Proyecto de Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Herramienta de gestión para sistemas de inspección basados en múltiples drones

Autor: Alvaro Ramiro Poma Aguilar

Tutor: Jesús Iván Maza Alcañiz y Aníbal Ollero Baturone

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto de Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Herramienta de gestión para sistemas de inspección basados en múltiples drones

Autor:
Alvaro Ramiro Poma Aguilar

Tutor:
Jesús Iván Maza Alcañiz y Aníbal Ollero Baturone

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto de Fin de Máster : Herramienta de gestión para sistemas de inspección basados en múltiples drones

Autor: Alvaro Ramiro Poma Aguilar
Tutor: Jesús Iván Maza Alcañiz y Aníbal Ollero Baturone

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Quiero aprovechar este momento para expresar mi más sincero agradecimiento a todas las personas que han sido fundamentales en la realización de mi Trabajo de Fin de Máster, así como a Dios y a los maestros que han sido una influencia invaluable en mi formación académica.

En primer lugar, me gustaría agradecer a Iván Maza por su orientación y apoyo a lo largo de todo el desarrollo de este proyecto. Su dedicación, conocimientos y motivación han sido cruciales para el éxito de mi TFM.

Asimismo, deseo expresar mi gratitud a Álvaro Caballero por su colaboración y apoyo durante todo el proceso de desarrollo del proyecto. Su contribución ha sido de gran importancia.

No puedo dejar de mencionar a Miguel Gil, quien ha sido mi compañero y ha complementado este trabajo. También quiero agradecer a Víctor Vega por su valiosa ayuda en las pruebas y por compartir generosamente su amplio conocimiento.

Por último, pero no menos importante, quiero dedicar un agradecimiento especial a mi familia, en particular a mi padre y a mi tía, por su incondicional apoyo durante esta etapa de mi vida.

Nuevamente, mi más profundo agradecimiento a todos aquellos que han contribuido de alguna manera a mi Trabajo de Fin de Máster. Vuestra colaboración y apoyo han sido de gran valor y estoy eternamente agradecido por ello.

Alvaro Ramiro Poma Aguilar
Alumno de la Escuela Técnica Superior de Ingenieria de la universidad de Sevilla

Sevilla, 2023

Resumen

El uso de múltiples vehículos aéreos no tripulados (UAVs por las siglas en inglés de Unmanned Aerial Vehicles) en tareas de inspección con vuelo autónomo requieren una interfaz de control simple y amigable que permita gestionar los vehículos no tripulados. El propósito de este Trabajo de Fin de Máster es realizar una estación de mando y control en tierra (GCS por las siglas en inglés de Ground Control Station) que permita controlar y monitorizar el vuelo de múltiples UAVs heterogéneos que realizan misiones de inspección autónomas sobre grandes infraestructuras. La flota utilizada consiste en una plataforma de ala fija con aterrizaje y despegue vertical (VTOL por las siglas en inglés de Vertical Take-Off and Landing) modelo DeltaQuad Pro y dos quadricópteros modelo Matrice M210 RTK V2 de la marca DJI. Cada UAV está equipado con un ordenador a bordo que amplía sus funcionalidades y, mediante el uso de ROS (Robot Operating System), y herramientas y paquetes para la conexión con el autopiloto, permite obtener telemetría del UAV, imágenes de las cámaras a bordo y controlar el vuelo a través de la GCS.

El desarrollo de la GCS se presenta como una solución escalable y compatible con múltiples autopilotos. Se basa en una arquitectura cliente-servidor y consta de dos programas que se comunican a través de una API y un websocket, lo que permite la interacción con varios usuarios y la integración con aplicaciones de terceros. El cliente es una interfaz de usuario basada en REACT.js, mientras que el servidor se encarga de la lógica y el control del sistema, estableciendo la conexión entre los UAVs y la interfaz.

Para validar la herramienta desarrollada ha sido necesaria la simulación de las distintas aeronaves, realizando simulaciones en hardware in the loop con el Matrice 210 y software in the loop con el autopiloto PX4 para simulaciones del VTOL modelo DeltaQuad. También se hizo la validación de todo el sistema mediante la inspección de lineas eléctricas de Endesa ubicadas en el campo de vuelo experimental ATLAS situado en Villacarrillo (Jaén). Estas pruebas se realizaron en el contexto del proyecto H2020 AERIAL-CORE [8]. Este trabajo también entra dentro del marco de trabajo de proyectos como OMICRON [9] y RESISTO [7].

Palabras clave: GCS, GUI (Graphical User Interface), Multi-UAV, UAV, drones, ROS, inspección automática, Estación de Mando y Control en Tierra, DeltaQuad, DJI M210.

Abstract

The use of multiple unmanned aerial vehicles (UAVs) in autonomous inspection tasks requires a simple and user-friendly control interface that allows managing the unmanned vehicles. The purpose of this Master's Thesis is to develop a Ground Control Station (GCS) that enables the control and monitoring of the flight of multiple heterogeneous UAVs executing autonomous inspection missions in large infrastructures. The UAV fleet used consists of a fixed-wing platform with vertical take-off and landing (VTOL) capabilities called DeltaQuad Pro, and two DJI Matrice M210 RTK V2 quadcopters. Each UAV is equipped with an onboard computer that extends its functionalities, and by using ROS (Robot Operating System), and tools, and packages for connecting to the autopilot, it allows monitoring telemetry, viewing images from onboard cameras, and controlling the flight through the GCS.

The development of the GCS is presented as a scalable solution compatible with multiple autopilots. It is based on a client-server architecture and consists of two programs that communicate through an API and a websocket, enabling interaction with multiple users and integration with third-party applications. The client is a user interface based on REACT.js, while the server handles the logic and control of the system, establishing the connection between the UAVs and the interface.

To validate the developed tool, it was necessary to simulate the different aircraft, conducting hardware-in-the-loop simulations with the Matrice 210 and software-in-the-loop simulations with the PX4 autopilot for the DeltaQuad VTOL model. Likewise, the validation of the entire system was performed by inspecting Endesa power lines located at ATLAS, an experimental test flight area Villacarrillo (Jaén), as part of the H2020 AERIAL-CORE project [8]. This work also falls within the framework of projects such as OMICRON [9] and RESISTO [7].

Key words: GCS, interfaz Multi-UAV, UAV, drones, ROS, Inspection, Ground control Station , Delta Quad, DJI Matrice M210.

Índice Abreviado

Índice

1 Introducción

1.1 Motivación

Los sistemas multi-UAV se refieren a la utilización de múltiples drones o UAVs en conjunto para llevar a cabo misiones de manera cooperativa. Estos sistemas aprovechan las capacidades de varios UAVs que trabajan en coordinación, permitiendo realizar de manera eficiente y efectiva una amplia gama de aplicaciones.

Dentro de los sistemas multi-UAV, el sistema de control, también conocido como estación de mando y control, desempeña un papel fundamental. Este sistema es responsable de gestionar y monitorizar los UAVs. Actualmente existen diversos sistemas de control de código abierto, como QGroundControl para UAVs con autopiloto PX4 y MissionPlanner para UAVs con autopiloto ArduPilot. Asimismo, los drones comerciales de marcas como DJI también cuentan con sus propias estaciones de control en tierra. Sin embargo, la mayoría de estos sistemas están diseñados para un solo UAV o para múltiples UAVs del mismo fabricante o con el mismo autopiloto.

Debido a las deficiencias de estas estaciones de control es de vital importancia el desarrollo de una estación de mando y control que tenga compatibilidad con diferentes autopilotos, donde se pueda representar información relevante para el monitorización, como datos de sensores externos y cámaras de cada UAV.

1.2 Objetivos

El contexto de este Trabajo de fin de Mester es desarrollar un software de gestión y monitorización de múltiples UAVs (Vehículos Aéreos no Tripulados) para aplicaciones de inspección de grandes infraestructuras. Esta herramienta permitirá trabajar con un equipo de UAVs que poseen diferentes características, incluyendo dos UAVs de tipo cuadricóptero modelo Matrice 210 V2 RTK de DJI y un UAV de tipo VTOL modelo DeltaQuad Pro.

La estación de control proporcionará telemetría en tiempo real de los UAVs, así como acceso a los datos de los sensores y cámaras embarcadas. Además, contará con una interfaz intuitiva que permitirá ejecutar y comandar misiones pre-planificadas para que los UAVs las lleven a cabo.

También se implementará una capa de abstracción en la estación de control para facilitar la comunicación con aplicaciones externas a los UAVs. Esta capa permitirá la integración del sistema con aplicaciones de terceros, ampliando así las posibilidades de uso y aprovechando sinergias con otros programas.

Adicionalmente, se implementará el sistema multi-UAV, donde cada UAV llevará a bordo un computador capaz de comunicarse eficientemente con la estación de control. Esto permitirá recibir comandos y mantener una comunicación bidireccional entre los UAV y la estación de control.

1.3 Organización de contenido

En el Capitulo 2 se realiza una descripción del sistema multi-UAV, para posteriormente en el Capitulo 3 explicar el desarrollo de la estación de mando y control en tierra. En el Capitulo 4 se explicará cómo se ha llevado a cabo la integración con el dron DJI Matrice 210. En el Capitulo 5 se describen las pruebas realizadas para la validación de la herramienta desarrollada. Finalmente el Capítulo 6 recoge las conclusiones y líneas de desarrollo futuras.

2 Descripción del Sistema Multi-UAV

En este capitulo se realizara una pequeña introducción sobre los sistemas multi-UAV y se describe cada una de las partes del sistema implementado a excepción del sistema de control en tierra que se profundizara en el Capitulo 3.

2.1 Introducción y trabajos previos

En los últimos años, hemos sido testigos de un aumento significativo en el uso de drones en diversas operaciones, desde la fotografía y el cine hasta la vigilancia y la entrega de paquetes. El desarrollo de nuevas tecnologías y la disminución de los costos de los drones han impulsado su adopción en diferentes sectores, lo que ha generado un aumento en la variedad de operaciones que se pueden llevar a cabo con estos dispositivos.

