha creado un nuevo botón en el panel principal llamado 'Selecciona la cámara' que abre una nueva ventana llamada 'Camera' como se observa en la **Fig. 5.1**. Antes de empezar a jugar con las poses, esta elección deberá estar configurada.

Para conseguir esto, se ha creado una nueva clase llamada 'Camera' donde están definidos todos los componentes y métodos que permiten llevarlo a cabo.

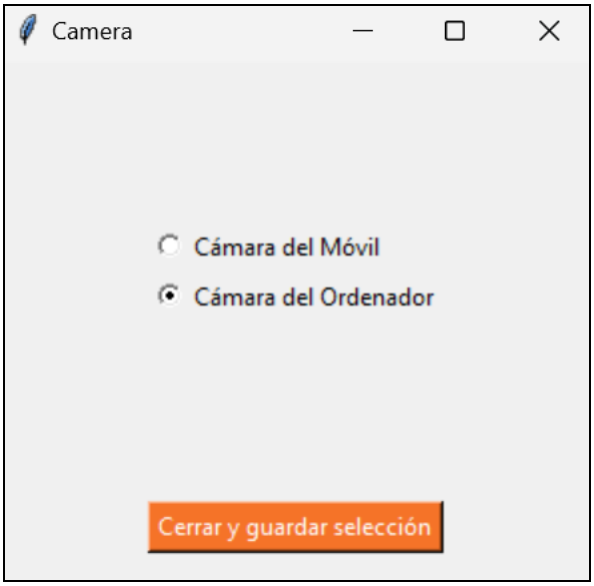

Fig. 5.1. Pestaña de la elección de la cámara**.**

Tal y como se muestra en la imagen anterior, se han añadido dos *radiobuttons* con valores diferentes. Cuando se selecciona la cámara del móvil, el valor del *radiobuttons* será 1 y, valdrá 2 en el caso contrario, y se irá actualizando cada vez que se selecciona uno de los *radiobuttons*. Esta funcionalidad está implementada en la función 'updateSelection', dónde se asigna el valor correspondiente a la variable definida para los *radiobuttons*.

Además, se ha definido también una variable de tipo *integer* que se utilizará para reconocer la decisión tomada en diferentes momentos de la ejecución. Al hacer clic en el *button* 'Cerrar y guardar selección', se le atribuye a esta variable *integer* el valor que se guarde del *radiobutton* seleccionado. La variable se introducirá como una nueva variable en las diversas clases que forman el circo, permitiendo saber en cualquier momento de la ejecución del código con qué cámara se está trabajando. Esta operatividad se implementa en la función 'closeSelection'.

Una vez resuelta la elección del tipo de cámara con el que interactuará, se adapta el código según esta elección de la siguiente manera. Se ha creado una clase llamada 'VideoStreamer', cuyo parámetro de entrada es un *integer* con la decisión del tipo de cámara y que según la decisión retransmitirá los *videoframes* de una fuente u otra. Al final del documento, en el Anexo B, se puede ver la estructura de esta clase y la separación de procedimientos según la elección.

Si la cámara seleccionada es la del móvil, y, por tanto, el valor del *integer* es 1, se establece la conexión MQTT para recibir las imágenes del móvil. Primero, se

define una instancia de un cliente MQTT llamado ‘VideoService’. A continuación, se establece el nombre de usuario y la contraseña para la autenticación en el servidor MQTT y establecer la conexión. Luego se define una función de devolución al cliente MQTT llamada ‘on_messageStreamer’, que se ejecutará cada vez que el cliente reciba un mensaje al tema al que se haya suscrito. Seguidamente, se establece la conexión con el servidor MQTT con las direcciones especificadas y el cliente se suscribe al mismo tema en el que se publican las imágenes procedentes de la cámara del móvil. Por último, se inicia un bucle de eventos que permite al cliente recibir mensajes del tema al que se ha suscrito.

La función ‘on_messageStreamer’ se ejecuta cuando el cliente recibe un mensaje que contiene datos de una imagen en base64. Se procesa, decodifica, redimensiona y almacena el mensaje en una variable llamada ‘current_img’.

En cambio, si la cámara seleccionada es la del ordenador (el valor del *integer* es 2), se define la variable ‘cap’ como objeto de ‘cv2.VideoCapture(0)’. Esta variable almacena capturas de vídeo del dispositivo principal (referencia ‘0’).

Paralelamente, para que en todas las secciones de código en las que se llama a esta clase se reciban las imágenes, se define una función llamada ‘getFrame’, que devuelve en cada caso el tipo de imagen correspondiente. Si es la cámara del móvil, se devuelve la variable ‘current_img’; y en el caso de que sea la cámara del ordenador, se captura un cuadro de vídeo de la cámara del ordenador mediante el método ‘read’ del objeto ‘cap’, lo redimensiona y lo retorna.

Por último, en la clase ‘VideoStreamer’, se define una función llamada ‘disconnect’. Si la cámara corresponde a la del móvil, se desconecta la conexión con el cliente MQTT y se detiene el bucle de eventos previamente inicializado. Si es la cámara del ordenador, se utiliza el método ‘release’ del objeto ‘cap’ para cerrar la cámara del ordenador y liberar recursos.

Con estas implementaciones, se ha conseguido permitir al usuario la elección de elegir con qué cámara trabajar (ordenador o móvil) y recibir las imágenes enviadas por el móvil al circo, permitiendo trabajar con ellas durante la detección de poses.

El código base se ha adaptado en función de esta elección. Por ejemplo, en las funciones de ‘practicar’ y ‘volar’, mencionadas en el capítulo anterior, se ha añadido esta elección de tal manera que, al iniciar la detección de la cámara, se llame a la clase ‘VideoStreamer’ para obtener la correcta transmisión.

5.3. Implementación de la web App

Para la transmisión de las imágenes del móvil al circo, se ha creado una web App muy sencilla que captura la imagen del móvil en *videostreaming*. Se ha desarrollado con Ionic y Vue.

La web App se divide en tres tabs, la pestaña principal llamada 'HomePage', la pestaña que graba en *videostreaming* llamada 'VideoPage', y la pestaña que hace capturas de foto llamada 'PhotoPage'. En este apartado se han implementado todas las pestañas, aunque para alcanzar el objetivo que necesitaba el circo solo se ha utilizado la 'VideoPage', ya que solo requería de la transmisión en vivo de la cámara del móvil.

En la **Fig. 5.2**, se puede observar la estructura y disposición de la web App. Se pueden observar los tres *tabs* mencionados anteriormente y la disposición del' VideoPage'.

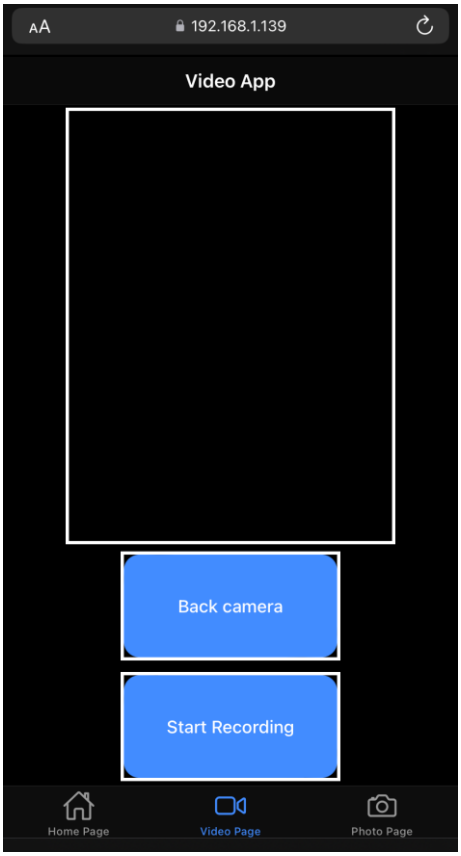

Fig. 5.2. Estructura de la Web App

La pestaña 'HomePage', es la pestaña de inicio y, la 'VideoPage' y la 'PhotoPage' tienen la misma estructura y los mismos elementos, pero con funcionalidades diferentes.

El 'VideoPage' y el 'PhotoPage' se componen de tres elementos principales; un *canvas* donde se mostrarán las imágenes, un botón donde se indica con qué cámara del móvil se graba (delantera o trasera) y un botón de empezar a grabar ('Start Recoding'), como se observa en la **Fig.5.2**.

La dinámica de esta pestaña es la siguiente: una vez elegida con qué cámara se quiere empezar a grabar (delantera o trasera), se clica el botón 'Start

Recording' que va a mostrar en el *canvas* de la web App las imágenes que captura de la cámara del móvil, acto seguido, si en el circo se ha establecido la conexión correcta con el bróker, el circo automáticamente recibirá las imágenes de la cámara del móvil.

En la parte del código del *template* de' VideoPage' se han definido todos estos elementos, el título de la pestaña, el *canvas* y sus estilos, los botones y sus características, etc. En la parte de *style* se han determinado los estilos de las pestañas.

En la parte del *script* se han definido los métodos y funciones que deben hacer los botones. Primero, se ha definido el método 'setup' donde se inicializa la conexión con el bróker MQTT. Luego, se han definido las funciones principales, como el 'StartCapture' y el 'StopCapture'.

El método 'StartCApture' inicializa la grabación teniendo en cuenta si la cámara es delantera o trasera, a continuación, llama a una API de navegadores que solicita el acceso a la cámara y se inicializa el *canvas* donde posteriormente se ubicará la grabación. Si todo se ejecuta correctamente y no sale ningún error, se llama a otra función llamada 'initCanvas'.

La función 'initCanvas', inicializa el *canvas* y establece las características de este. Captura cuadros de vídeo en intervalos de 500 milisegundos (0,5 segundos). Luego convierte el contenido a imágenes con formato jpeg codificadas en base64 y las publica en el bróker a través de MQTT. Paralelamente, muestra el contenido en el *canvas* mediante la función 'showVideo'.

La función 'showVideo', recoge la imagen que está recibiendo, la procesa mediante la librería de OpenCV y muestra el resultado en el *canvas* de la aplicación.

A diferencia, la función StopCapture() detiene la captura de vídeo deteniendo todos los *tracks* del flujo de imágenes e interrumpe también el intervalo de captura.

Por último, se encuentran dos funciones llamadas 'selectFront' y 'selectBack', que actualizan el valor del booleano 'front' que indica si está seleccionada la cámara delantera o trasera del móvil.

Por otro lado, el código del 'PhotoPage', es muy similar, lo único que cambia es la función 'initCanvas', en la que, en este caso, no existe un intervalo para capturar los cuadros de vídeo, sino que captura un cuadro de vídeo, el que está viendo en el determinado momento que se clica el botón de tomar una captura. Luego lo muestra en el *canvas* tratándolo con formato foto, no con formato vídeo, mediante la función 'showImage'.

CAPÍTULO 6. CIRCO DE LAS IMÁGENES

En este capítulo se detallarán todos los procedimientos y códigos implementados para alcanzar el objetivo principal de este proyecto: la creación de un nuevo circo. Se abordarán las tres partes que conforman este circo, se explicará el código empleado y se presentarán los resultados obtenidos.

6.1. Introducción

Como se ha mencionado anteriormente, el propósito central de este proyecto es establecer un nuevo circo; 'El circo de las Imágenes'. En este contexto, se busca trabajar con una variedad de imágenes, vídeos, panorámicas, etc., integrándolas con drones.

El objetivo principal de este circo es que los usuarios que estén conectados a la web App puedan recibir y descargar las imágenes capturadas por la cámara del dron, independientemente de su formato.

Para cada uno de estos casos, se ha implementado el código necesario para añadirlos al segmento del servidor del circo, y simultáneamente se ha desarrollado un sitio web App para que los usuarios puedan recibir las imágenes en sus teléfonos móviles. La parte correspondiente a la implementación del circo se encuentra disponible en el siguiente repositorio de GitHub: https://github.com/anna22itu/CircoDronesTello.git, mientras que la parte de la aplicación web App se encuentra en otro repositorio: https://github.com/anna22itu/WebAppCircoImagenes.git.

En la parte del circo, para poder implementar esta iniciativa se ha definido un panel principal, donde se encuentran tres botones fundamentales; el botón de configuración, el botón de conexión con el dron y el botón para empezar el espectáculo, tal y como se muestra en la **Fig. 6.1**. Adicionalmente, tiene un *label* que muestra el nivel de batería del dron una vez se establece la conexión.

Una vez se accede al botón de 'Jugar', en la ventana de juego representada en la **Fig. 6.2**, se presentan las tres opciones de entretenimiento: imágenes, vídeos y panorámicas.

En la parte de la web App, se han implementado 3 *tabs*: 'Photo', 'Panoramic', 'Video'. En el archivo 'TabsPage' se define la estructura y el comportamiento de las pestañas de navegación en una interfaz de aplicación, conectando cada pestaña con una página específica dentro de una aplicación.

A lo largo de este capítulo, se explicarán con detalle las tres aplicaciones, cómo ha sido el proceso de implementación, etc., tanto en el circo como en la web App.