Una de las aplicaciones más interesantes y en constante crecimiento es la inspección de grandes infraestructuras. Hasta día de hoy, equipos de profesionales trepaban con ayuda de cuerdas, andamios o plataformas elevadoras para inspeccionar estas infraestructuras, lo que obliga en ciertas ocasiones a la interrupción de servicios con costes muy elevados. Esta aproximación manual no solo consume mucho tiempo sino que también es extremadamente costosa. Los robots aéreos ofrecen la posibilidad de inspeccionar localizaciones de difícil acceso sin necesidad de poner en riesgo a personal humano. Es decir, los robots aéreos pueden recoger imágenes y datos detallados para evaluar y prever mejor las necesidades de mantenimiento de los activos, eliminando el peligro que conlleva el trabajo humano, reduciendo el riesgo de averías y mejorando la competitividad al minimizar el tiempo de inactividad de la producción. Es por esto, que se consideran una herramienta valiosa para la inspección de infraestructuras críticas con gran impacto en la seguridad, medio ambiente y a nivel financiero.

Las tareas de inspección y mantenimiento tanto de infraestructuras públicas como industriales, incluye diversas actividades como inspecciones a nivel general, cercanas o muy cercanas, de contacto y manipulación de la estructura para labores de limpieza, instalación de dispositivos, etc. Así, por ejemplo, en el proyecto H2020 AEROARMS [20], robots aéreos llevaron acabo trabajos de inspección de contacto con tuberías y tanques en refinerías para determinar el grosor de las paredes en altura como aplicación de manipulación robótica aérea [22] desarrollado en este proyecto y en el proyecto FP7 ARCAS. Además, este tipo de tecnologías ha sido empleada también en infraestructuras públicas, en particular para inspección de puentes en el proyecto H2020 AEROBI. Aunque este TFM estará enfocado especialmente a la inspección sin contacto a través de drones comerciales ligeramente modificados.

Dentro de la aplicación de este tipo de tecnologías a inspecciones, destacan las inspecciones de líneas eléctricas, una de las áreas más importantes y desafiantes. Este tipo de infraestructuras, supone un gran riesgo para el ser humano por su altitud y el riesgo eléctrico que conllevan. Este

tipo de inspección involucra muchos miles de kilómetros en Europa, lo que requiere un seguimiento preciso de las líneas y la detección y localización de situaciones peligrosas, como pueden ser fallos detectables a través de tecnología termográfica, fallos en aisladores, incluyendo el levantamiento de modelos 3D a partir de los datos recogidos y el cálculo de distancias a edificios y vegetación.

Las líneas de servicios públicos, como las líneas de transmisión eléctrica y tuberías, son inspeccionadas en la actualidad una vez cada ciertos periodos de tiempo, que pueden oscilar entre 3 y 18 meses. En las pasadas décadas, se empleaban helicópteros tripulados para realizar estas labores de inspección, lo cual resulta muy costoso y arriesgado, o bien a través de equipos terrestres con vehículos limitados al terreno donde se encuentre la línea eléctrica. Una vez más, el uso de vehículos aéreos no tripulados ofrece ventajas notables respecto a este tipo de técnicas. Algunas empresas han dedicado sistemas multirrotores para este tipo de tareas dentro de la línea de visión visual (Visual Line of Sight, VLOS en inglés), como por ejemplo muestra el vídeo *https://www.youtube.com/watch?v=Bj5ByZKVNao*. Este ejemplo emplea diferentes tipos de cámaras (visuales, con tecnología infrarroja y ultravioleta), como será el caso de aplicación de este TFM ya que emplearemos multirrotores de características muy similares a los del ejemplo. Una de las diferencias que se implementan en este trabajo con respecto al ejemplo, es ampliar el rango de operación fuera de la línea de visión visual (Beyonf Visual Line Of Sight, BVLOS en inglés) para inspecciones de largo alcance. Este tipo de operaciones han estado limitadas por la autonomía de vuelo, la seguridad y la normativa actual.

Como se está demostrando en el proyecto AERIAL-CORE de H2020 [8], los robots aéreos podrían reducir drásticamente los costes de inspección de líneas eléctricas al tiempo que aumentan la frecuencia y la calidad de la supervisión.

Hoy en día, los multi-rotores, con un alcance y autonomía reducidos, y generalmente con pilotos remotos en línea de visión visual, se aplican para inspecciones locales de torres de transmisión [11], [2] o segmentos de la red [12]. Sin embargo, esto no es suficiente para la inspección de larga duración de la red eléctrica compuesta por líneas de media y alta tensión. Motivadas por esto, están surgiendo nuevas líneas de investigación para mitigar las limitaciones de estos robots [13]. Además de a nivel de investigación, la industria está lanzando productos comerciales al mercado que buscan mejorar las características de equipos anteriores con estas limitaciones. Dos ejemplos de empresa que han dado este paso tecnológico son Virtual Technologies y una de las más conocidas por su implicación con tanto en el ámbito industrial como recreativo, DJI con los dispositivos que se emplearán en este trabajo de fin de Máster.

La inspección de varios kilómetros de líneas eléctricas o en general de infraestructuras extensas, puede llevarse a cabo de manera más eficiente con múltiples UAVs [3]. Numerosas ventajas se consiguen con estos sistemas multi-UAV [17]: reducción del tiempo total de la inspección; minimización de retrasos en la detección de anomalías; aplicación de técnicas de trabajo cooperativo como es en nuestro caso de aplicación; compaginando una vista cerrada en la inspección local a través de los multirrotores y una vista global y menos detallada por parte de la aeronave VTOL de ala fija; y mejoras en la fiabilidad al no depender de una sola plataforma. Un ejemplo de arquitectura para sistemas multi-UAV se puede encontrar en [16]. En ese artículo se presenta una arquitectura distribuida para múltiples vehículos aéreos no tripulados que permitirá su coordinación y cooperación de una forma autónoma. Algunos de los problemas que se resuelven en ese trabajo relacionados con la ejecución de misiones para este tipo de sistemas son: la descomposición de tareas complejas, asignación de tareas, detección y resolución de conflictos, etc. También en esta tesis se presenta una interfaz persona-máquina para poder manejar la plataforma. Con esto podemos concluir que es un documento muy a tener en cuenta en nuestro proyecto ya que podríamos adaptar este tipo de arquitectura a nuestro caso de estudio.

A nivel de implementación de este tipo de arquitecturas cabe destacar el framework ROS MAGNA [19] donde se muestra un entorno de trabajo basado en ROS para la definición y el manejo de misiones cooperativas para sistemas multi-UAV. Este entorno de trabajo permite la integración de

varios autopilotos y crea máquinas de estados para el control las misiones individuales de cada UAV, además incluye multitud de experimentación simulada y real. Este enfoque se acerca más a lo que se pretende conseguir en este trabajo de Fin de Máster debido a que la herramienta ROS está más que extendida en el uso de sistemas robóticos tanto en la industria como en la investigación, y por supuesto por la labor que consiguen a la hora de hacer transparente el autopiloto del dispositivo. Esta última labor nos es vital para integrar plataformas distintas en un mismo sistema heterogéneo, en nuestro caso con dos robots comerciales con dos autopilotos distintos y distinta tipología de UAV.

2.2 Arquitectura del sistema

El sistema multi-UAV utilizado posee una arquitectura centralizada de múltiples vehículos aéreos heterogéneos que buscan ejecutar en el menor tiempo una tarea, donde se posee una estación se control que gestiona el vuelo de todos los vehículo aéreo como se visualiza en la Figura 2.1.

El sistema esta conformado por una flota de vehículos aéreos no tripulados heterogéneos compuestos por aeronaves Matrice M210 y DeltaQuad. Estas aeronaves son complementarias en las tareas de inspección de forma que el DeltaQuad realizará las tareas que involucren recorrer largas distancias y el Matrice M210 realizará tareas que requieren una mejor precisión en distancias más cortas. Cada UAV poseerá un ordenador a bordo que permitirá su gestión mediante la conexión con el autopiloto, y la comunicación con la estación de control en tierra (GCS) a través de la capa de comunicación, la cual trabaja con WiFi de largo alcance en la capa física.

Los UAVs cooperarán de manera intencional, donde cada UAV individualmente tendrá definida una tarea que pertenecerá a la misión global y el sistema de control se encarga de enviar las tareas o misiones a cada uno de los UAVs. Las tareas serán planificadas por un software externo y desde la estación de control solo se podrán supervisar y enviar a los UAVs.

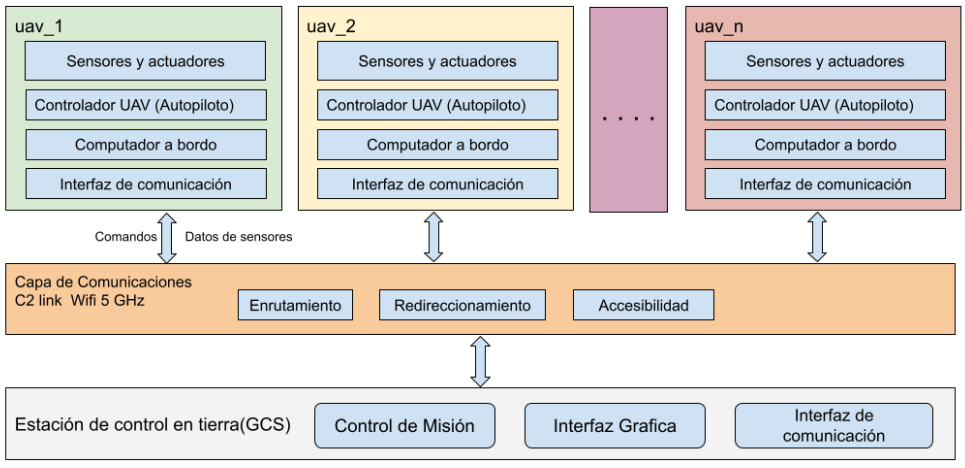

Figura 2.1 Arquitectura del sistema multi-UAV.

2.3 Comunicaciones

Cada UAV se comunica directamente con la GCS a través de una comunicación V2G con topología estrella, utilizando el estándar WiFi 802.11 a/b/g/n/ac a una frecuencia operativa de 5 GHz. Para ello, se utilizaron dispositivos Ubiquiti Bullet AC en cada UAV como interfaz de comunicación. En tierra

se encuentra una Ubiquiti Rocket Prinm 5AC Gen 2 con antenas omnidireccionales configurada como estación, mientras que la Ubiquiti que se encuentra abordo de los UAVs se configura como punto de enlace, lo que permite que la GCS se comunique con los UAVs. Cada Ubiquiti se alimenta por el puerto Ethernet mediante PoE (Power over Ethernet). En el caso de los UAVs la conexión de red de la Ubiquiti se conecta directamente al ordenador a bordo y la Ubiquiti en tierra se conecta a un switch. De este modo se pueden comunicar los UAV y la estación de control como se ve en la Figura 2.2.

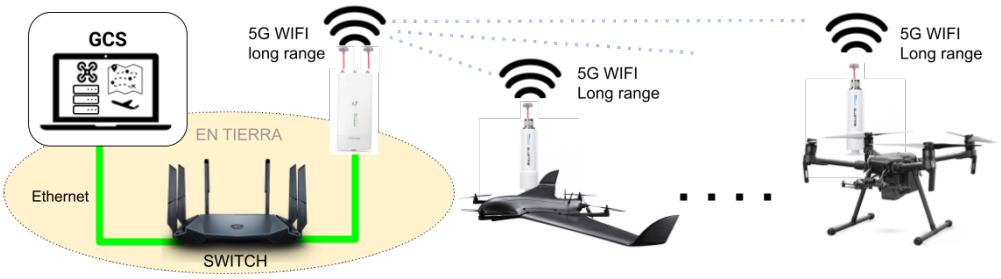

Figura 2.2 Esquema del sistema de comunicación de enlace C2.

2.4 Aeronaves

La flota de aeronaves utilizada en este proyecto consiste en dos UAVs de tipo quadrotor Matrice M210 de la empresa DJI y un VTOL DeltaQuad PRO de la de la empresa Vertical Technologies.

2.4.1 DJI Matrice 210 V2 RTK

El Matrice 210 RTK V2 (M210 RTK V2) es una potente plataforma de drones industriales con una agilidad y alta velocidad, con barómetros y IMUs redundantes con una alta fiabilidad, y funciones de vuelo inteligentes que facilitan la captura de tomas complejas. Los sensores visuales de la aeronave permiten una precisión mejorada en el vuelo estacionario, incluso cuando se vuela en interiores o en entornos donde el GNSS no está disponible [6].

El M210 RTK V2 tiene incorporado el DJI D-RTK™ M 2, que proporciona datos de orientación más precisos para el posicionamiento. Un avanzado sistema de gestión de energía junto con baterías duales garantiza el suministro de energía y mejora la seguridad de vuelo. Sin carga útil, el M210 RTK V2 tiene un tiempo de vuelo de hasta 33 minutos y está equipado con numerosos puertos de expansión para ampliar sus aplicaciones. En la Figura 2.3 se muestra un dron M210 RTK V2 con una cámara Zenmuse XT2 y una Zenmuse X5S.

La serie Matrice 210 V2 solo es compatible con los montajes X4S, X5S, X7, Z30, XTS, XT2, H20, H20T en el gimbal I, y no es necesario utilizar un controlador para obtener la transmisión de vídeo.

Además DJI proporciona herramientas y plataformas de desarrollo para que los desarrolladores de software puedan crear aplicaciones personalizadas y ampliar las capacidades de los productos de DJI, entre las que tenemos DJI Cloud API, Mobile SDK, Payload SDK, UX SDK, Windows SDK, Onboard SDK y Onboard SDK ROS, de las cuales nos centraremos en las dos últimas.

Onboard SDK permite comunicar directamente a una computadora con el controlador de vuelo de DJI y con sensores que posea la aeronave a través de una interfaz serie y una comunicación Ethernet sobre USB. El Onboard SDK ROS trabaja en conjunto con el Onboard SDK proporcionando acceso a la telemetría de la aeronave, el control de vuelo y otras funciones de la aeronave a través de topics

Figura 2.3 DJI Matrice M210 .

de ROS. Estas herramientas permiten al desarrollador integrar su propia lógica al control del UAV mediante lenguajes de programación como C++ y Python.

2.4.2 DeltaQuad PRO

El DeltaQuad PRO (ver Figura 2.4) de la empresa Vertical Technologies es una aeronave tipo V-TOL (Vertical Take-Off and Landing) totalmente eléctrica que está diseñada para ofrecer una gran estabilidad tanto en modo cuadricóptero como en modo de ala fija, con una transición fluida entre ambos y tiempos de vuelo de hasta 110 minutos y alcances operativos de hasta 100 Km, Su diseño permite llevar a cabo misiones o tareas de manera totalmente autónoma, como inspección, cartografía, vigilancia y entrega de carga [25].

Algunas de las especificaciones generales del DeltaQuad PRO son:

- Carga máxima de pago: 1.2 kg
- Envergadura: 235 cm
- Velocidad de crucero: 65 km/h
- Velocidad máxima: 100 km/h
- Velocidad de pérdida: 43 km/h
- Velocidad máxima del viento: 12 m/s
- Altitud máxima: 3000 m

El DeltaQuad PRO puede albergar una amplia variedad de cargas útiles: cámaras tales como Sony A7R IV, FLIR Duo Pro R, MicaSense Altum, y sensores medioambientales y otros personalizados. De todos estas nos centraremos en la cámara FLIR Duo Pro R que es la que posee el DeltaQuad PRO. Esta cámara combina una cámara térmica/infrarroja y una cámara RGB 4K en un solo dispositivo y puede proporcionar transmisión de vídeo en vivo para fines de vigilancia y reconocimiento.

Además el DeltaQuad PRO posee un controlador de vuelo basado en Pixhawk 4, que ejecuta el software autopiloto de código abierto PX4 lo que hace la plataforma de vuelo altamente personalizable. Pixhawk 4 ofrece funciones avanzadas como la evitación de obstáculos, visión por ordenador, la planificación de trayectorias, etc. Además posee la capacidad de comunicarse por comunicación serie a un computador a bordo que ejecute ROS mediante PX4-Ros 2 bridge o MAVROS. El software PX4 admite varios sensores, cargas útiles y protocolos de comunicación.

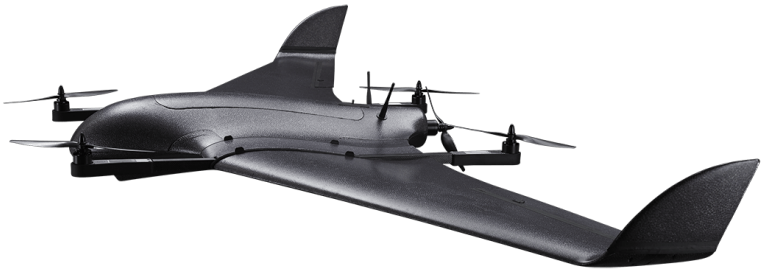

Figura 2.4 UAV DeltaQuad Pro .

2.5 Computador a bordo

Para el desarrollo de este trabajo se utilizó como ordenador a bordo la placa de desarrollo Raspberry Pi 4 con 4GB de RAM. Este ordenador aunque no es muy potente, permite comunicarnos y acceder a funcionalidades del autopiloto de cada UAV, procesar la información y comunicarnos con la estación de control. El ordenador abordo se comunica con la Ubiquiti para establecer conexión con la GCS y con el autopiloto del UAV como se muestra en la Figura2.5.

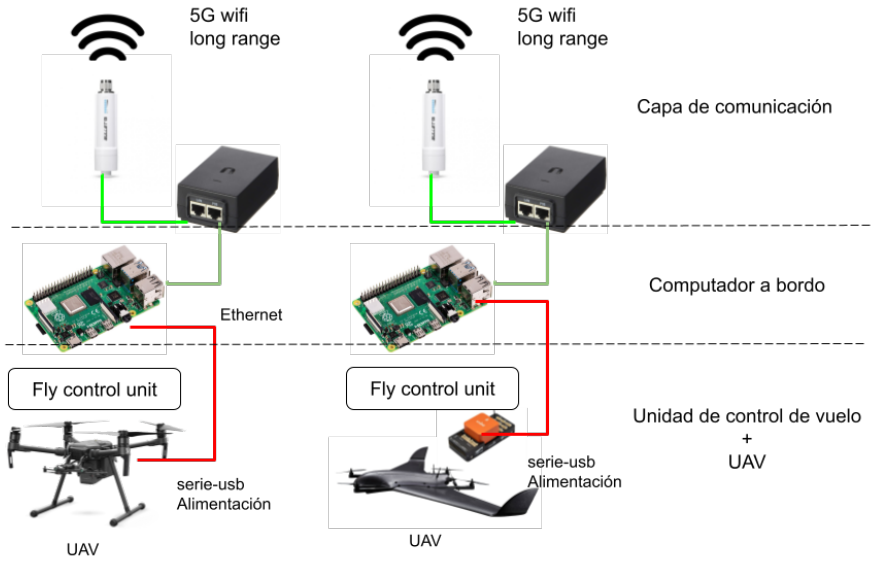

Figura 2.5 Conexión del computador a bordo en el DJI Matrice M210 y en el DeltaQuad Pro .

El computador a bordo provee de versatilidad al UAV dando la capacidad de implementar nuevas funcionalidades dependiendo de la tarea a realizar. Las funcionalidades varían dependiendo de la aplicación: detección de obstáculos, estimación de posición basada en visión, planificación dinámica, integración de sensores personalizados, etc.
Para que los UAVs realicen la tareas de inspección y el recorrido de rutas planificadas en cada ordenador a bordo se ejecuta un nodo de ROS capaz de recibir información de la GCS y ejecutar acciones sobre el UAV. Además hay otros nodos que publican la información de autopiloto, a la cual la GCS se suscribe para mostrarla en la interfaz de control.

Para establecer la conexión con su correspondiente UAV, cada ordenador necesita características especificas y tienen diferentes programas para conectar con el autopiloto correspondiente, como se muestra en la Tabla 2.1.

Tabla 2.1 Características del software del ordenador a bordo.