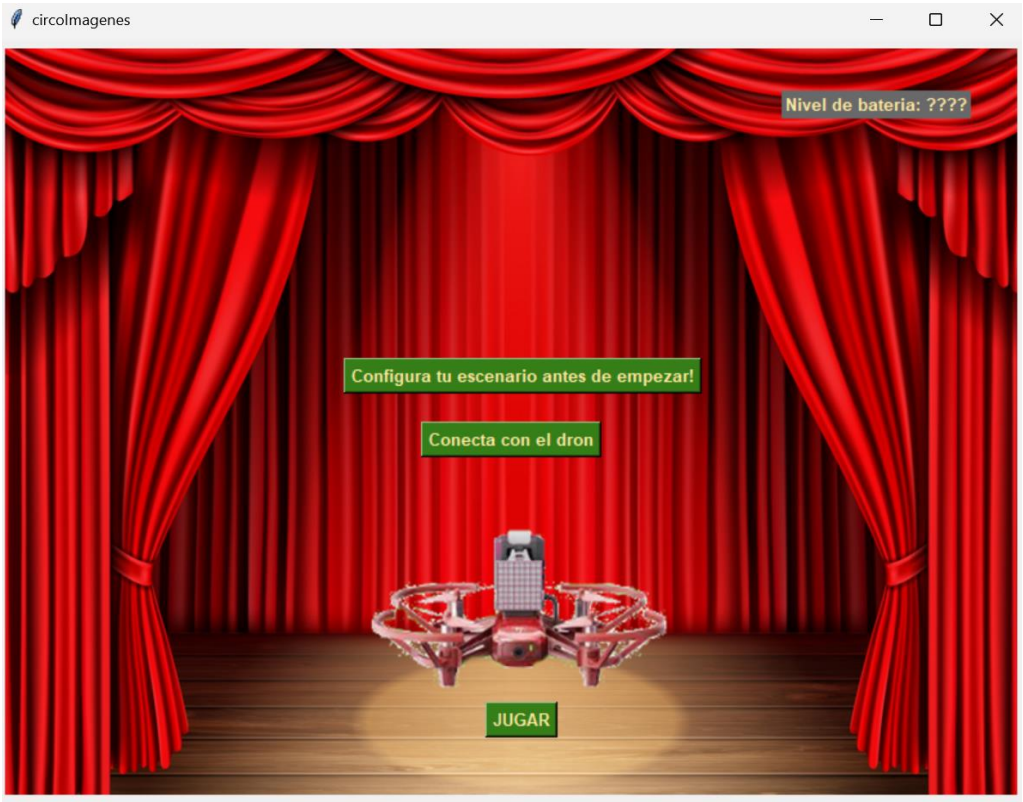

Fig. 6.1. Panel principal del Circo de las Imágenes**.**

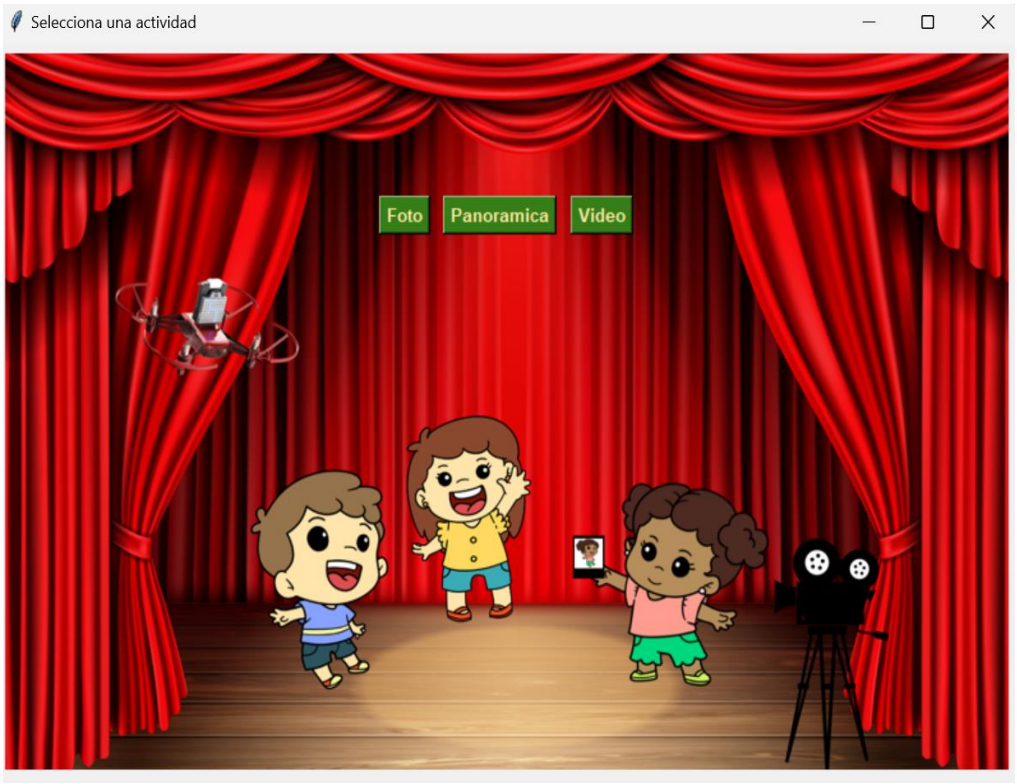

Fig. 6.2. Disposición del panel secundario del circo de las Imágenes.

6.2. Imágenes

La primera opción del Circo de las Imágenes son las fotos. En esta sección, se capturan fotografías mientras el dron está en movimiento, enviándolas a una web App, para que los usuarios puedan recibirlas en sus móviles, y además conseguir crear, tanto en el circo como en la web App, una galería de imágenes donde se puedan visualizar todas las fotos tomadas durante del juego. Esta opción se configura todo en una nueva clase llamada 'escenarioImagenes'.

6.2.1. 'Imágenes' en el Circo

Para la creación de la sección de las imágenes en el circo, se ha desarrollado un panel principal, tal y como se muestra en la **Fig. 6.3**. Este panel está dividido en dos secciones: la zona de control del dron y la sección de tratamiento de imágenes. También cuenta con botones para acceder a la galería y cerrar la ventana.

El objetivo de esta sección es capturar imágenes mientras el dron se encuentra en modo vuelo, visualizarlas en el *canvas* y posteriormente enviarlas a la web App para que los usuarios puedan verlas. Además, ofrece la opción de visualizar todas las imágenes tomadas durante el juego en la galería y volver a enviarlas.

En la clase 'escenarioImagenes', en primer lugar, se definen todos estos nuevos elementos, la conexión MQTT necesaria para la comunicación entre circo y web App, variables iniciales y la estructura de este panel.

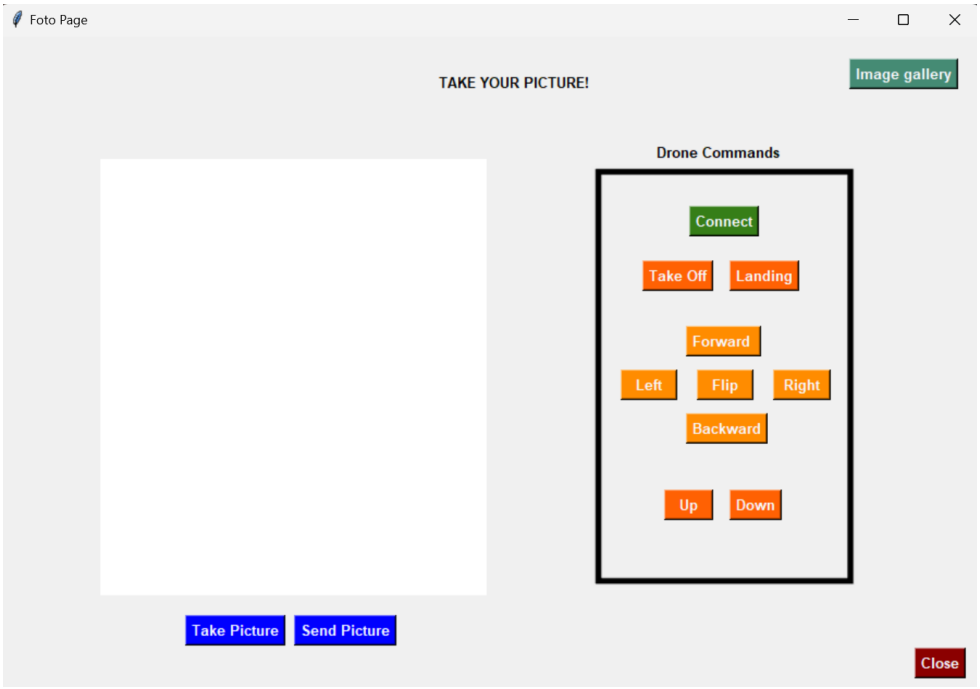

Fig. 6.3. Panel principal de las imágenes del Circo de las Imágenes

En el panel de control de comandos del dron, se han incluido todos los botones necesarios para controlar el dron de manera segura, como el de la conectividad, el de despegue, aterrizaje, delante, atrás, entre otros. Se han definido todos los métodos que permiten tener estas funcionalidades (‘takeoff’, ‘flip’, ‘left’, ‘right’, etc.).

La función ‘connect’, establece la conexión entre el dron y el circo, y es imprescindible para poder continuar con el juego, además, al iniciar la conectividad con el dron, se pone en marcha un hilo de ejecución (*Threat*) que ejecuta de manera paralela la función ‘detecting’.

La función ‘detecting’, inicializa la transmisión en *streaming* de la cámara del dron, crea una nueva ventana, captura estas imágenes y las muestra en ella. Esto permite visualizar en todo momento lo que la cámara del dron está transmitiendo, una vez establecida la conexión, facilitando la elección del momento adecuado para tomar la foto deseada.

En la parte izquierda del panel, se encuentra la interacción con las imágenes: se muestra un *canvas* donde se mostrarán las imágenes tomadas, un botón ‘Take Picture’ que captura la imagen y la muestra en el *canvas*, y otro botón ‘Send Picture’ que guarda la imagen en una carpeta interna del proyecto y la publica a un bróker MQTT, permitiendo posteriormente su visualización en la web App.

La función ‘takePicture’ captura una imagen de la cámara del dron y lo muestra en el *canvas*. Este proceso lo logra obteniendo los fotogramas de la cámara del dron, convirtiéndolos en imágenes, redimensionándolos, cambiándolos en un formato compatible con Tkinter, y luego mostrándolos en el *canvas*, borrando previamente cualquier contenido previo del *canvas*. Además, incrementa el valor en uno de la variable ‘count’ definida inicialmente con valor de 0, que permitirá más adelante llevar un registro de las fotos capturadas. También, actualiza la variable ‘takePictureBool’ a *True*, que permite tener un control de si hay imágenes capturadas o no en el *canvas* para poder enviarlas a la web App de manera segura.

Por otro lado, está la función ‘sendSave’ que solo se ejecuta si hay una imagen capturada y mostrada en el *canvas*, y lleva a cabo dos operaciones: guardar la imagen en una carpeta interna y enviar la imagen a la web App.

Para guardar la imagen, primero las almacena en un vector de imágenes, que servirá para tener ubicadas todas las fotos tomadas. Luego, convierte el fotograma de un espacio de color BGR a RGB utilizando la biblioteca OpenCV. Finalmente, se guarda la imagen capturada en formato JPEG en una carpeta interna mediante el comando:

```
>>cv2.imwrite("assets/telloImages/foto_tello"+str(count) + ".jpg", frame_brg).
```

Para enviar la imagen a la web App, la función 'sendSave' llama a la función 'sendmqtt', donde se genera una ruta de archivo basándose en el 'count' y abre el archivo de imagen en modo de lectura binaria mediante el comando siguiente:

```
>>with open (image_path, "rb") as image_file.
```

A continuación, lee los datos binarios ("rb") del archivo de imagen y los almacena en la variable 'image_data'. Luego, codifica los datos de la imagen a Base64 y los publica en el topic "ImagenAnna" del cliente MQTT definido e inicializado previamente.

Adicionalmente, el botón 'Image gallery' abre una ventana nueva dónde se muestran todas las imágenes guardadas y enviadas. Para ello, se ha creado una nueva clase llamada 'GalleryMedia'. Este panel, también ofrece la opción de volver a enviar las fotos de la galería, mediante el botón de 'SendGallery'.

Esta nueva clase recibe como input el tipo de imagen con el que trabajará, ya sean fotos o vídeos, y ejecuta la visualización de los archivos de una manera u otra, respectivamente. En la parte inicial del código se definen todos los componentes de la galería y los métodos que la conforman.

Al inicializarse la galería, siempre y cuando haya archivos que visualizar, se llama a una función llamada 'create_vector' en la que se rellena un vector de imágenes con las URL de todas las imágenes que se habían almacenado previamente en la carpeta interna 'assets' del proyecto. En este caso se trabajará con imágenes de formato JPG.

Posteriormente, se llama a la función 'create_gallery' en la que se van cargando todas las imágenes que se quieren mostrar en la galería. Para ello, primero, se definen unos parámetros iniciales, como el tamaño de las imágenes, el espacio entre imágenes, etc., luego se itera a través del vector de imágenes creado en la función 'create_vector_images' y carga cada imagen con la ruta específica en el vector.

A continuación, se reajustan algunas características de las imágenes como el tamaño, formato, etc., se crea un *canvas* para cada imagen y se colocan en cada uno de ellos. Para finalizar, se genera un evento (clic del ratón) con el que se permite seleccionar cualquier imagen de la galería e identificar cada imagen con facilidad, gracias al *integer* 'number_img', actualizado previamente. Este evento llama a la función 'select_image' que guarda la URL de la imagen seleccionada en una variable llamada 'selected_image'.

Por otro lado, tiene un botón de 'Send Picture', que permite volver a enviar la imagen seleccionada de la galería al bróker MQTT. Este botón llama a la función 'sendGallery', que se ejecuta solo si hay una imagen seleccionada. Para enviar la imagen al bróker, sigue el mismo procedimiento que la función 'sendSave'; primero se abre y se lee el archivo en modo de lectura binario ("rb"), se almacena en una variable, se convierten los datos de la imagen a

Base64, y luego se publican en el cliente MQTT correspondiente mediante un *topic* específico.

También, se ha añadido un botón ‘Close’ que cierra la transmisión de imágenes de la cámara del dron y cierra la pestaña de ‘Gallery’ de ‘Fotos’.