UAV	SO	Ros Versión	Comunicación con autopiloto
Matrice M210	Ubuntu 18	Melodic	DJI OSDK
DeltaQuad	Ubuntu 20	Noetic	Mavros

2.6 Estación de mando y control en tierra

La estación de control en tierra o GCS es generalmente un programa que se ejecuta en un computador y se comunica con uno o varios UAVs. Tiene la función de mostrar información de cada UAV en tiempo real: la posición, la velocidad, la orientación, imágenes de la cámara a bordo o información de otros sensores en forma de gráficos, texto o visualizada con instrumentos como los de un avión. Además la GCS también se puede utilizar para controlar el vuelo del UAV, cargar comandos de misión y establecer parámetros.

Estas características de la GCS no son limitantes y permiten aumentar las funcionalidades personalizando en función del la aplicación requerida, El sistema desarrollado en este trabajo cuenta con las siguientes funcionalidades:

- Visualización de telemetría y estado en tiempo real de los UAVs.
- Commandar UAVs.
- Interfaz amigable.
- Múltiple ventanas.
- Capa de abstracción para integración con programas de terceros

En la Figura 2.6 se muestra una captura de la GCS en las pruebas realizadas en la Plaza del Agua situada junto a la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla.

El desarrollo de la GCS es la parte principal de este trabajo por lo que se profundizara en el desarrollo de la misma en el Capitulo 3.

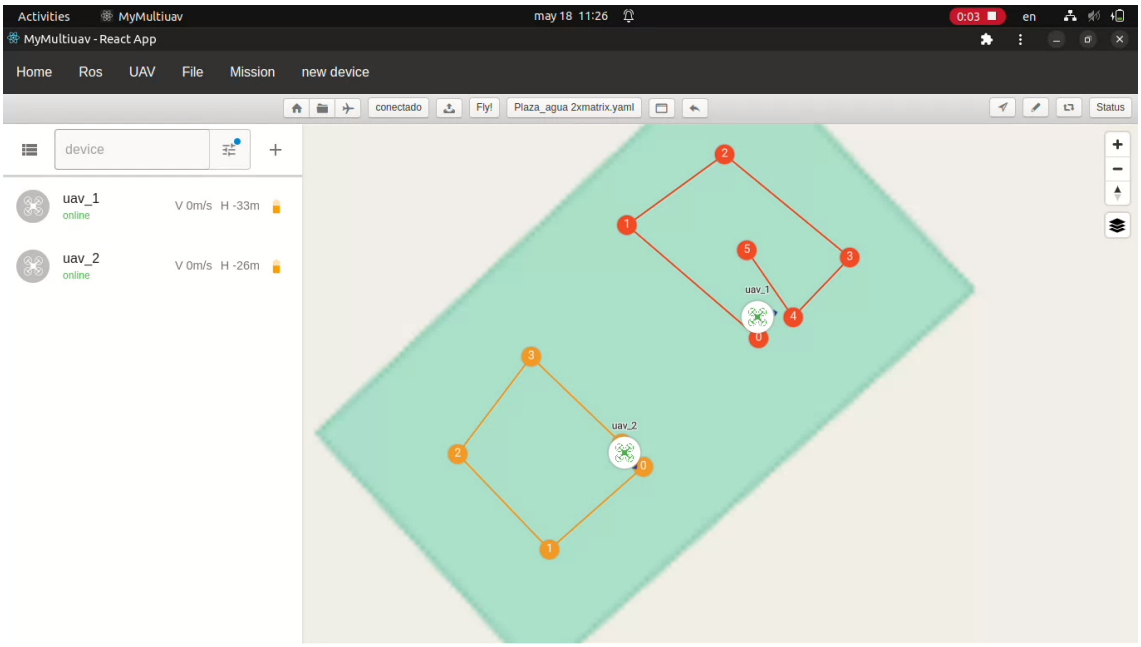

Figura 2.6 Interfaz Principal de la GCS durante una prueba en la Plaza del Agua situada junto a la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla .

3 Estación de Mando y Control en Tierra (GCS)

La estación de mando y control en tierra (GCS) es el software que proporciona al operador la facilidad de controlar y monitorizar los vehículos aéreos no tripulados. En este capitulo se describirá la estructura, la comunicación y el funcionamiento de la GCS desarrollada en este trabajo.

3.1 Requerimientos del sistema

El desarrollo de la Estación de Mando y Control en Tierra (GCS, por sus siglas en inglés) se encuentra regido por los requisitos de la aplicación en la cual se utilizará. La GCS ha sido desarrollada como parte de diversos proyectos en los cuales el Grupo de Robótica, Visión y Control (GRVC) participa activamente, como los proyectos RESISTO, AERIAL-CORE y ÓMICRON, de los que se ha obtenido el caso de estudio que se representa en la Figura 3.1.

Los requisitos del sistema incluyen la necesidad de contar con una interfaz que permita la visualización en tiempo real de la telemetría, del estado de cada UAV, así como la capacidad para cargar comandos y tareas, tener soporte de streaming de vídeo, y permitir la integración de diferentes tipos de vehículos autónomos que operen con ROS.

Con respecto al desarrollo de misiones automáticas, tanto individuales como conjuntas, se procederá a cargar una misión utilizando un archivo de tipo YAML, en el cual se especificará el UAV correspondiente y la ruta que deberá seguir.

La GCS permitirá la apertura de múltiples ventanas que posibiliten la visualización de la información de los UAV en el mismo o en distintos ordenadores, de forma que pueda haber más de un usuario controlando y monitorizando los UAVs. Asimismo, proporcionará facilidades para que aplicaciones de terceros puedan establecer comunicación con los UAVs mediante la GCS, con el propósito de adquirir información y enviar comandos de alto nivel destinados al control de los vehículos aéreos no tripulados.

3.2 Arquitectura de la GCS

Dados los requerimientos del sistema se decide implementar una arquitectura cliente-servidor mediante el uso de tecnologías web. En la parte del cliente se tendrán una o varias interfaces gráficas a las que llamaremos MUAV-GUI y en la parte del servidor se encuentra la lógica y el control de los UAVs a la que llamaremos MUAV-LC. Estas dos partes se comunican a través de una API que

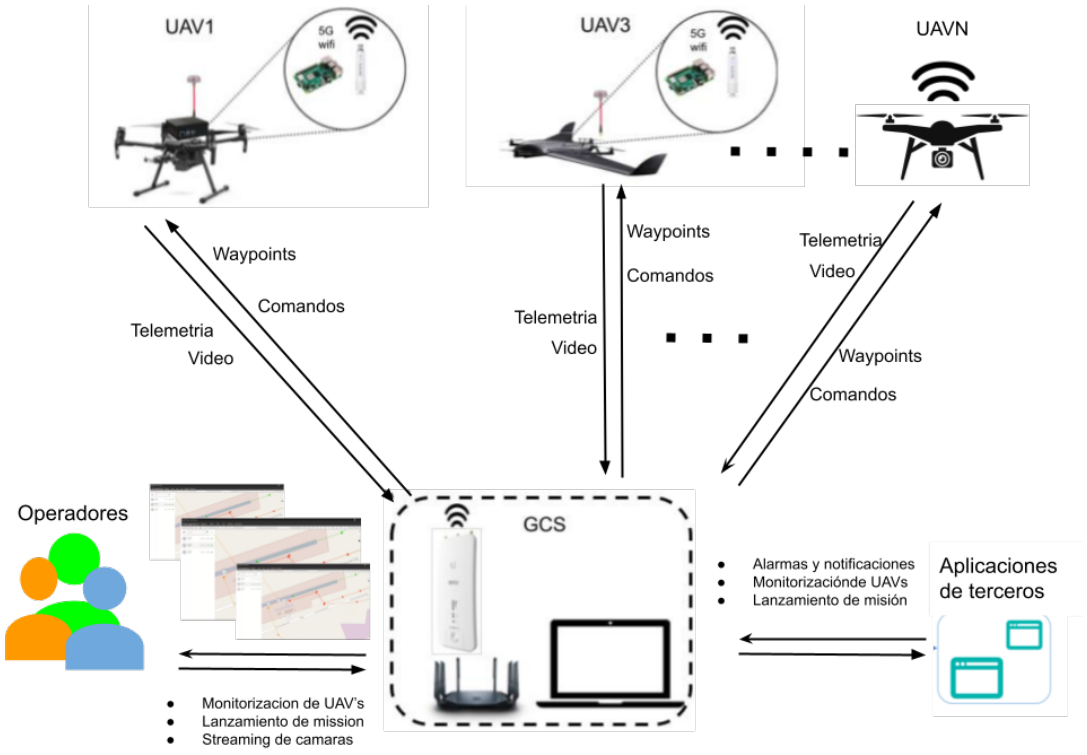

Figura 3.1 Diagrama de caso de estudio de la GCS. .

se utiliza para el envío de información y acciones al servidor, y WebSocket para la sincronización de información en tiempo real. Esta arquitectura se puede ver en la Figura3.2.

La arquitectura cliente-servidor permite tener varios clientes en la misma red de área local sin tener que realizar una configuración adicional. De la misma forma en que la MUAV-GUI puede comunicarse con la MUAV-LC, también se pueden comunicar aplicaciones externas mediante el uso de la API y WebSocket desarrollados. Para el desarrollo de la MUAV-GUI se ha usado HTML, CSS y Javascript con la librería REACT.js [18] ya que permite un desarrollo rápido y modular de los elementos que componen la interfaz, y posee una gran comunidad de desarrolladores.

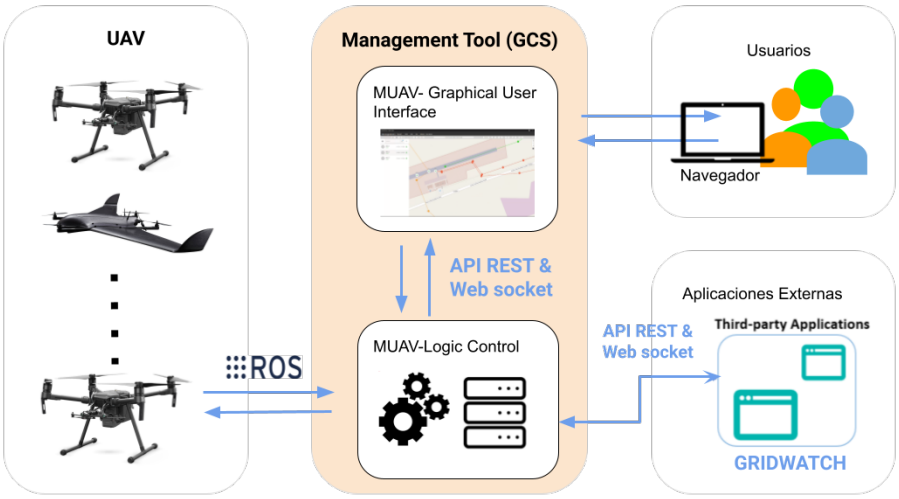

Figura 3.2 Arquitectura seguida en la implementación de la GCS.

En el caso del MUAV-LC, debido a que tiene que comunicarse con ROS se utiliza roslib.js que es una librería de Javascript desarrollada por RobotWebtools [21] que permite comunicarse a cualquier entorno de ROS donde se encuentre ejecutando el paquete Rosbridge[15] a través de WebSockets. La MUAV-LC está formada por dos entornos de desarrollo: uno en Node.js que se encarga de comunicarse con la MUAV-UV, gestionar los UAVs y adquirir los datos del entorno de ROS, y un entorno de ROS donde se tendrá un nodo de Rosbridge para la comunicación con la parte de NodeJS[10], un Nodo de Multimaster [27] con el cual se comunicara con los UAVs y un nodo de Maquina de estados, el cual administra las maquinas de estado de cada UAV, y representa estado de los UAVs de forma compacta. La arquitectura de la GCS se presenta de forma mas detallada en la Figura 3.3

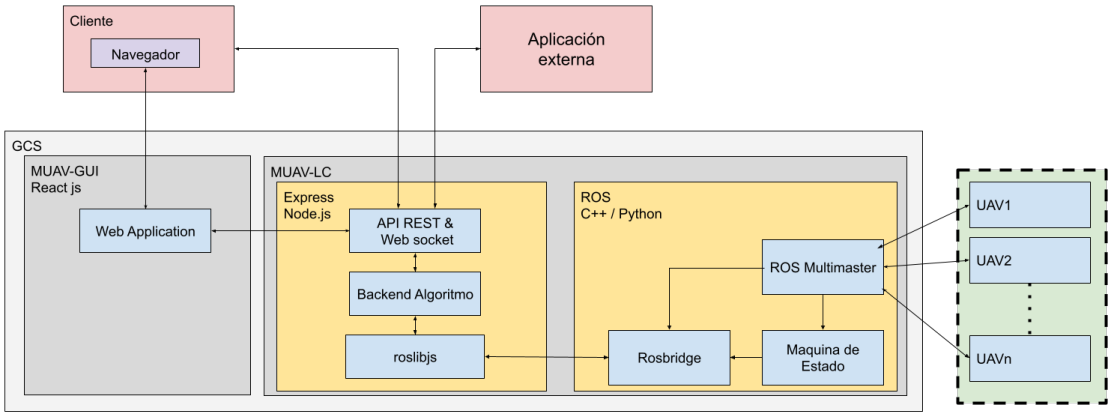

Figura 3.3 Detalles de la arquitectura de la GCS.

3.3 Adquisición de datos de los UAV

Para adquirir los datos de los UAVs en la GCS se utiliza roslib.js que es una librería que permite acceder a características de ROS a través del uso de WebSocket y posee la propiedad de poder suscribirse a topics y llamar a servicios. Para poder realizar esto la computadora donde se encuentra la GCS tiene que tener instalado ROS y se debe levantar un nodo de multimaster y un nodo de rosbridge. Esto se realiza mediante el Código 3.1, mientras que la configuración del paquete de multimaster se muestra en el Apéndice C.

Código 3.1 Comando para ejecutar Multimaster y Rosbridge en la GCS.

```
roslaunch aeriealcore_gui conect_uas.launch
```

Una vez levantados los nodos de ROS, se tendrá acceso a los topics de cada UAV que estén conectados a la misma red y que se encuentren ejecutando el nodo multimaster. Es necesario que todos los nodos de ROS en los UAVs se levanten con un NameSpace. El NameSpace sirve como un identificador de cada UAV y tiene que ser único entre los UAVs para que la GCS pueda identificar los topics y los servicios de cada UAV y los pueda añadir correctamente.

Para adquirir la información de telemetría el algoritmo de Backend se suscribirá a los topic de cada UAV mediante el uso de roslib.js, y para comandar se hará uso de los servicios de cada UAV. En la Tabla 3.1 se recogen los topics y servicios a los que se conecta la GCS para el DJI Matrice 210 y en la Tabla 3.2 se recogen los topics y servicios a los que se conecta la GCS para el DeltaQuad Pro.

Tabla 3.1 Topics y servicios del Matrice M210.

Tipo	dirección	Data	tipo
Topic	uav_id/dji_osdk_ros/rtk_position	Posición	sensor_msgs/NavSatFix
Topic	uav_id/dji_osdk_ros/rtk_yaw	Yaw	std_msgs/Int16
Topic	uav_id/dji_osdk_ros/velocity	Velocidad	geometry_msgs/Vector3Stamped
Topic	uav_id/dji_osdk_ros/battery_state	Bateria	sensor_msgs/BatteryState
Topic	uav_id/video_stream_compress	Streaming cámara	sensor_msgs/CompressedImage
Topic	/muav_sm/uav_id/uavstate	Estado UAV y misión	muav_state_machine/UAVState
Service	uav_id/dji_control/start_mission	Comandar misión	std_srvs/SetBool
Service	uav_id/dji_control/configure_mission	Cargar lista de Waypoints	aerialcore_common/ConfigMission
Service	uav_id/dji_control/send_bags	Sincronizar Rosbag	bool

Tabla 3.2 Topics y servicios del DeltaQuad.

Tipo	dirección	Data	tipo
Topic	uav_id/mavros/global_position/global	Posición	sensor_msgs/NavSatFix
Topic	uav_id/mavros/global_position/compass_hdg	Yaw	std_msgs/Float64
Topic	uav_id/mavros/local_position/velocity_local	Velocidad	geometry_msgs/Vector3Stamped
Topic	uav_id/mavros/altitude	Altitud	mavros_msgs/Altitude
Topic	uav_id/mavros/battery	Bateria	sensor_msgs/BatteryState
Topic	uav_id/video_stream_compress	Streaming cámara	sensor_msgs/CompressedImage
Topic	/muav_sm/uav_id/uavstate	Estado UAV y misión	muav_state_machine/UAVState
Service	uav_id/mission/start_stop	Comandar misión	std_srvs/SetBool
Service	uav_id/mission/new	Cargar lista de Waypoints	aerialcore_common/ConfigMission
Service	uav_id/mission/send_bags	Sincronizar Rosbag	bool

La adquisición de los datos de telemetría mediante la suscripción a los topic se realizará una vez se haya añadido el UAV a la plataforma y hasta que se elimine al UAV de la misma. En el caso de los servicios, se ejecutaran solo y cuando el usuario lo haya indicado mediante el uso de la interfaz MUAV-GUI o la API.

3.3.1 Streaming de video

El streaming de video se ha realizado de dos formas: una es mediante un topic de ROS usando el nodo de ROS simple_vs y la otra es usando WebRTC [26] con Gstreamer[24], Gstreamer es un framework multimedia libre que posee herramientas de tratamiento de vídeo, basado en una arquitectura de plugins, y permite procesar y transmitir datos multimedia.

3.3.2 Nodo simple_vs

Este nodo permite la transmisión de imágenes a través de ROS, suscribiéndose al streaming de vídeo que el autopiloto se encuentra publicando en un nodo de ROS, al cual le aplica una compresión base 64 transformando la imagen a una cadena de texto, reduciendo su peso y el coste de transmisión, para posteriormente publicarla en el topic uav_id/video_stream_compress. Para ejecutar la transmisión con simple_vs se puede ejecutar el Código 3.2.

Código 3.2 Comando del nodo simple_vs.

```
roslaunch simple_vs video_streamer_compress.launch uav_id:=1
```

Adicionalmente, simple_vs permite la transmisión de vídeo de una cámara que se encuentre conectada en el ordenador a bordo, en este caso es mejor utilizar WebRTC.

3.3.3 WebRTC

WebRTC (Web Real-Time Communication) es un conjunto de tecnologías y protocolos que permite comunicación punto a punto en tiempo real para intercambio de multimedia. Se usa principalmente para streaming de vídeo y vídeollamadas, y entre sus principales características esta la alta calidad y la baja latencia. WebRTC necesita un servidor de señalización para el intercambio de información y establecer la conexión entre las aplicaciones como se ve en la Figura 3.4 extraída de [1].

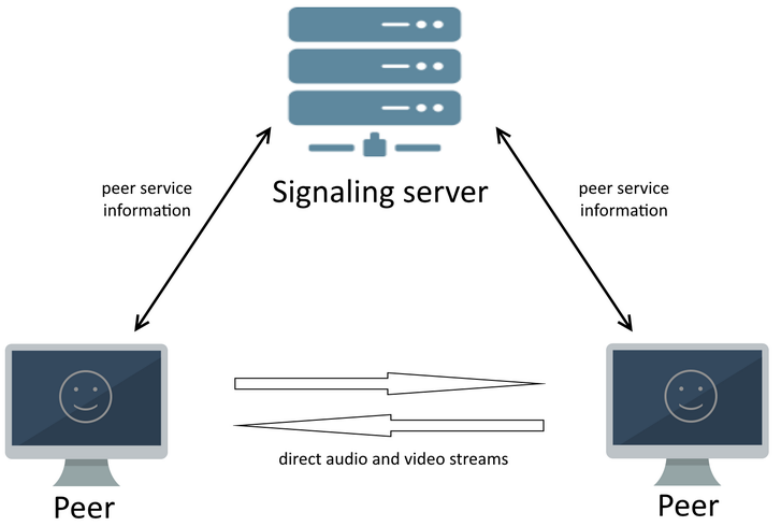

Figura 3.4 Comunicación de WebRTC estándar en una red de área local, sin firewall según Medialooks[1].