6.2.2. ‘Imágenes’ en la web App

Por otro lado, se ha creado una web App que permita la visualización de las imágenes enviadas por el circo, así como ver la galería de imágenes recibidas. Esta web App, representada en la **Fig. 6.4**, está compuesta por varios botones que facilitan la manipulación de las imágenes que llegan del bróker y que se muestran automáticamente en pantalla.

En primer lugar, los botones de ‘Forward’ y ‘Backward’ que permiten visualizar la imagen anterior y posterior, respectivamente. En segundo lugar, el botón de la ‘Galería’ posibilita ver todas las imágenes recibidas del bróker. Finalmente, el botón ‘Descargar’ que permite descargar en el teléfono móvil la imagen que se está visualizando la pantalla del usuario.

Fig. 6.4. Ventana las imágenes de la web App

El código de la web App, desarrollado en Ionic y Vue, se divide en tres secciones: el *template* donde se definen todos los componentes que forman la web App, mencionados anteriormente; el *script* donde se establecen los métodos necesarios para el correcto funcionamiento; y el *style* donde se determinan los estilos de los elementos.

Los métodos definidos en el *script* que permiten la perfecta interacción de la web App son los siguientes: 'mostrarImagen', 'decargarImagen', 'showGallery', 'goBack', 'showPrevious' y 'showNext'.

El método 'mostrarImagen' se ejecuta automáticamente una vez montado el componente, se suscribe al bróker y registra el evento para recibir el mensaje publicado en el circo. Las imágenes se muestran instantáneamente a los usuarios que están conectados a la web App.

Las URL de las imágenes se almacenan en un vector de *strings*, después de ser enviadas al bróker, lo que permitirá exponer el conjunto de imágenes recolectadas durante el juego. Mediante el método 'showGallery', se leen todas las imágenes del vector y se visualizan de manera ordenada en pantalla, ocultando el resto de botones.

Por el contrario, si regresar a la configuración inicial de los botones desde la galería, se utiliza el botón 'Back' que activa el método 'goBack' y devuelve a la configuración inicial.

El método 'descargaImagen' realiza la descarga de la imagen en formato PNG. Comprueba la URL de la imagen, define un elemento de anclaje 'a' con la URL, y define el atributo 'download' y luego lo activa emulando un clic para iniciar la descarga.

Por último, los métodos 'showPrevious' y 'showNext' permiten visualizar las imágenes anterior y posterior, respectivamente. Utilizando el vector de imágenes, mencionado previamente, y ajustando el índice del vector, se consigue este efecto de desplazamiento hacia adelante o hacia atrás en la galería.

Gracias a estas implementaciones se consigue uno de los objetivos del proyecto; capturar imágenes en tiempo real de la cámara del dron y enviarlas a la web App para que los usuarios conectados la visualicen y puedan descargársela.

6.3. Panorámica

La segunda opción del 'Circo de las Imágenes' son las panorámicas. El objetivo principal de este apartado es colocar a los usuarios en una fila recta en la pared, mientras el dron vuela por delante haciendo fotos a una cierta altura. Luego se creará una foto panorámica con todas las fotos capturadas y se enviará esta captura mediante un bróker a la web App para que los usuarios reciban la foto panorámica.

Para ello, se utilizará el algoritmo Stitcher, capaz de generar una foto panorámica a partir de la captura de una secuencia de imágenes, el cual se explicará a lo largo de este apartado.

6.3.1. ‘Panorámica’ en el Circo

El panel principal de la ‘Panorámica’ se representa en la **Fig. 6.5**, y se ha creado una clase nueva para implementar toda esta sección llamada ‘escenarioPanoramica’.

Este panel tiene una parte de control del dron que es la misma que el apartado de ‘Foto’, donde se encuentran todos los comandos necesarios para controlar el dron y establecer la conexión con él. En la parte inicial de la clase, se definen todos estos métodos de control del dron, la conexión MQTT con el bróker y variables iniciales.

De la misma manera, la función ‘connect’ establece la conexión con el dron y, paralelamente, inicializa un hilo de ejecución (*Threat*) que llama a la función ‘detecting’ y muestra en todo momento lo que la cámara del dron está transmitiendo. La conexión con el dron y el despegue son condiciones que se deben cumplir para tomar la foto panorámica. Adicionalmente, en esta sección es necesario introducir como *input* la distancia que se desea recorrer durante la captura de imágenes.

Además, aquí no se trabajará con la galería de fotos, sino que solamente se recibirá la foto panorámica en la web App, y los usuarios, si lo desean, podrán descargársela.

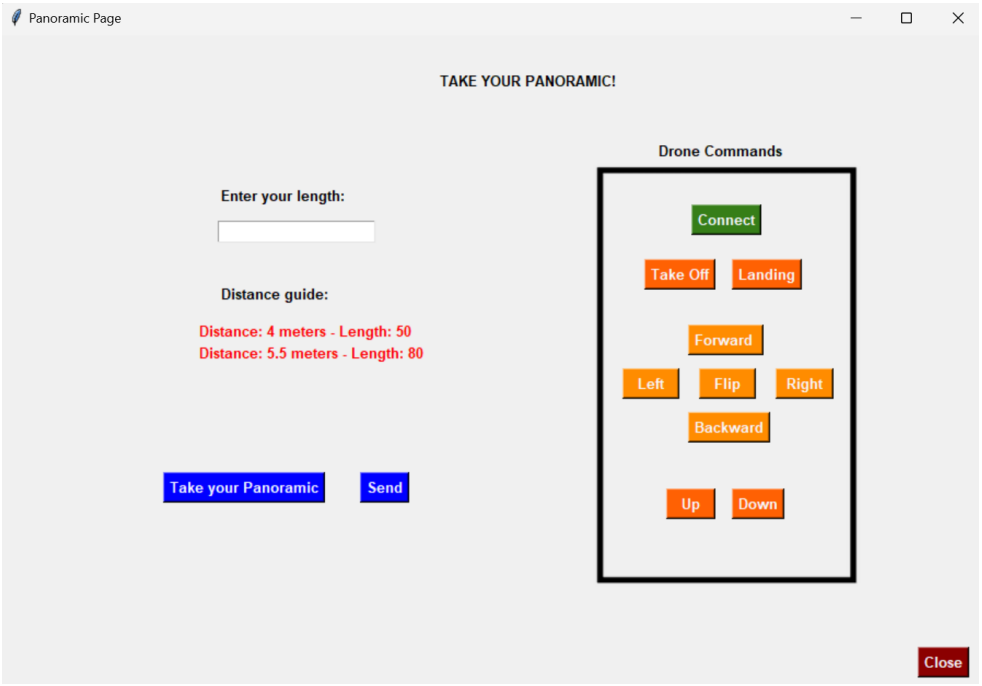

Fig. 6.5. Panel principal de las panorámicas del Circo de las Imágenes.

El panel de ‘Panorámica’ cuenta con dos botones diferentes a los de ‘Foto’. El botón ‘Take your Panoramic’ y el botón de ‘Send’; que permitirán tomar todas las imágenes en secuencia, crear la panorámica, recortarla y enviarla al bróker

para que los usuarios la reciban en la web App de manera segura. El dron recorrerá los metros introducidos como *input* para crear la panorámica.

Cuando se selecciona el botón 'Take your Panoramic', se ejecuta el método 'play_panoramic' de una nueva clase llamada 'pano_configuration', donde se define todo el proceso de implementación para crear la foto panorámica. Esta nueva clase tiene varias funciones implementadas; 'Open', 'record_pano' y 'makePano' y 'callSlider'.

La función 'Open' define la condición de creación de panorámicas, define el valor de 'length' con el parámetro de entrada, crea una instancia de la clase 'slider', explicada con más detalle más adelante, y llama a la función 'record_pano', cuyo elemento de entrada es el dron.

En la función 'record_pano' se ejecutan todos los comandos del escenario para poder tomar todas las imágenes secuenciales que formarán la panorámica. Primero, se definen unas variables como el vector donde se irán acumulando las imágenes tomadas, la ruta donde se almacenarán estas imágenes, la URL en la que se guardarán las fotos, entre otros. También se define una variable de tipo *integer* llamada 'interval' que permitirá tener un control de las fotos tomadas.

Posteriormente, bajo la condición de poder realizar panorámicas, se inicia un bucle *while* activo mientras 'interval' sea menor que 'length', donde el dron captura imágenes cada 5 unidades de intervalo. Para cada intervalo se obtiene una imagen de la cámara del dron, la convierte de formato RGB a BGR y lo guarda en la dirección especificada. A continuación, lo almacena en el vector, y paralelamente se le envía al dron el comando para mantenerse en movimiento con una velocidad de 30 hacia la derecha, consiguiendo que el dron se mueva hacia la derecha mientras tomas las imágenes cada cierto tiempo.

Una vez tomadas y guardadas todas las imágenes, se aterriza el dron antes de crear la panorámica y recortarla con las funciones 'makePano' y 'callSlider' respectivamente.

La función 'makePano' mediante el uso del algoritmo Stitcher crea la panorámica a partir de todas las imágenes tomadas previamente, lee las imágenes guardadas en la carpeta interna y las guarda en un vector. Luego, se procesan con el algoritmo y se crea una nueva imagen panorámica con la unión de estas, guardada, también, en una carpeta interna.

Por último, en la función 'callSlider', se llama a una nueva clase llamada 'slider' que permitirá recortar la panorámica mediante diferentes comandos. Esta clase carga la imagen panorámica creada previamente y le añade un marco rojo a su alrededor mediante la función 'agregar_marco_rojo'. Mediante cuatro métodos distintos de 'actualizar_borde', se logra ajustar el grosor de este marco. Se ajustan las medidas del borde hasta conseguir no visualizar ninguna zona de la imagen en negro. Posteriormente, se recorta la imagen según los valores de grosor establecidos previamente y se descarga la imagen. La imagen

descargada contendrá el marco rojo a su alrededor, no tendrá ninguna zona negra y tendrá un mensaje de despedida encima de ella.

Por otro lado, tenemos la función ‘send_panoramic’ que se ejecuta cuando se clica en el botón ‘Send’ y envía la foto panorámica al bróker. Primero, se define un valor de calidad para controlar el tamaño de la imagen y hacer más eficiente la transmisión. En este caso se trabajará con un valor de calidad de 50, para conseguir precisión y resolución en la imagen sin sobrecarga ni alto tiempo de transmisión. A continuación, se lee el archivo en formato binario ‘rb’, se codifica primero con el parámetro de calidad y posteriormente en formato base64, y se envía al bróker con el tema especificado.

Con esta implementación concluida, hemos establecido las bases para la captura de imágenes panorámicas en el circo. Ahora, se puede avanzar hacia la segunda parte para la implementación de la web App.

6.3.2. ‘Panorámica’ en la web App

Paralelamente, se ha implementado la pestaña de la panorámica en la interfaz web App, que se representa en la **Fig. 6.6**. En ella, se observa que tiene un botón ‘Descargar Panorámica’ que permite la descarga de la imagen panorámica enviada por el ‘Circo de las Imágenes’ a la web App.

Fig. 6.6. Ventana las panorámicas de la web App

Al igual que todos los códigos de web App, el código de la ‘Panorámica’ se divide en tres secciones; el *template*, el *script* y el *style*.

En la sección del *template* se define este botón de descarga y elementos adicionales que permitirán el dinamismo de la página.

Por otro lado, en la sección de *script,* se definen los métodos 'mostrarImagen' y 'descargarImagen'.

El método 'mostrarImagen' tiene como finalidad recibir la panorámica y mostrarla en pantalla. Para ello, se suscribe al bróker con un tema determinado, registra eventos con los mensajes recibidos y crea una URL de la imagen a partir del mensaje en base64 en formato JPEG. Posteriormente, asigna esta URL a un elemento predefinido de tipo 'Imagen' en el *template*, permitiendo su visualización.

El método 'descargarImagen' permite la descarga de la panorámica mediante la asignación de la URL a un elemento de anclaje 'a' creado previamente. Además, se le asignan también propiedades como 'download' o 'click'.

En la sección de *style* se definen todos los estilos y formatos de estos elementos.

Con esta implementación de la funcionalidad panorámica, se consigue alcanzar otro de los objetivos principales de este proyecto, consiguiendo trabajar con diferentes tipos de imágenes y con diferentes tipos de dinámicas e interacciones.

6.4. Vídeo

En la última sección del 'Circo de las Imágenes', se quiere trabajar con los vídeos y lograr visualizar tanto la transmisión en vivo de las imágenes de la cámara del dron en la web App, como los archivos de vídeo que se han ido grabando durante la ejecución. Para ello se ha creado una nueva clase llamada 'escenarioVideo' en la que se determinan todos los elementos y funciones que lo forman.

6.4.1. 'Vídeo' en el circo

En la **Fig.6.7**, se puede observar la disposición del panel del 'Video' del 'Circo de las Imágenes'. Tiene el mismo conjunto de comandos para el control del dron que la actividad de 'Foto' y 'Panorámica', a la derecha del panel.

Al iniciarse la conexión se inicializa la transmisión en *streaming* de la cámara del dron en una ventana nueva, mediante la función 'detecting'.