Para adaptar WebRTC en un entorno GCS - UAV, cuando el usuario necesite visualizar streaming de la cámara de un UAV, la aplicación de WebRTC de la MAUV-UI tiene que comunicarse con una aplicación WebRTC en el computador a bordo por lo que se utiliza Mediamtx. En la Figura 3.5 se

muestra como realizar la transmisión de vídeo desde el computador abordo hasta la MUAV-GUI.

Mediamtx es un servidor que permite al usuario publicar, leer y retransmitir en tiempo real secuencias de vídeo y audio. Este servidor recibirá la imagen de la cámara y esperará una conexión de WebRTC para trasmitir vídeo. De igual forma Mediamtx posee un servidor de señalización para permitir el traspaso de información para establecer la conexión con WebRTC.

Para capturar vídeo proveniente de las cámaras se utilizara Gstreamer ya que permite codificar el vídeo al formato H.264 que es compatible con WebRTC para luego enviarlo al servidor de vídeo Mediamtx. GStreamer es un framework multimedia de código abierto que proporciona una gran variedad de componentes para el manejo de multimedia. Además utiliza una arquitectura de tuberías (pipelines) para procesar y transmitir datos multimedia.

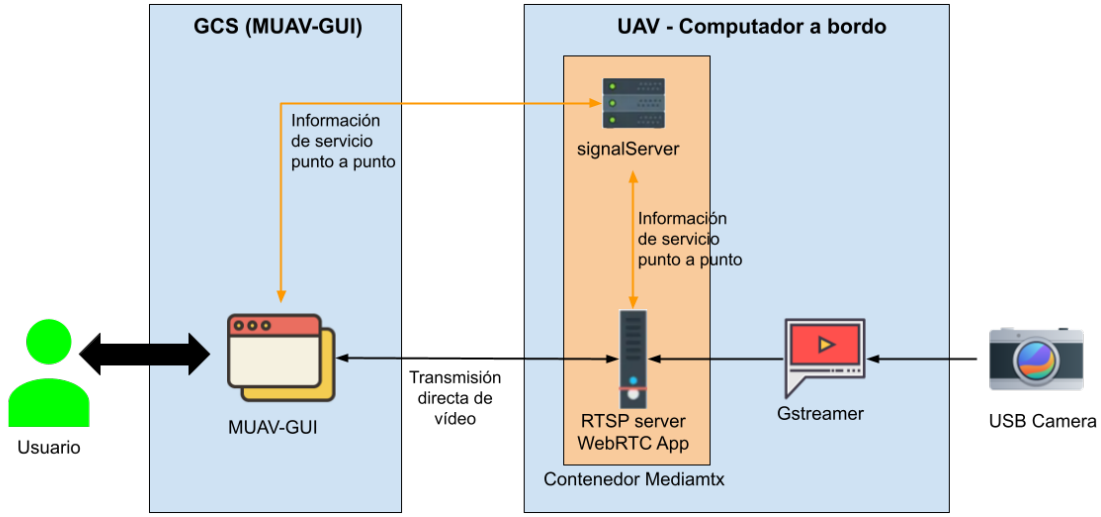

Figura 3.5 WebRTC para trasmisión de vídeo entre el computador a bordo y MUAV-GUI .

En el caso de la GCS, para que pueda recibir la transmisión de vídeo en directo solo es necesario que establezca la conexión y el canal vídeo entre el ordenador abordo del UAV y la MUAV. Para el ordenador a bordo del UAV es necesario tener instalado GStreamer y Docker[23], y el servidor de Mediamtx se ejecuta dentro de un contenedor de Docker.

Implementación

Para poder obtener el streaming de vídeo de cada una de las cámaras se tiene que ejecutar el servidor Mediamtx (ver Código 3.3).

Código 3.3 Comando para ejecutar el servidor de Mediamtx en el computador abordo.

```
docker run --rm -it --network=host -v aler9/rtsp-simple-server
```

También se tiene que ejecutar el Código 3.4 de Gstreamer para transmitir la información de la cámara al servidor de Mediamtx.

Código 3.4 Comando para trasmisión de vídeo de camara0 con GStreamer en Linux.

```
gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-raw,width=640,height
    =480 ! videoconvert ! x264enc bframes=0 tune=zerolatency bitrate=500
     speed-preset=superfast ! h264parse ! rtspclientsink location=rtsp
    ://0.0.0.0:8554/video0 rtpjpegpay name=pj
```

Dependiendo de la ubicación en la que se encuentre el vídeo de la cámara, se deberá cambiar el device=/dev/video0 por el correspondiente. De igual manera, hay que cambiar la ubicación a la que se publica el vídeo con location=rtsp://0.0.0.0:8554/video0.

3.4 Comunicación de la GCS

Para la comunicación ente la interfaz MUAV-GUI y el servidor MUAV-LC se requiere la implementación de una API y WebSocket.

3.4.1 API

Una API es un conjunto de reglas y protocolos que permiten que diferentes aplicaciones se comuniquen entre sí. En este caso fue necesario el desarrollo de una API-REST porque se utilizó la arquitectura REST que realiza solicitudes HTTP usando los métodos GET,POST, PUT,DELETE y PATH.

Cada vez que se ejecuta una acción en la interfaz de la GCS (MUAV-GUI), se envía una petición mediante la API a la MUAV-LC para que se ejecute la acción correspondiente. Por ello se han realizado varios recursos de la API-REST como se indica en la Figura 3.6. Para mayor información sobre la API desarrollada también se puede ver el Apéndice A.

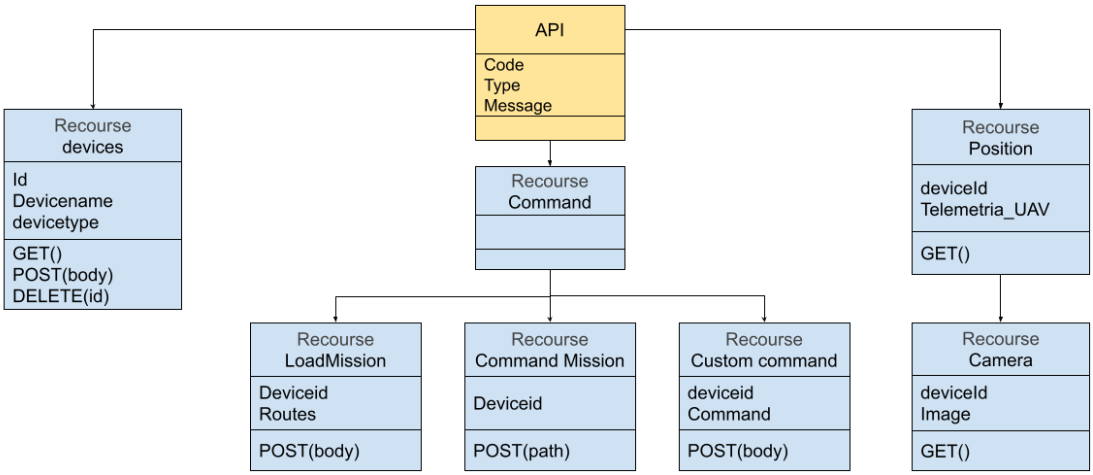

Figura 3.6 Diagrama UML de la API.

En el caso de las actualizaciones en tiempo real se recomienda usar WebSockets debido a su latencia reducida.

3.4.2 WebSockets

Los WebSockets son una tecnología de comunicación bidireccional y en tiempo real que permite una interacción continua entre un cliente y un servidor a través de una única conexión persistente. A diferencia de las solicitudes HTTP tradicionales, que son unidireccionales y requieren que el cliente realice solicitudes para obtener información actualizada, los WebSockets establecen una conexión

persistente que permite la comunicación en tiempo real y en ambos sentidos.

Los WebSockets son especialmente útiles para aplicaciones que requieren actualizaciones instantáneas, como aplicaciones de chat en vivo, juegos en tiempo real, aplicaciones de colaboración en línea, etc. Proporcionan una forma eficiente de enviar y recibir datos en tiempo real sin la necesidad de realizar solicitudes repetitivas al servidor.

Al abrir la interfaz MUAV-GUI se establece conexión con el WebSocket del MUAV-LC para obtener información y sincronizar datos. Una vez establecida la conexión, el servidor le envía enviar de manera continua la información de los dispositivos, posición (telemetría del UAV), estado del servidor e imágenes de cámara en el caso de que se esté utilizando el nodo simple_vs en el UAV, como lo indica la Figura 3.7.

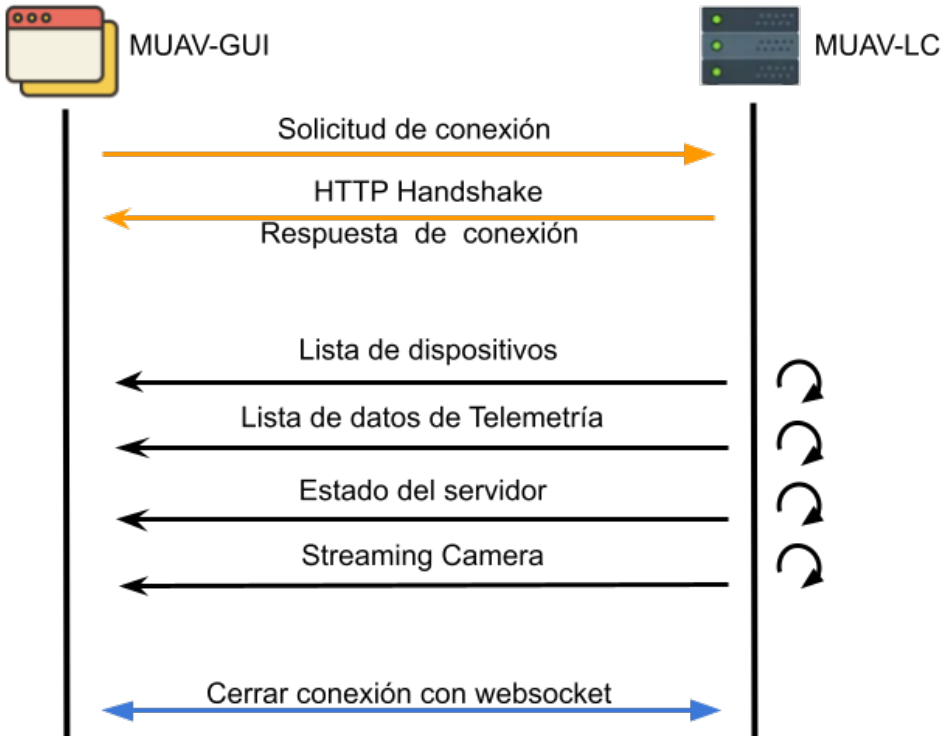

Figura 3.7 Diagrama de secuencia de WebSocket.

En el Apéndice B se encuentra detallada la información enviada por el servidor mediante el uso de WebSocket para la actualización de los datos de telemetría de cada UAV.

3.5 Interacción con el usuario

En el diagrama de la Figura 3.8 se muestran las principales acciones que puede realizar el usuario en la interfaz MUAV-GUI y como estas interactuaran con los componentes de la API REST, MUAV-LC y ROS. En rojo se encuentran las acciones que puede realizar el usuario en la MUAV-GUI, en amarillo las interacciones con la API, en azul las acciones realizadas por el MUAV-LC y en morado se muestran los procesos ejecutados en la interfaz.

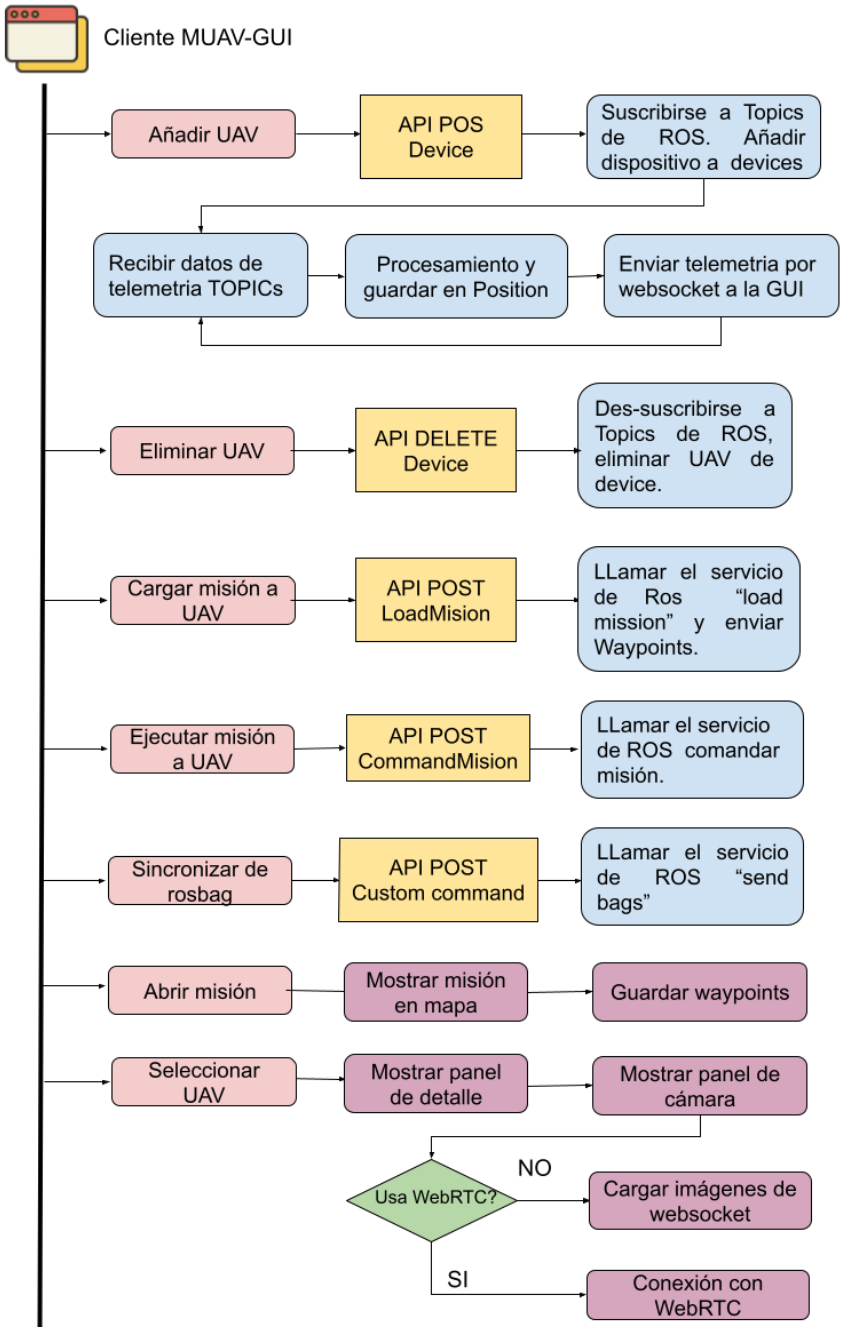

Figura 3.8 Diagrama de procesos simplificado..

3.6 Interfaz de usuario

La MUAV-GUI es una interfaz de una sola pagina que permite acceder a los datos de los UAVs que se encuentran registrados en la GCS. Para su desarrollo se han utilizado tecnologías web HTML, CSS y Javascript, permitiendo que varias MUAV-GUI se conecten al mismo MUAV-LC, lo que permite que haya varios usuarios monitorizando el sistema desde distintas maquinas.

Adicionalmente MUAV-GUI se desarrolló utilizando la librería REACT.js que permite dividir los elementos de la interfaz en componentes, facilitando el desarrollo de la aplicación, además de permitir construir aplicaciones con datos que cambian rápidamente manteniendo una buena experiencia de usuario.

El diseño de la interfaz permite visualizar los datos importantes de cada UAV mientras se muestra en un mapa la ejecución de las tareas de movimiento. Además permitirá poder acceder a información más detallada de cada UAV cuando se seleccione un UAV, como las imágenes de la cámara y datos de telemetría. La interfaz está formada por: barra General, Lista de UAVs, Mapa, Panel de cámara y Panel de detalle como se puede ver en la Figura 3.9.

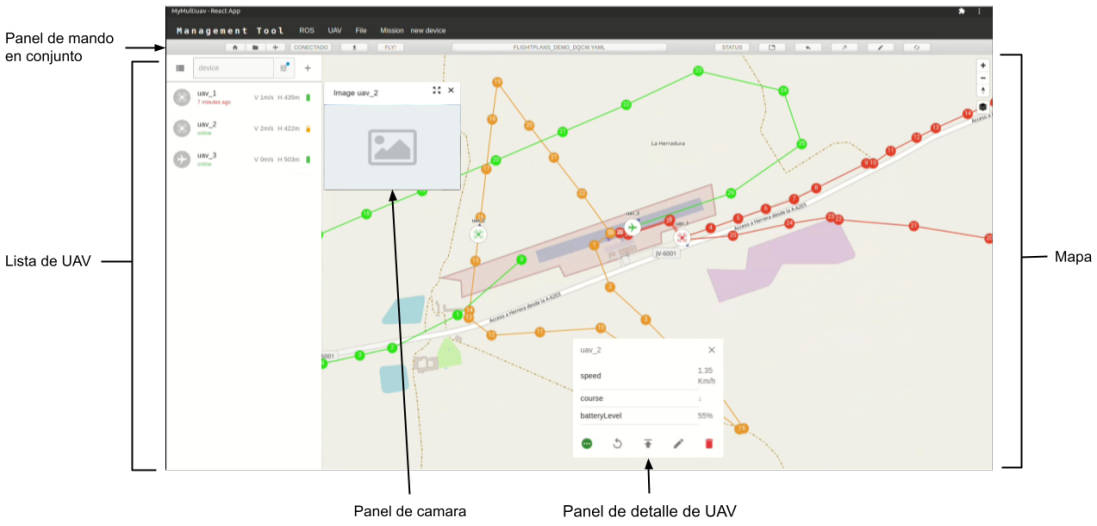

Figura 3.9 Partes de la interfaz de la GCS (MUAV-GUI).

3.6.1 Partes de la interfaz

Barra general

En la barra general se encuentran los botones y menús para el manejo de la GCS. Desde aquí se pueden añadir UAVs, comandar misiones y abrir misiones como se muestra en la Figura 3.10 donde se ven las funciones que permite realizar cada elemento.

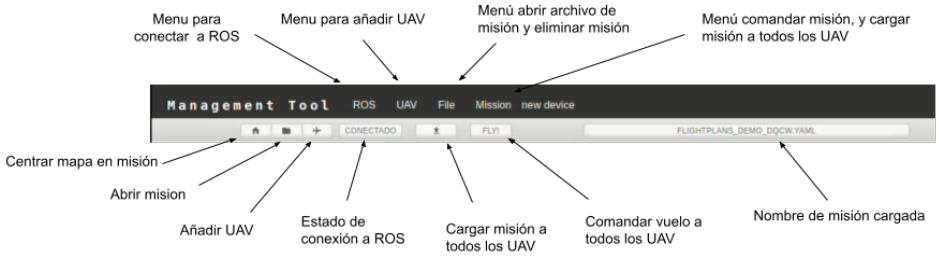

Figura 3.10 Barra general de la GCS (MUAV-GUI)..

Lista de UAVs

En la lista de UAVs se muestra información importante de forma intuitiva, de forma que no se sature al usuario. Para cada UAV se muestra el identificador del UAV, el estado del UAV, la altitud, la velocidad y el nivel de batería como se muestra en la Figura 3.11.

Mapa

En el mapa se muestra la posición de cada UAV, junto con la orientación como se indica en la Figura 3.11 de forma que el operador pueda monitorizar la tarea que se esta realizando. En el mapa también se visualizan las misiones, teniendo cada misión un color diferente y cada waypoint su numeración como se muestra en la Figura3.15.