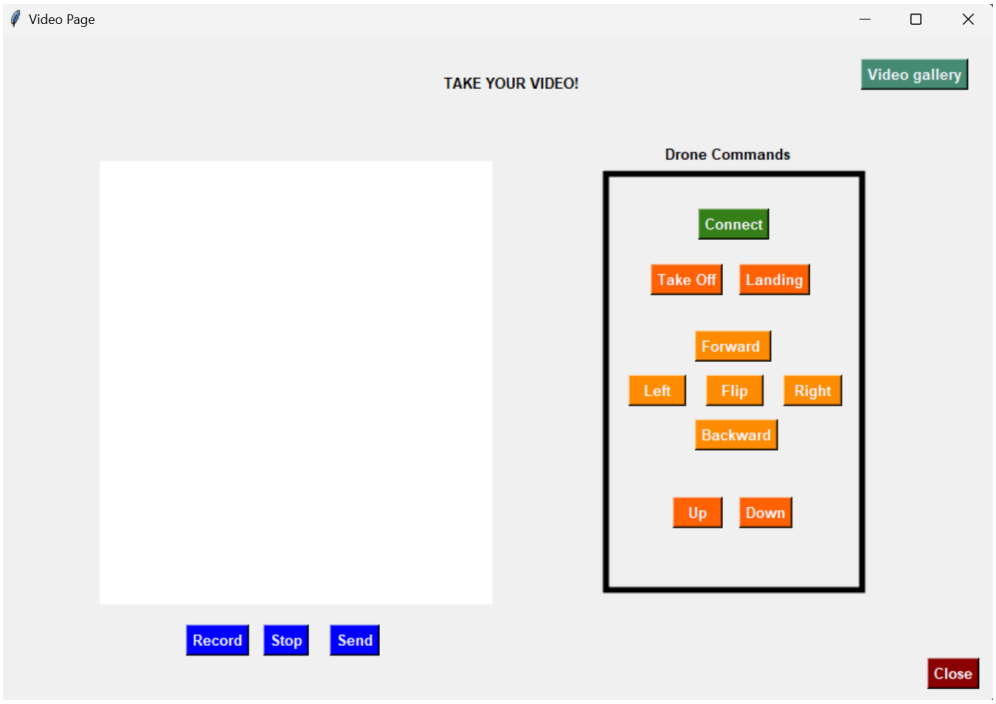

Fig. 6.7. Panel principal de los vídeos del Circo de las Imágenes**.**

En la izquierda del panel, se encuentran tres botones: ‘Record’, ‘Stop’ y ‘Send’ encargados de gestionar las dinámicas del juego.

El botón ‘Record’ muestra en el *canvas* la transmisión en *streaming* de las imágenes de la cámara del dron mediante la función ‘recordVideo’. En esta función, se define un vector de imágenes ‘video_frames’ utilizado más adelante, se actualiza la condición que indica si se está capturando o no, y se ejecutan dos hilos de ejecución (*Threat*): uno para la función ‘recordingVideo’ y otro para ‘recordMQTT’.

El método ‘recordingVideo’ activa la opción de ‘enviar’ los *frames* a la web App y se ejecuta un bucle siempre que se esté capturando. En este bucle, se capturan los cuadros de vídeo de la cámara del dron, se almacenan en el vector ‘video_frames’, se redimensionan y se muestran en el *canvas*. Para acabar, el bucle añade un retraso de 0.10 segundos.

Por otro lado, el método ‘recordMQTT’ inicializa un bucle que se ejecuta mientras la condición de ‘enviar’ sea *True*, y publica en el bróker MQTT los *frames* en *streaming*.

Debido a que los brókeres MQTT no están diseñados ni preparados para transmitir mensajes muy grandes ni un gran volumen de mensajes, fue necesario ajustar la calidad en la que se envían los *frames*. Para ello, se definió un parámetro de codificación con su correspondiente variable de calidad, que permitirá controlar el tamaño de las imágenes transmitidas. Tras algunas pruebas, se decidió trabajar con una calidad de imagen de 20, que proporciona la suficiente calidad para que las imágenes lleguen a la web App con buenas

características y, a su vez, un reducido tamaño que permite una rápida transmisión.

De este modo, codifica los *frames* con este parámetro de calidad y luego los codifica en base64, para posteriormente publicarlos en el bróker. Para finalizar, añade un retraso de 0.10 segundos para emitir a la misma frecuencia que la transmisión del vídeo y conseguir una sincronización entre transmisiones.

El botón de '*Stop',* asociado al método 'stopVideo', finaliza los dos *Threats* inicializados previamente en la transmisión y detiene la transmisión tanto en el *canvas* como en la web App, actualizando el valor de las variables 'recording' y 'sending' a *False*.

Por último, el botón 'Send' inicializa el método 'sendVideo'. En este método, ya no se trabaja con todos los *videoframes* en *streaming,* sino con el vector de frames que se han ido añadiendo durante la transmisión. Se almacenan todos los *frames* capturados desde que se hace clic en el botón 'Record' hasta el 'Stop', en un archivo de vídeo en una carpeta interna del proyecto. Para crear este archivo de vídeo, se llama a la función 'save_video', que genera un archivo de formato MP4 donde se guarda el vídeo y crea un objeto 'VideoWrite' donde se escriben los *frames* del vídeo. Posteriormente, la función 'sendVideo', abre el vídeo creado previamente, lo lee en formato binario, lo codifica en formato base64 y lo publica en el bróker para que los usuarios lo puedan visualizar.

Además, similar a las imágenes, cuenta con un botón 'Video gallery' que permite observar todos los vídeos grabados previamente y ser enviados otra vez al bróker. Para ello se ha utilizado la clase 'GalleryMedia', mencionada en el segundo apartado de este capítulo, pero ahora teniendo en cuenta que el tipo de archivos con los que se trabaja son los vídeos. Se utilizan las mismas funciones y se consigue que los usuarios puedan volver a recibir vídeos previamente enviados.

6.4.2. 'Vídeo' en la web App

Por consiguiente, se ha implementado también en la web App una nueva pestaña para interactuar con el vídeo que abarca dos implementaciones diferentes: la recepción de los *videoframes* en *streaming* y la recepción de archivos de vídeos.

Una vez inicializada la captura de vídeo en el circo, transmitida en el *canvas,* se inicia la transmisión en *streaming* en la web App mediante la función 'recordMQTT', mencionada previamente. Con la conexión establecida, los usuarios reciben automáticamente la transmisión en el *canvas* de la web App.

Cuando se finalice la transmisión en vivo desde el circo, mediante el botón 'Stop', se abre la posibilidad de visualizar y descargar el archivo de vídeo en la web App, enviado desde el circo.

Para ello, se ha añadido en la web App un botón llamado ‘Video’ en la pantalla principal, que se utilizará para visualizar los archivos de vídeo recibidos en el bróker, que habilita dos botones adicionales: ‘Back’ para regresar a la transmisión en *streaming* y ‘Descargar’, que permitirá la descarga del archivo de vídeo. En el *template* se definen todos estos elementos de la pestaña.

En la parte del *script* se definen los métodos que permiten dar funcionalidad a estos elementos. Una vez se configura la web App, por defecto, se ejecutan dos métodos: ‘mostrarVideo’ y ‘recibirVideo’, definidos en el ‘setup’ junto con las variables iniciales.

El método ‘mostrarVideo’ se encarga de la transmisión en *streaming* de los *frames* que recibe del circo de la cámara del dron. Primero, se suscribe al bróker mediante un determinado ‘topic’ y registra eventos para gestionar los mensajes recibidos. A continuación, crea una URL de la imagen a partir del mensaje en base64 en formato JPEG, crea un elemento de imagen, ajusta las propiedades tanto del *canvas* como de la imagen y luego muestra la imagen en el *canvas* de la web App.

Por otro lado, el método ‘recibirVideo’, cuya finalidad es recibir el archivo de vídeo, se suscribe al bróker con otro ‘topic’ diferente, donde se publican los archivos de vídeo desde el circo y registra eventos para controlar los mensajes recibidos. En este caso, crea una URL a partir del mensaje en base64 en formato MP4. Posteriormente, crea un elemento de vídeo, ajusta algunas variables y lo muestra en la pantalla de la web App, enviando un mensaje de ’vídeo recibido’. Para acceder a este archivo de vídeo se deberá seleccionar el botón de ‘Videos’.

Además, existe la posibilidad de descargar el archivo de vídeo recibido desde el bróker, mediante la función ‘descargarVideo’. En esta función, se crea un elemento de encaje ‘a’, al que se le asignan diferentes características como la URL, el atributo de descargar y el atributo de clic que activa la descarga.

Adicionalmente, se definen los métodos ‘showVideos’ y ‘goBack’ que permiten cambiar la disposición de la pantalla para estar viendo la transmisión en vivo o los ficheros de vídeo.

En la sección de *style* se han ajustado todos los estilos y formatos de todos los elementos de la página de ‘Video’.

Con esta implementación de todas estas funcionalidades, se finaliza el desarrollo del ‘Circo de las Imágenes’, logrando cada uno de los objetivos planteados inicialmente. Se ha logrado desarrollar una plataforma que interactúa con tres tipos de imágenes diferentes y permite su transmisión a una web App.

En el siguiente capítulo, se reflexionará sobre estas implementaciones y sobre los procesos de pruebas que han tenido que pasar estos códigos para su perfecta implementación.

CAPITULO 7. PRUEBAS Y VALIDACIONES

Antes de dar por finalizado el código, es crucial llevar a cabo pruebas para verificar y validar el correcto funcionamiento de las implementaciones realizadas. En este capítulo, se describirá de manera detallada cómo se han llevado a cabo las evaluaciones y pruebas en cada aspecto del proyecto.

7.1. Introducción

Primero, se realizan pruebas generales para verificar el buen funcionamiento del código; mediante la ejecución del código, pasando por todos los botones, sometiéndolo a todas las opciones posibles y analizando y corrigiendo los errores obtenidos durante el proceso. Este proceso ha permitido hacer más robusto el código, consiguiendo mayor capacidad para gestionar los errores, seguridad, adaptabilidad a los cambios, y mayor eficiencia.

En segundo lugar, se realizan una serie de evaluaciones por parte de usuarios externos al proyecto, a los que se les invita a interactuar con el circo, realizar pruebas y proporcionar *feedback* sobre la facilidad, entretenimiento, dinamismo e interactividad del código. Los resultados de estas evaluaciones fueron satisfactorios, evidenciando que los usuarios comprendieron rápidamente la dinámica del juego.

Una vez finalizado este proceso de depuración de código y asegurado su correcto funcionamiento, con los resultados esperados, se realizan pruebas más específicas según las necesidades de cada una de ellas, explicados en los siguientes subapartados.

Además, mencionar también, que, en la parte de recepción y transmisión de imágenes por la web App, tanto del Capítulo 5 como del Capítulo 6, se han realizado pruebas de compatibilidad para garantizar que el código funcione de manera efectiva en diversos dispositivos. Debido a la amplia variedad de plataformas, estas pruebas en los dispositivos móviles se centraron en el sistema operativo iOS de Apple y Android. Este análisis de compatibilidad asegura una experiencia de usuario óptima y coherente.

Por último, todas las implementaciones han pasado por unas pruebas de código en tierra y otras mientras el dron estaba en modo vuelo. Con estas pruebas previas al vuelo se ha querido evitar colisionar y provocar accidentes, asegurándose de que todo funciona correctamente antes de volar el dron. Una vez todo funcionaba como se esperaba, se realizaron las pruebas en vuelo.

7.2. Pruebas de la integración de la transmisión del circo con el móvil

En este capítulo, se ha introducido una nueva funcionalidad que permite a los usuarios interactuar con el código base utilizando la cámara de sus dispositivos móviles. Para ello, como se ha podido ver detalladamente a lo largo del Capítulo 5.

El reto con el que se ha lidiado en este capítulo ha sido coordinar el mecanismo de elección del tipo de cámara. Para lograr esto, se han añadido los booleanos necesarios para sincronizar los *radiobuttons,* asegurando así una transmisión coherente de la elección realizada por el usuario a lo largo de ejecución del código. Esta coordinación es crucial para garantizar que el sistema responda adecuadamente a la selección de la cámara preferida por el usuario, proporcionando una experiencia de usuario fluida y sin interrupciones.

Además, se enfrentaron desafíos significativos en la transmisión tanto en la web App como en el circo. En la web App, la tarea inicial fue acceder a la cámara del dispositivo móvil, logrando el éxito mediante diferentes pruebas. Por otro lado, en la parte del circo, se experimentaron problemas de transmisión, ya que las imágenes no llegaban con sincronización al circo, presentando un retraso considerable que afectaba a la dinámica del juego. Sin embargo, se logró superar estos obstáculos y optimizar la transmisión.

Por otro lado, a diferencia del Capítulo 6, como se detallará a continuación, en este apartado no ha habido problemas de calidad de imagen ni de tamaño, por lo que no ha sido necesario codificar las imágenes teniendo en cuenta un parámetro de calidad.

En el Anexo C.1, ubicado al final de esta memoria, se presentan los resultados obtenidos. En la **Fig. C.1**, se muestra la imagen de la cámara del móvil mostrada en la web App. A continuación, esa imagen es transmitida al circo, tal y como se ve en la **Fig. C.2**, la cual se recibe correctamente. Una vez, definidas todas las poses con las que se desea trabajar, por ejemplo, las de la **Fig. C.3**, se puede iniciar el juego. La **Fig. C.4** presenta el panel de las poses guardadas previamente y la opción de juego de ‘Practicar’. Al realizar la práctica de la pose 1, se obtiene el resultado de la **Fig. C.5**.

En este caso se han mostrado los resultados con los dedos, pero para las poses de medio cuerpo sería el mismo procedimiento.

7.3. Pruebas del Circo de las Imágenes

En el Capítulo 6, como se ha explicado previamente, se describe la creación del ‘Circo de las Imágenes’, el cual se divide en secciones dedicadas a Imágenes, Panorámica y Vídeos. Se realizaron diversos tipos de pruebas y comprobaciones para asegurar el correcto funcionamiento de cada una de estas opciones

7.3.1. Imágenes

Un desafío general en el proyecto ha sido la pérdida de calidad de las imágenes durante la transmisión. Esto ocurre debido a que el bróker MQTT utilizado en este proyecto no es eficiente trabajando con un gran volumen de imágenes y alta calidad. Para combatir con ello, se han codificado las imágenes teniendo en cuenta un factor de calidad.

Sin embargo, en el caso de las fotos no ha hecho falta trabajar con este factor, ya que al transmitir un único mensaje con formato imagen, el bróker lo toleró adecuadamente. Más adelante se detallará la razón por la que fue necesario emplear este enfoque en los apartados de vídeos y panorámicas.

En esta aplicación, las pruebas y test de comprobación de código, así como la transmisión de imagen, resultaron satisfactorias desde el principio. Una vez implementado el código, las imágenes llegaron a la web App con buena calidad y en un tiempo aceptable, entre otros aspectos positivos. Por lo tanto, no fue necesario realizar pruebas adicionales.

En el Anexo C.2.1, se pueden observar los resultados obtenidos. En la **Fig. C.6**, se muestra la captura de la transmisión de la cámara del dron. La **Fig. C.7** presenta en el *canvas* la captura tomada tras clicar el botón 'Take Picture'. Por último, en la **Fig. C.8** se aprecia, la captura enviada por el circo a la web App y recibida en el móvil. Se observa una buena calidad de imagen, sin embargo, se aprecia una leve distorsión de la imagen, en cuanto a distribución de tamaño. Este fenómeno de sebe al redimensionamiento que han experimentado las imágenes para garantizar su correcta visualización.

Como se ha mencionado en apartados anteriores, se comprobó, también, la compatibilidad con diferentes dispositivos, obteniendo una respuesta satisfactoria.

7.3.2. Panorámicas

En la sección de las fotos panorámicas, se abordaron varios desafíos fundamentales, entre los cuales destacan el ajuste del escenario de trabajo, la implementación del algoritmo Stitcher para la creación de panorámicas y la gestión del tamaño resultante de las panorámicas.

La creación del escenario de trabajo fue un componente crítico para el éxito de esta operación. Antes de empezar con las pruebas de vuelo y dinámicas relacionadas con la creación de la panorámica, se realizó un estudio previo cuyo objetivo principal era determinar variables de configuración del dron. Este análisis incluyó factores como distancia a la que se debe colocar el dron de la pared y de los usuarios para tomar la foto correcta, la velocidad de vuelo, la frecuencia de captura de imágenes, el solapamiento deseado, entre otros.

Para hallar la distancia a la que se debe colocar el dron de la pared, se han hecho una serie de verificaciones en las que se evaluaba el área fotografiada,

para asegurar que los usuarios se ajustan perfectamente en la foto. Se ensayaron diferentes distancias, analizando las dimensiones del área de las fotos y observando los cambios. Después de estas pruebas se concluyó que debe de haber una distancia de 1,5 metros aproximadamente entre la pared y el dron, asumiendo que los usuarios se encuentran a uno 20 centímetros de la pared.

Para determinar el solapamiento óptimo entre imágenes, se implementó un código en Python separado del circo para realizar estas pruebas. Con unas fotos sacadas de internet que simulaban la fila de personas, pasándolas por el algoritmo Stitcher, se analizaba cómo afectaba el solapamiento en la fusión de las imágenes y la calidad del resultado final.

Se concluyó que el algoritmo Stitcher no trabaja bien con un solapamiento menor al 50%, ya que presenta dificultades para encontrar los puntos en común entre imágenes, haciéndolo insuficiente. Por otro lado, con solapamientos mayores al 50%, se comprobó que el algoritmo funcionaba correctamente, ya que había más áreas comunes entre las imágenes para encontrar correspondencias y tenía más facilidad para unirlas. Sin embargo, a pesar de que un mayor solapamiento efectuaba panorámicas con un resultado final con mayor precisión y calidad, un mayor solapamiento también conllevaba una mayor cantidad de imágenes innecesarias para la creación de la imagen panorámica final. Al aumentar el solapamiento, se obtenía mayor carga computacional y mayor gasto de procesamiento innecesario.

Tras este estudio, se determinó que el solapamiento del 50% era el más efectivo y óptimo para crear las fotos panorámicas.

Posteriormente, se determinó una velocidad fija y estable, no muy rápida para que el dron no se desestabilizase, de 30 unidades. A partir de entonces, empezaron las pruebas de vuelo para poder determinar la frecuencia de capturas de los fotogramas desde la cámara del dron y el intervalo de para lograr una sincronización adecuada entre el número de fotos y la distancia entre ellas.

Se empezaron las pruebas con una distancia definida de 50, con una frecuencia de captura de imagen de 0.5 segundos y con un intervalo de 10, pero se captaron pocas imágenes muy parecidas. Por lo que, se optó por aumentar la distancia a 100 con una frecuencia de 2 segundos con el mismo intervalo, y se consiguió aumentar el número de imágenes, aunque las fotos se tomaban con mucha separación. Por último, se redujo la frecuencia a 1 segundo, con un intervalo de 5 y una distancia de 100, logrando una sincronización entre el número de fotos y la distancia entre ellas.

Paralelamente, se evaluó con el algoritmo Stitcher su eficiencia para procesar un gran número de imágenes. Se comprobó que el algoritmo Stitcher tolera un máximo de 20 a 25 imágenes con un tiempo de procesado de 20 segundos aproximadamente. Por ello, se le ofrece al usuario la opción de elegir la distancia que desea capturar, considerando que un *length* menor a 80 asegura

un rendimiento óptimo del algoritmo y que el algoritmo no tendrá problemas de ejecución.

Mencionar también la importancia de proporcionar información visual al dron para mejorar su estabilidad y capacidad de navegación. Al proporcionarle al dron información visual, como patrones o dibujos tanto en el suelo como en la pared, se consiguen mejores puntos de referencia visuales que le permiten tener una percepción más clara de su entorno y así, tener un vuelo más estable y preciso.

En la **Fig. C.9** del Anexo C.2.2, se muestra el escenario que se utilizó para las pruebas y el ruido visual que se empleó para lograr la estabilidad del dron. Se necesitó ruido visual tanto para la cámara frontal del dron como para la cámara inferior del sistema de posicionamiento visual.

Una vez montado el escenario, como se ha mencionado anteriormente, se realizan las pruebas de vuelo para diferentes distancias y situaciones. En la **Fig. C.10** del Anexo C.2.2, se puede observar el resultado de la panorámica a una distancia de 5.2 metros (*length*: 80). En la imagen, se aprecian unas zonas negras, lo cual puede ocurrir por diversos motivos, como el desalineamiento del dron con respecto a la pared en algunos momentos, falta de ángulo de visión por parte de la cámara del dron, problemas de superposición de las imágenes, entre otros.

Ante estos resultados, se decidió implementar un código adicional que permitiese recortar las zonas negras de las panorámicas. Como se expuso en la sección 6.3.1, se desarrolló la clase 'slider', donde se hacía la apertura de la imagen panorámica junto con un control deslizante de bordes, como se muestra en la **Fig. C.11**. Una vez ajustados los bordes rojos, que se irán ensanchando según las regulaciones, se procede al recorte de la imagen. Las **Fig. C.12** y **Fig. C.13**, ejemplifican una panorámica antes y después de ser recortada, respectivamente.

Por último, la imagen se envía a la web App donde el usuario la recibe de forma automática. Hay que destacar también, que en el caso de la panorámica fue necesario el uso del factor de calidad de imagen para reducir su tamaño y favorecer la transmisión, dado que las panorámicas tienen dimensiones mayores y requieren mayor resolución.

7.3.3. Vídeos

En la parte del vídeo, durante la transmisión de los *frames* en *streaming*, al igual que con las imágenes, fue necesario trabajar con el factor de calidad. Esto se debió a la necesidad de manejar muchos *frames* de manera rápida y eficaz. Al principio de las pruebas, sin el factor de calidad, no se obtenía una buena recepción de las imágenes debido al gran volumen de imágenes transmitidas a alta velocidad. Sin embargo, al introducir el factor de calidad se pudo controlar tanto el tamaño de las imágenes transmitidas como su frecuencia de transmisión, coordinándolas con la transmisión de imágenes en

el *canvas* del circo. De esta manera, se logró obtener una buena recepción con una calidad satisfactoria.

Por otro lado, en la parte de transmisión del archivo de vídeo, no fue necesario trabajar con este factor de calidad, ya que el bróker no tuvo problemas de transmisión ni de recepción al enviar un único mensaje de vídeo. En la **Fig. C.14**, se muestra la web App al recibir el archivo de vídeo junto con las opciones de reproducir el vídeo, agrandarlo, cambiarle la velocidad, entre otros.

Un aspecto pendiente que se observó durante las pruebas fue la velocidad de reproducción de los archivos de vídeo. Durante las pruebas de ejecución de código y comprobación del funcionamiento, se percibió que el archivo de vídeo recibido en la web App del móvil reproducía a una velocidad más rápida que la velocidad original de envío. Se intentó abordar este problema tanto en la parte de transmisión del circo como en la parte de recepción en la web App, pero no se logró encontrar una solución exitosa.

Esta cuestión queda registrada como una contribución para futuros estudiantes que continúen con el desarrollo del proyecto, brindándoles la oportunidad de explorar posibles soluciones y mejoras en este aspecto específico.

CAPÍTULO 8. INTEGRACIÓN EN EL ECOSISTEMA

En este capítulo se detallará la integración de las implementaciones realizadas en el ecosistema. Se explicarán los diferentes módulos creados en sus respectivos repositorios, su estructura y organización, así como las funcionalidades de cada uno de ellos.

Esta integración está dividida en tres módulos diferentes, creados en tres repositorios diferentes: 'Web App Mobile Tello Drone Circus', 'Circo Drones Tello' y 'Web App Circo Imágenes'.

8.1. Módulo 'Web App Mobile Tello Drone Circus'

El primer módulo es el llamado 'Web App Mobile Tello Drone Circus' y está ubicado en el siguiente repositorio: https://github.com/anna22itu/WebAppMobileTelloDroneCircus.git. En este módulo, se desarrolla toda la implementación de la web App que permite transmitir las imágenes en vivo al circo cuando se elige la opción de 'Móvil'.

Este repositorio está formado por diferentes ficheros y carpetas que forman todo el código. También, incluye un archivo 'Readme.md' donde se documenta el funcionamiento y la estructura del código, así como un archivo '.git' que facilita la gestión y seguimiento de la relación con el repositorio.

Esta web App está formada por tres pestañas: 'Home', 'Video' y 'Foto'. La pestaña 'Video' es la que permite enviar la transmisión en vivo de la cámara del móvil al circo. Está compuesta por un *canvas* donde se muestra la transmisión, un botón que permite seleccionar la cámara con la que se quiere capturar las imágenes del móvil (delantera o trasera) y un botón que inicializa la transmisión. Una vez conectado el circo a la web App, automáticamente, se visualiza la transmisión del móvil en el circo. Por otro lado, las pestañas 'Home' y 'Foto', no facilitan ninguna funcionalidad, fueron creados de modo de prueba, para evaluar dinamismos y detectar errores.

8.2. Módulo 'Circo Drones Tello'

El segundo módulo es el llamado 'Circo Drones Tello', ubicado en el siguiente repositorio https://github.com/anna22itu/CircoDronesTello.git. En este módulo, se desarrollan las nuevas implementaciones del circo, tanto la implementación de la web App que permite la detección con la cámara del móvil como el 'Circo de las Imágenes'.

Este repositorio está formado por todos los ficheros '.py' que permiten el funcionamiento del código y carpetas facilitan la dinámica del circo. Además, al igual que el resto de módulos, tiene sus correspondientes fichero 'Readme.md' y '.git'.

El ‘Circo de drones’ está formado por tres circos: ‘El Circo de los Colores’, ‘El Circo de las Imágenes’ y ‘El Circo de las Poses’. A su vez, cada circo tiene su panel de configuración y de conectividad antes de entrar en la dinámica del juego.

En este proyecto no se ha trabajado con el ‘Circo de los Colores’, por lo que, tal y como se explicó en el Capítulo 2, sigue teniendo tres posibilidades de juego: ‘Match Ball’, ‘Atrapa el Ladrón’ y ‘Sígueme’

El ‘Circo de las Poses’, además de la configuración de escenario y de conectividad, en el panel principal tiene la opción de seleccionar la cámara con la que se trabajará, móvil u ordenador. Por otro lado, tiene dos modos de juego: ‘Crea tus poses’ y ‘Empezar espectáculo’.

En la primera opción, tal y como se observa en la **Fig. 2.3**, hay la posibilidad de crear poses con la mano, con medio cuerpo y con cuerpo entero, para luego detectarlas y practicar o guardarlas y utilizarlas en el espectáculo.

En el espectáculo, existen cuatro opciones: ‘Sígueme’, ‘Dedos’, ‘Poses’ y ‘Caras’. En la opción de ‘Sígueme’, el dron juega a seguir objetos o personas, según el color o la pose. En el resto de las opciones, se controla e interactúa con el dron tanto con los dedos, las poses o las caras, respectivamente.

8.3. Módulo ‘Web App Circo Imágenes’

El tercer módulo es el llamado ‘Web App Circo Imágenes’ y está ubicado en el siguiente repositorio https://github.com/anna22itu/WebAppCircoImagenes.git. En este módulo, se desarrolla una web App que permite toda la interacción con el ‘Circo de las Imágenes’, cuyo objetivo es la recepción de las imágenes, panorámicas y vídeos para los usuarios.

Este repositorio está formado por todos los ficheros que permiten el funcionamiento del código, junto con sus correspondientes ficheros ‘Readme.md’ y ‘.git’.

Está compuesto de tres pestañas: ‘Foto’, ‘Panorámica’ y ‘Vídeo’, que se han explicado a lo largo de todo el Capítulo 6.

CAPÍTULO 9. CONCLUSIONES

En este apartado se van a desarrollar las conclusiones obtenidas de este proyecto, se va a analizar sus objetivos y los resultados obtenidos, así como las futuras posibles contribuciones.

9.1. Conclusiones Técnicas

El objetivo principal de este proyecto es implementar la creación de un nuevo circo y dar continuidad a las ideas de los proyectos que formaron la base de este ecosistema. A lo largo de todo el proyecto, se alcanzaron todos los objetivos propuestos, incluyendo la implementación de las contribuciones pendientes en el código base, la posibilidad de trabajar con la cámara del móvil y la creación del 'Circo de las imágenes' con tres funcionalidades diferentes: foto, panorámica y vídeo.

Tal y como se ha observado durante el proyecto, los principales objetivos mencionados en el tercer capítulo se han alcanzado. A continuación, se mencionan los objetivos logrados:

- Introducción de mejoras en el Circo de las Poses. Esto se ha logrado mediante la implementación de funcionalidades que faltaban por desarrollar, especialmente en la detección de poses de medio cuerpo, tanto en el nivel difícil como en el fácil.

- Integración la posibilidad de capturar imágenes utilizando un dispositivo móvil en el entorno del "Circo de las Poses". Para conseguir este objetivo, se ha implementado una plataforma web App capaz de enviar las imágenes en tiempo real al circo, donde se recogían las imágenes y luego se trabajaba con ellas.

- Creación del "Circo de las Imágenes" como un nuevo componente del ecosistema. Para alcanzar este objetivo, se han implementado tanto los paneles de ejecución de la parte del circo, como una web App que permite la visualización de los tres tipos de archivos: foto, vídeo y panorámica.

- Integración del código en el ecosistema para documentar todo el proyecto para futuros estudiantes. Esto se ha logrado creando repositorios individuales, donde se ha registrado toda la información necesaria del código, ya sea la instalación, vídeos de demostración, organización del código, entre otros.

El Diagrama de Gantt de la **Fig. 9.1**, se puede observar la distribución final de estos objetivos, mencionados previamente.

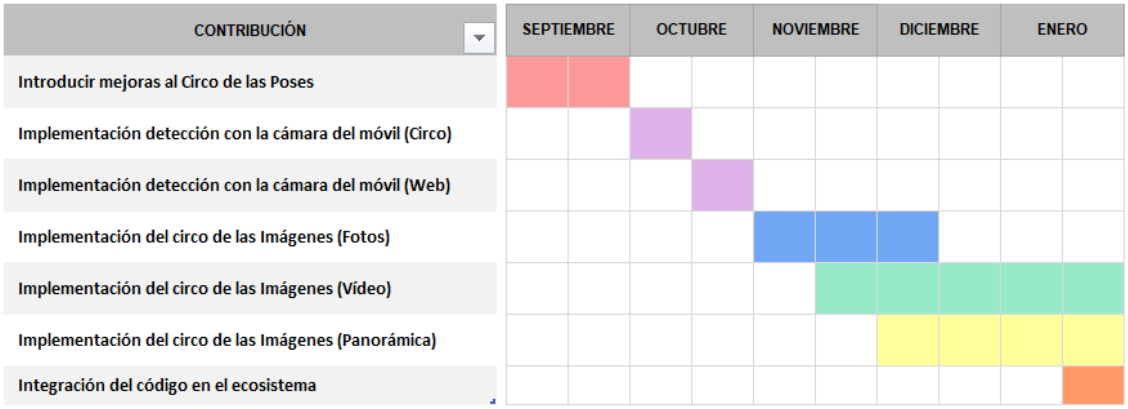

Fig. 9.1. Diagrama de Gantt de la Planificación Final

Se observa que al inicio del proyecto los objetivos se fueron cumpliendo a tiempo. Sin embargo, la implementación del 'Circo de las Imágenes' requirió más tiempo y dedicación de lo inicialmente estaba previsto. La interacción con diferentes tipos de imágenes implicó un esfuerzo adicional para comprender su manejo e implementación.

Tanto las mejoras como la implementación de la opción del circo con la cámara del móvil se ejecutaron de manera dinámica y resolutiva, lo que implicó empezar a tiempo el ‘Circo de las Imágenes’.

Las partes del ‘Circo de las Imágenes’ que necesitaron más tiempo fueron el vídeo y la panorámica. En la parte ocurrió porque hubo problemas con la transmisión de las imágenes en tiempo real al móvil y por el formato del archivo de vídeo recibido. La parte de la panorámica requirió un estudio previo bastante importante y un periodo de pruebas bastante extenso.

9.2. Líneas Abiertas

Durante la investigación y estudio de este proyecto, se evidenció el potencial y alcance de esta iniciativa, dando lugar a una gran variedad de proyectos, conferencias y aplicaciones. La versatilidad para trabajar tanto en el *backend* como en el *frontend*, combinada con las implementaciones realizadas, construye un futuro prometedor para la integración de los drones en la sociedad, cambiando la perspectiva general sobre ellos.

Al concluir este proyecto, se abren nuevas oportunidades a futuros proyectos y desarrollos dentro del ‘Drone Engineering Ecosystem’. A continuación, se nombran algunas posibles mejoras:

1. Mejora de la transmisión de los archivos de vídeo para regular la velocidad de reproducción, así como mejorar la calidad en la transmisión de imágenes mediante el bróker MQTT. Es un asunto que ha quedado pendiente y estaría bien conseguir pulir estos matices.

2. Creación de nuevas funcionalidades para las fotos panorámicas. Por ejemplo, creación de fotos panorámicas con un giro de 360 grados en lugar de en línea recta.

3. Creación de una web App que permita controlar los comandos de la creación de la foto panorámica desde el móvil. Esto permitiría realizar espectáculos y controlar el escenario para el montaje de la foto panorámica desde el móvil, facilitando así su ejecución.

4. Creación de un nuevo circo. Por ejemplo, que trabaje con luces LED.

Son muchas las opciones disponibles para contribuir en el desarrollo del Dron Engineering Ecosystem.

9.3. Conclusiones personales

Durante todo el proceso de este trabajo, mi experiencia ha sido enriquecedora y llena de aprendizajes. La inmersión en nuevas tecnologías me ha permitido explorar y aplicar conocimientos innovadores. Trabajar cerca de los drones ha sido particularmente fascinante.

Este proyecto me ha permitido descubrir tecnologías, aplicaciones, protocolos de comunicación, entre otros, que hasta ahora desconocía. La adquisición de estos conocimientos no solo ha ampliado mi comprensión del campo de los drones y la tecnología asociada, sino que también me ha brindado oportunidad de aplicar estos conceptos en otros contextos en el futuro.

Para finalizar, quiero agradecer a todos los estudiantes que han formado parte de este propósito, y recalcar el valor del apoyo a los drones y a esta iniciativa que busca mejorar la imagen de los drones en la sociedad.

BIBLIOGRAFÍA

[1] DJI Education., ‘Unleash Your Creativity’, https://www.dji.com/es/robomaster-tt

[2] Jose Luis Barrios, ENAIRE, ’Los drones y sus aplicaciones, grandes alas para la aviación’, 6 JUNIO 2016, https://fundacionenaire.es/conocimiento/drones-aplicaciones-aviacion/

[3] DJI, ‘Robomsater TT Tello Talent User Manual’, 03/2021, v.01.

[4] DJI, ‘DJITelloPy API Reference’, 2024, https://djitellopy.readthedocs.io/en/latest/tello/

[5] DJI, ‘Tello SDK 2.0 User Guide’, 11/2018, v1.0

[6] dronsEETAC repository, TelloDroneCircus, https://github.com/dronsEETAC/TelloEngineeringEcosystem.git

[7] IonicFramework, 2024, https://ionicframework.com/

[8] Ionic Docs, 2024, https://ionicframework.com/docs/vue/quickstart

[9] Python, 2024, https://www.python.org/doc/

[10] Medium, 2024, https://medium.com/@pritam.mondal.0711/stream-live-video-from-client-to-server-using-opencv-and-paho-mqtt-674d3327e8b3

[11] MediaPipe, 2024, https://pypi.org/project/mediapipe/

[12] MediaPipe, ‘MediaPipe Legacy Solutions’ https://mediapipe.readthedocs.io/en/latest/solutions/hands.html

[13] OpenCV, 2024, https://docs.opencv.org/

ANEXOS

Anexo A. Funciones 'Practicar' y 'Volar'

Practicar:

```
def practising(self):

    cont = 0
    prevCode = -1

    cv2.namedWindow('videoPractising')

    while self.state == 'practising':

        success, image = self.videostreamer.getFrame()

        if not success:
            print("Ignoring empty camera frame.")
            continue

        code, img = self.detector.detect(image, self.level)

        print('detecto ', code)
        img = cv2.resize(img, (800, 600))

        if code != prevCode:
            cont = 4
            prevCode = code
        else:
            cont = cont - 1
            if cont < 0:
                direction = self.__setDirection(code)
                cv2.putText(img, direction, (50, 450),
cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 255), 10)

        cv2.imshow('videoPractising', img)
        cv2.waitKey(1)
    cv2.destroyWindow('videoPractising')
    cv2.waitKey(1)
```

Volar:

```
def flying(self):

    self.movementGenerator = MovementGenerator(self.anchura,
self.altura, self.profundidad)
    comandos = [self.adelante, self.atras, self.izquierda,
self.derecha, self.arriba, self.abajo, self.flip]

    self.returning = False

    self.direction = ''
    numberOfOperations = 8
    next = 0

    cv2.namedWindow('videoFlying')
```

```
    if self.videostreamer is None:
        self.videostreamer = VideoStreamer(self.int_camera)


    while next < numberOfOperations:
            n =
self.movementGenerator.NewMovement(self.drone.get_battery())
            print("n", n)
            self.comando = comandos[n]
            print("comando: ", self.comando)
            self.drone.send_control_command(self.comando)
            cont = 0
            detected = False
            self.aviso = 0
            code = 0
            start_time = threading.Timer(self.alarma//2, self.avisar)
            start_time.start()
            while not detected and self.aviso != 2:

                success, image = self.videostreamer.getFrame()

                if not success:
                    print("Ignoring empty camera frame.")
                    continue

                code, img = self.detector.detect(image, self.level)
                img = cv2.resize(img, (800, 600))
                self.direction = self.__setDirection(code)
                cv2.putText(img, self.direction, (50, 450),
cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 0, 255), 10)

                cv2.imshow('videoFlying', img)
                cv2.waitKey(1)
                if code == n+1:
                    cont = cont + 1
                    if cont == 8:
                        detected = True
                else:
                    cont = 0

            if self.aviso == 2:
                self.drone.send_control_command(self.corazon_roto)
                next = numberOfOperations + 1

            else:
                start_time.cancel()
                self.drone.send_control_command(self.corazon)
                time.sleep(2)
                if code == 1:  # adelante
                    self.drone.go_xyz_speed(50, 0, 0, 100)
                    self.movementGenerator.forward()
                elif code == 2:  # atras
                    self.drone.go_xyz_speed(-50, 0, 0, 100)
                    self.movementGenerator.back()
                elif code == 3:  # izquierda
                    self.drone.go_xyz_speed(0, -50, 0, 100)
                    self.movementGenerator.left()
                elif code == 4:  # derecha
                    self.drone.go_xyz_speed(0, 50, 0, 100)
                    self.movementGenerator.right()
```

```
                elif code == 5:  # arriba
                    self.drone.go_xyz_speed(0, 0, 50, 100)
                    self.movementGenerator.up()
                elif code == 6:  # abajo
                    self.drone.go_xyz_speed(0, 0, -50, 100)
                    self.movementGenerator.down()
                elif code == 7:  # flip
                    self.drone.send_control_command("flip l")
                # self.connectButton['text'] =
str(self.drone.get_battery())
                next = next + 1

    if (next == numberOfOperations):
        self.drone.send_control_command(self.ole)
    left_right, forward_back = self.movementGenerator.GetPosition()
    if (left_right < 0):
        for n in range(abs(left_right)):
            self.drone.go_xyz_speed(0, 50, 0, 100)
    else:
        for n in range(abs(left_right)):
            self.drone.go_xyz_speed(0, -50, 0, 100)
    if (forward_back < 0):
        for n in range(abs(forward_back)):
            self.drone.go_xyz_speed(50, 0, 0, 100)
    else:
        for n in range(abs(forward_back)):
            self.drone.go_xyz_speed(-50, 0, 0, 100)

    self.drone.land()
    messagebox.showwarning("Success", "Ya estamos en casa",
parent=self.master)
    self.landButton.grid_forget()
    self.closeButton2.grid(row=1, column=1, columnspan=2, padx=5,
pady=5, sticky=N + S + E + W)
    self.movementGenerator.Reset()

    self.connectButton['bg'] = '#CC3636'
    self.connectButton['text'] = 'Connect'
    self.takeOffButton['bg'] = '#CC3636'
    self.takeOffButton['text'] = 'TakeOff'
    self.connected = False
    self.taken_off = False
    self.state = 'initial'
    cv2.destroyWindow('videoFlying')
    cv2.waitKey(1)
```

Anexo B. Clase 'VideoStreamer'

```
import paho.mqtt.client as mqtt

import cv2 as cv2
import numpy as np
import base64

class VideoStreamer:

    def on_messageStreamer(self, cli, userdata, message):
        command = message.topic
```

```
        if command == 'videoFrameAnna':
            image =
base64.b64decode(bytes(message.payload.decode("utf-8"), "utf-8"))
            npimg = np.frombuffer(image, dtype=np.uint8)
            frame = cv2.imdecode(npimg, 1)
            img = cv2.resize(frame, (450, 600))
            self.current_img = cv2.flip(img, 1)

    def __init__(self, camera):
        self.img = None
        self.current_img = None
        self.source = camera
        self.client = None

        self.global_broker_address = "classpip.upc.edu"
        self.global_broker_port = 8000  # 8883

        if self.source == 1:
            # Cámara del móvil
            self.client = mqtt.Client("VideoService",
transport="websockets")
            self.client.username_pw_set("dronsEETAC", "mimara1456.")

            self.client.on_message = self.on_messageStreamer  #
Callback function executed when a message is received
            self.client.connect(self.global_broker_address,
self.global_broker_port)
            self.client.subscribe('videoFrameAnna')
            print('Waiting connection ...')
            self.client.loop_start()

        if self.source == 2:
            # Cámara del ordenador
            self.cap = cv2.VideoCapture(0)

    def getFrame (self):
        if self.source == 1:
            # Cámara del móvil
            if self.current_img is None:
                return False, self.current_img
            return True, self.current_img
        if self.source == 2:
            # Cámara del ordenador
            success, res = self.cap.read()
            img = cv2.resize(res, (800, 600))
            image = cv2.flip(img, 1)
            return success, image

    def disconnect(self):
        if self.source == 1:
            # Cámara del móvil
            self.client.disconnect()
            self.client.loop_stop()
        elif self.source == 2:
            # Cámara del ordenador
            self.cap.release()
```

Anexo C. Resultados de las implementaciones

C.1. Resultados de la implementación del circo con la cámara del móvil.

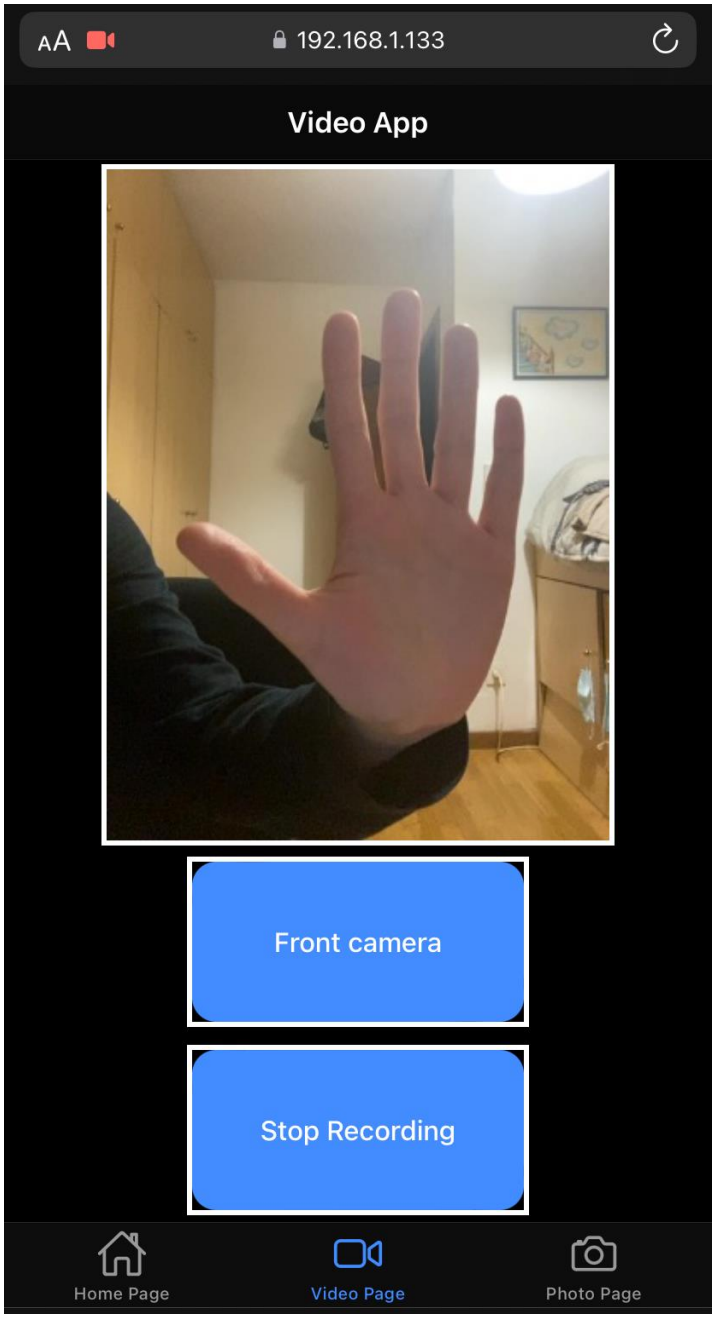

Fig. C.1. Captura de las imágenes desde la cámara del móvil.

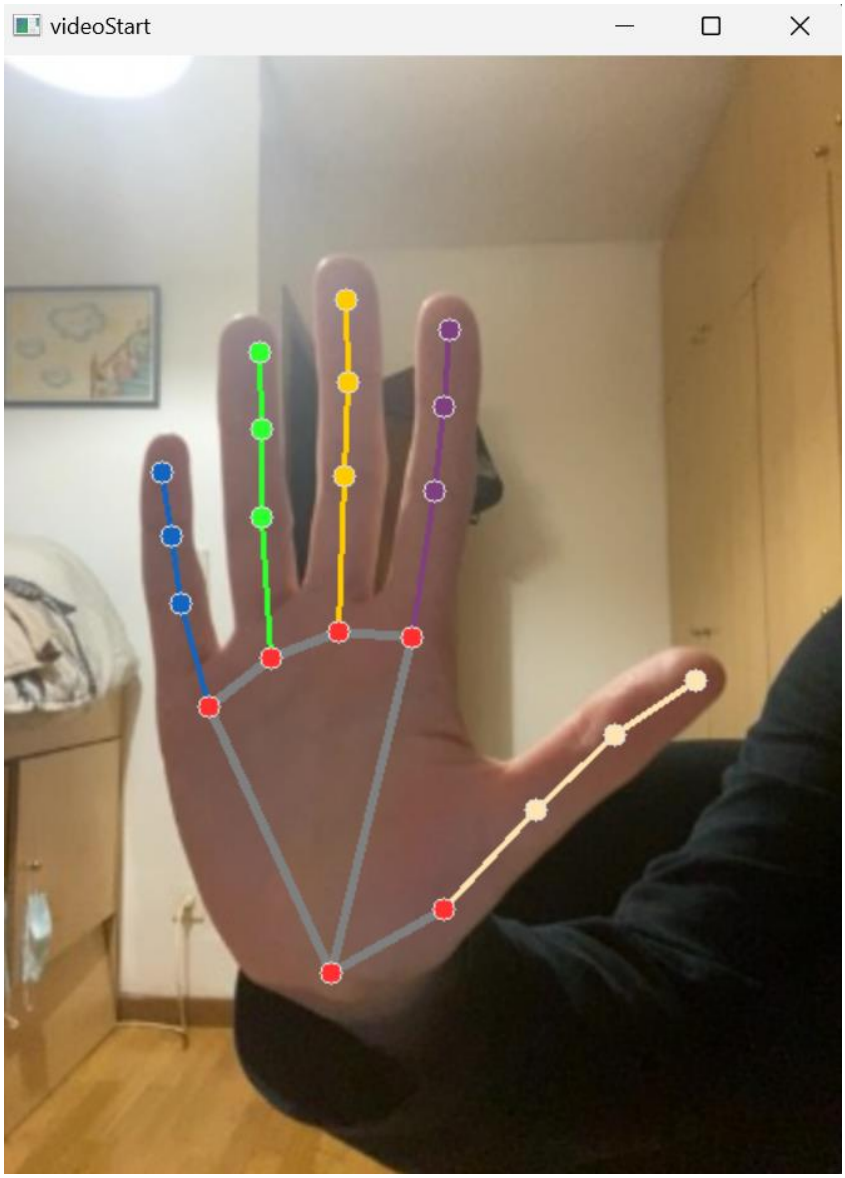

Fig. C.2. Imagen recibida al circo desde el móvil.

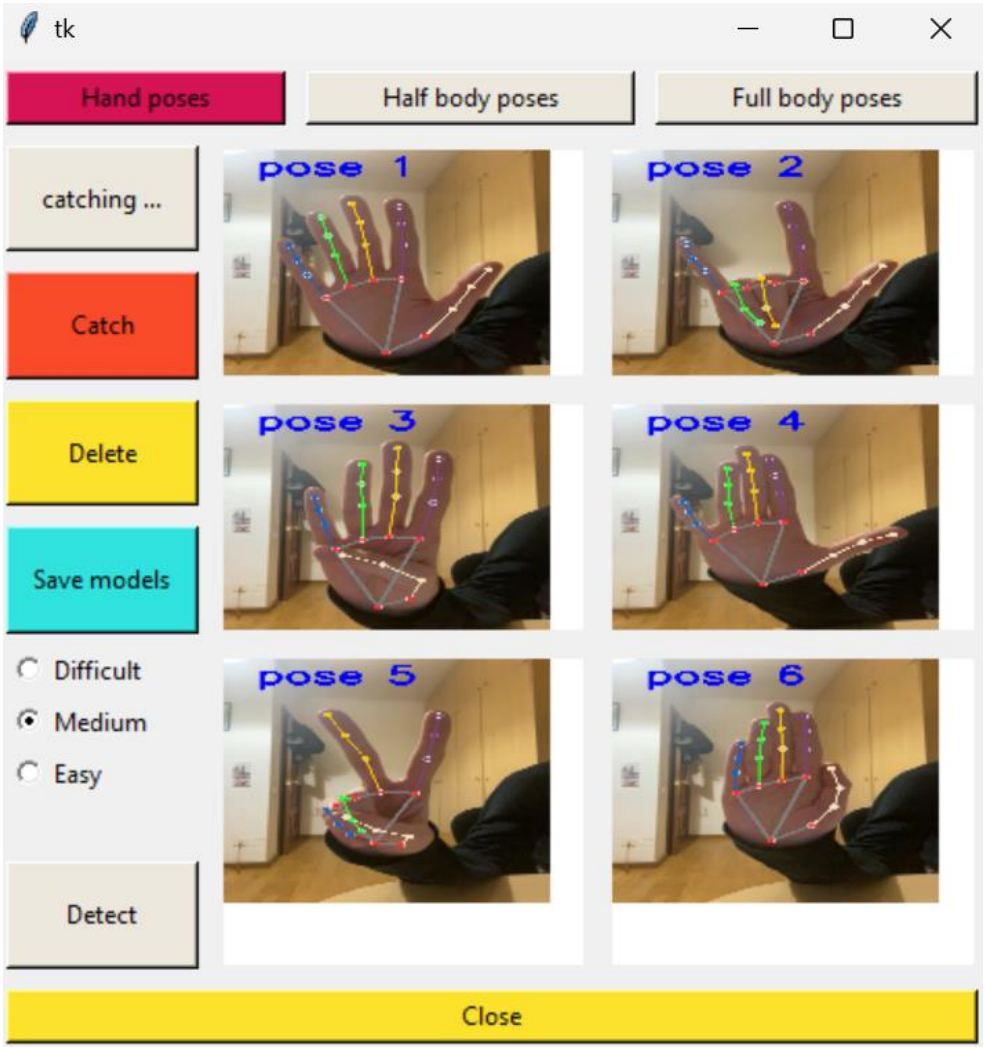

Fig. C.3. Captura de diversas fotos mostradas en los *canvas*

Fig. C.4. Imágenes de los *canvas* en el nivel difícil.

Fig. C.5. Detección de la Pose 1 en el nivel difícil

C.2. Resultados del Circo de las Imágenes

C.2.1. Resultados de las Imágenes

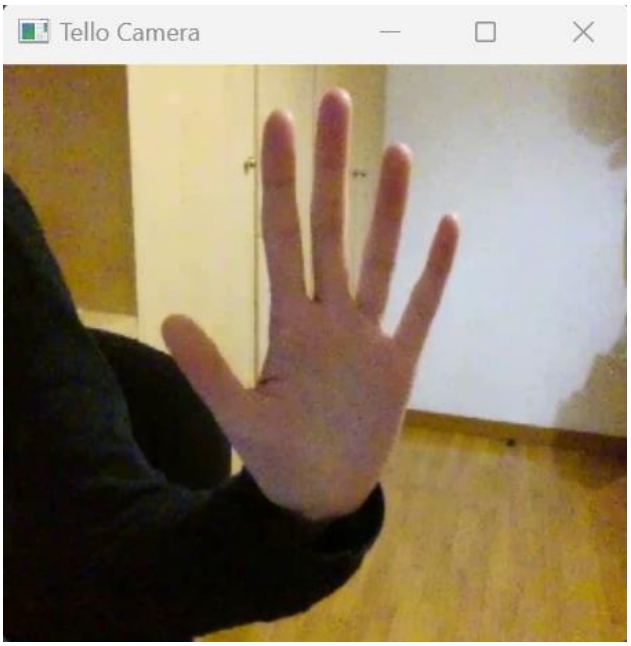

Fig. C.6. Transmisión en *streaming* de la cámara del dron

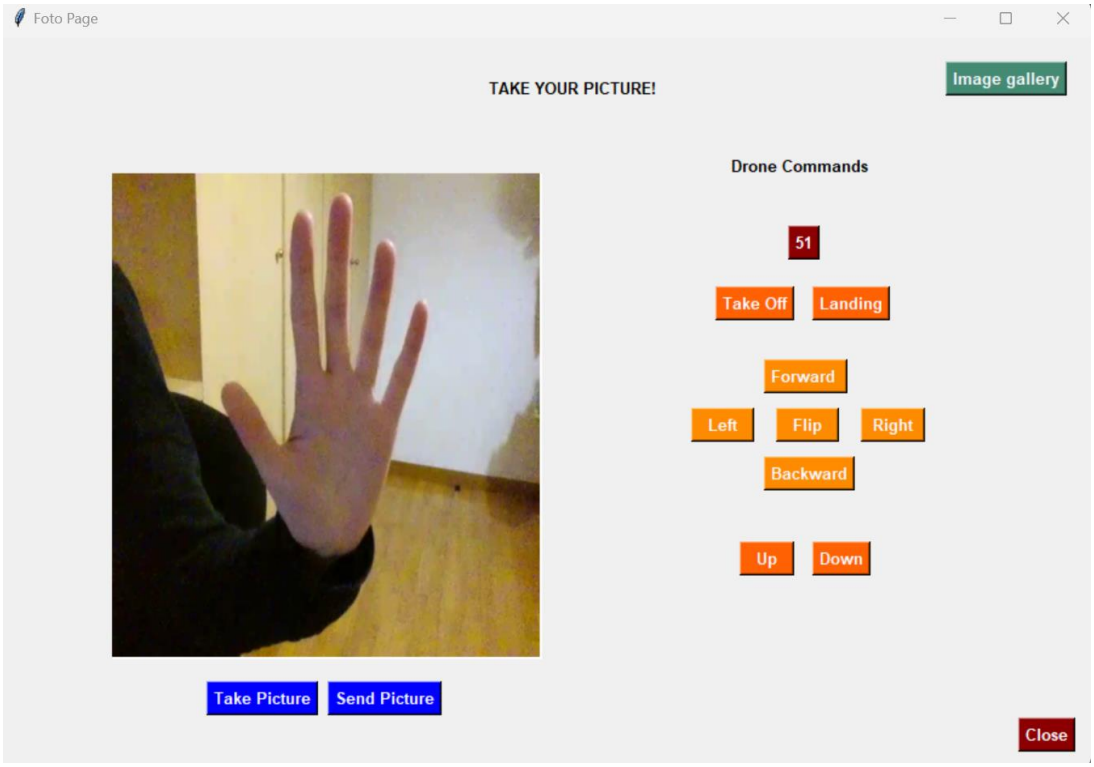

Fig. C.7. Transmisión de la captura de la imagen.

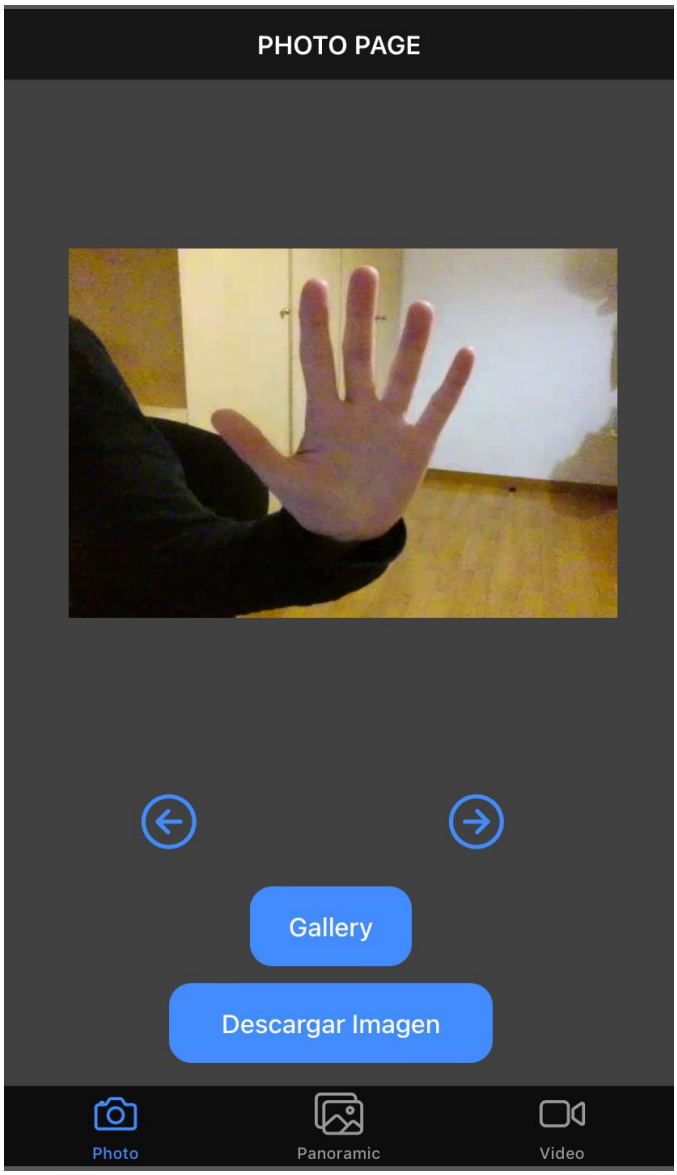

Fig. C.8. Recepción de la captura de pantalla recibida por el circo.

C.2.2. Resultados de las Panorámicas

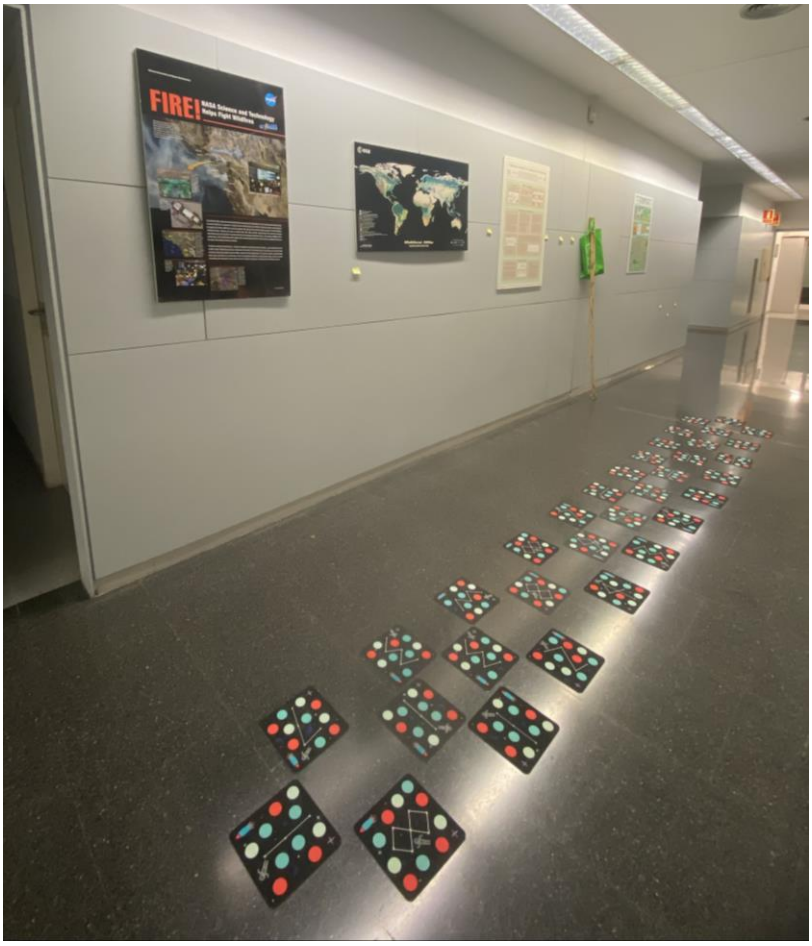

Fig. C.9. Escenario para la creación de las panorámicas.

Fig. C.10. Resultado de una panorámica

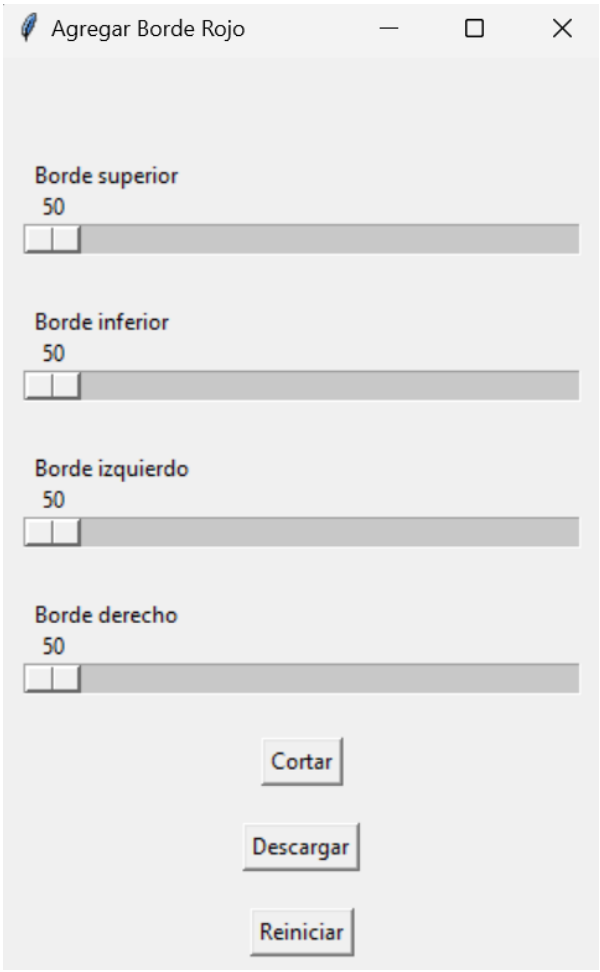

Fig. C.11. Regulador del tamaño del borde rojo de la panorámica.

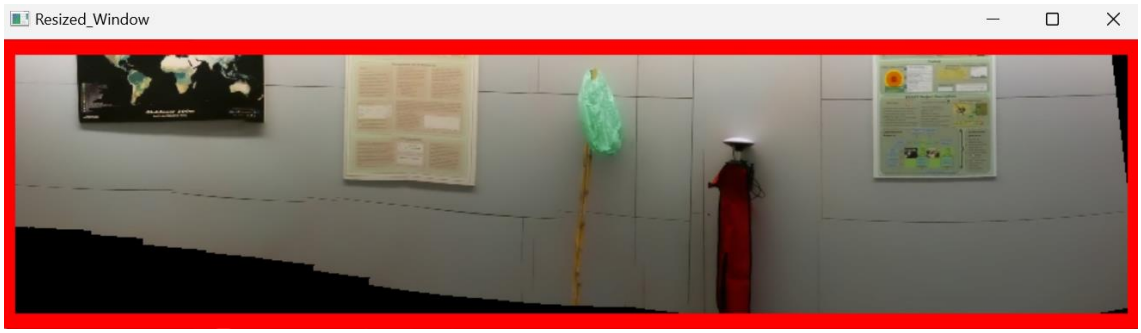

Fig. C.12. Panorámica antes de ser recortada**.**

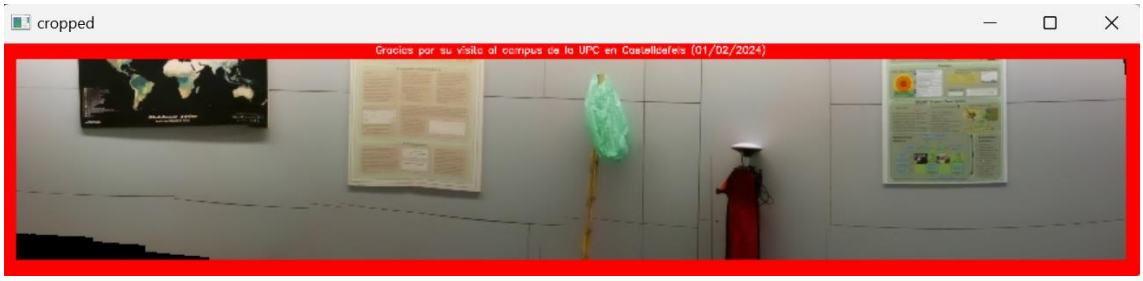

Fig. C.13. Panorámica recortada**.**

C.2.3. Resultados de las Panorámicas:

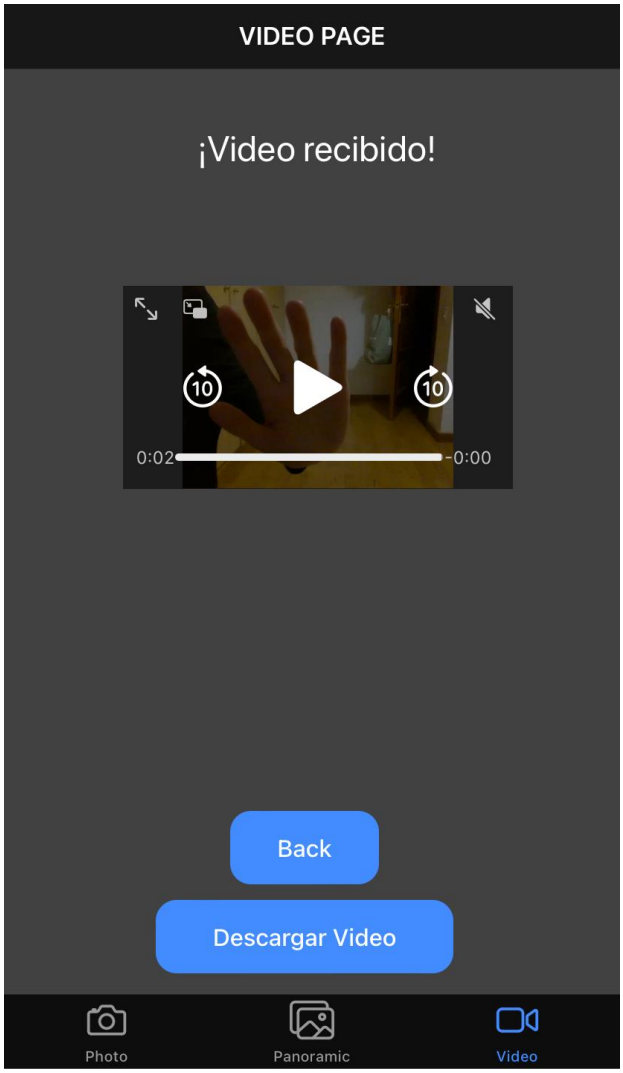

Fig. C.14. Recepción del archivo de vídeo en la web App.