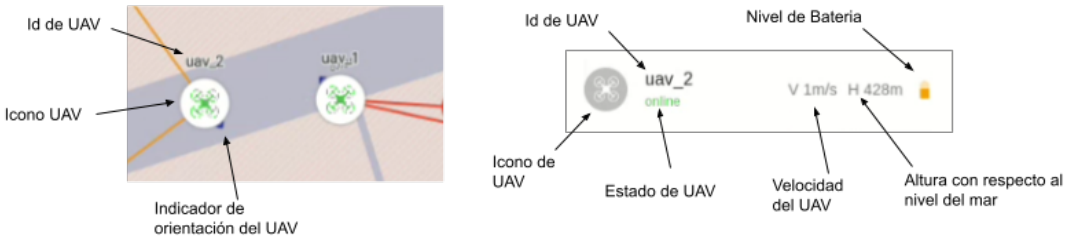

Figura 3.11 UAV en mapa y UAV en la lista de UAVs .

Panel de detalle

El panel de detalle aparece solamente cuando se encuentra seleccionado un UAV, mostrando información adicional del UAV seleccionado. Además permite ejecutar algunas acciones como se muestra en la Figura 3.12.

Panel de cámara

El panel de la cámara al igual que el panel de detalle aparece solamente cuando se encuentra seleccionado un UAV y muestra el streaming de vídeo como se muestra en la Figura 3.12. Además el panel se puede ampliar pulsando en el botón de ampliar imagen y en el caso de que no se logren obtener imágenes del streaming de vídeo se mostrará una imagen en gris.

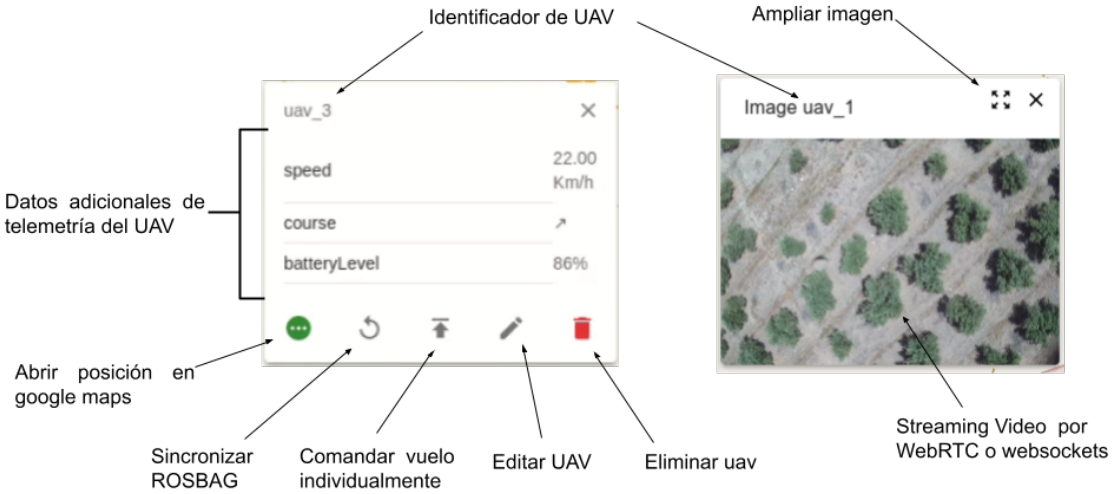

Figura 3.12 Panel de detalle y panel de cámara de la GCS (MUAV-GUI) .

3.7 Funcionamiento de la GCS

Para ejecutar la GCS es necesario tener instalado node.js V18 e iniciar la parte de ROS como se indicar en el Código 3.1. Para ejecutar la GCS se utiliza el Código 3.5, y una vez ejecutado se podrá

acceder a la interfaz mediante un navegador web mediante la url "http://localhost:4000". En el caso de las máquinas que se encuentren en la misma red y requieran acceder a la MUAV-GUI, podrán acceder mediante un navegador con la misma url cambiando localhost por la IP del ordenador donde se esta ejecutando la GCS.

Código 3.5 Ejecutar GCS.

```
npm run server
```

El procedimiento general al realizar la inspección con múltiples UAVs usando la herramienta desarrollada en este TFM se indica en la Figura 3.13, donde se pueden ver en color azul las actividades que tiene que realizar el operador en la MUAV-GUI. Estos procedimientos explicaran a continuación.

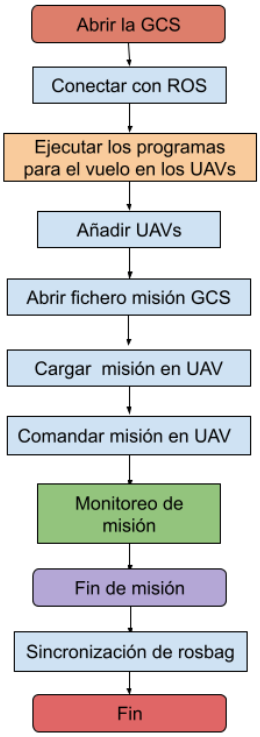

Figura 3.13 Proceso para la ejecución de una misión con la GCS .

Es necesario indicar que para poder conectar los UAVs a la herramienta cada UAV tiene que ejecutar los programas necesarios para el vuelo. En el caso del DJI, en la sección 4.3 se indica como hacerlo. Tanto la GCS como los UAVs tienen que tener configurado el nodo multimaster. Para mayor información sobre la configuración del multimaster se puede ver el Apéndice C.

3.7.1 Conexión a ROS

Para establecer una conexión entre la GCS y ROS, se debe hacer clic en la opción “Conectar con ROS” ubicada en la barra general tal como se muestra en la Figura 3.10. Este menú permite cambiar el estado de la conexión entre conectado y desconectado según sea necesario. Además, el estado de la conexión se reflejará en el indicador de estado de conexión a ROS. Este indicador debe mostrar el texto “Conectado” para que se puedan gestionar los UAVs de manera adecuada.

3.7.2 Añadir UAV

Para añadir los UAVs se puede hacer de dos formas: Se puede hacer clic en la barra general o en la lista de UAVs como se indica en la Figura 3.14. Al hacerlo aparecerá el panel para añadir UAVs donde se deben introducir el ID y el tipo de UAV, que son campos requeridos. La dirección IP es otro campo necesario en caso de que se utilice streaming de vídeo por WebRTC. Por defecto se encuentra seleccionado WebSocket para el streaming de vídeo que obtiene información del topic video_stream_compress. En el caso de WebRTC está seleccionado por defecto video0. Si se utiliza otra fuente de vídeo, habría que cambiarla.

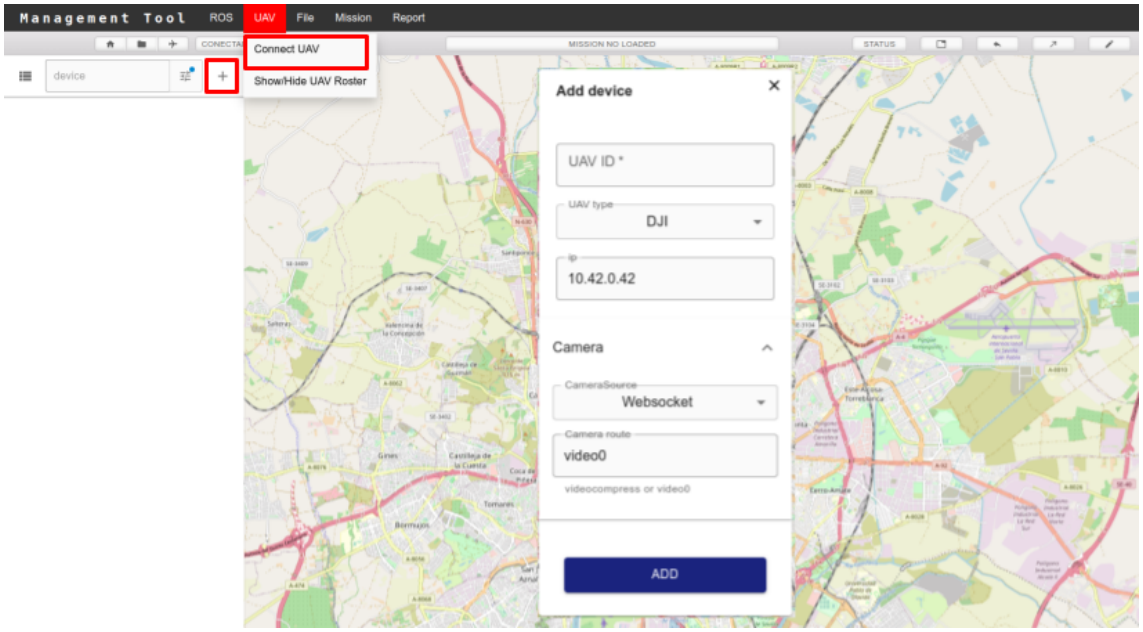

Figura 3.14 Añadir UAV a la GCS (MUAV-GUI) .

3.7.3 Abrir fichero de misión

Para abrir un fichero de misión hay que seleccionar en la barra general en File la opción Open Mission. Esto abrirá una venta para seleccionar el archivo de misión y cargarlo en el sistema. Una vez cargado el archivo de misión de podrá visualizar la ruta de cada UAV en diferente color, cada waypoint se encuentra enumerado según su orden y si se quiere conocer la información de cada waypoint se hace clic y aparece una ventana con información como se ve en la Figura 3.15.

3.7.4 Cargar misión en el UAV

Una vez se tenga cargado en el mapa la misión a ejecutar y todos los UAVs estén registrados en la GCS, se puede procede a cargar la misión a todos los UAVs. Se puede hacer de forma conjunta mediante clic en la barra general en el botón de cargar misión a todos los UAVs como se indica en la Figura 3.10.

3.7.5 Comandar misión al UAV

Para comandar la misión en los UAVs, se puede realizar de forma conjunta mediante clic en la barra general en el botón de comandar vuelo a todos los UAVs como se indica en la Figura 3.10. También se puede comandar el vuelo de forma individual dentro del panel de detalle con el botón comandar vuelo individualmente como lo indica la Figura 3.12.

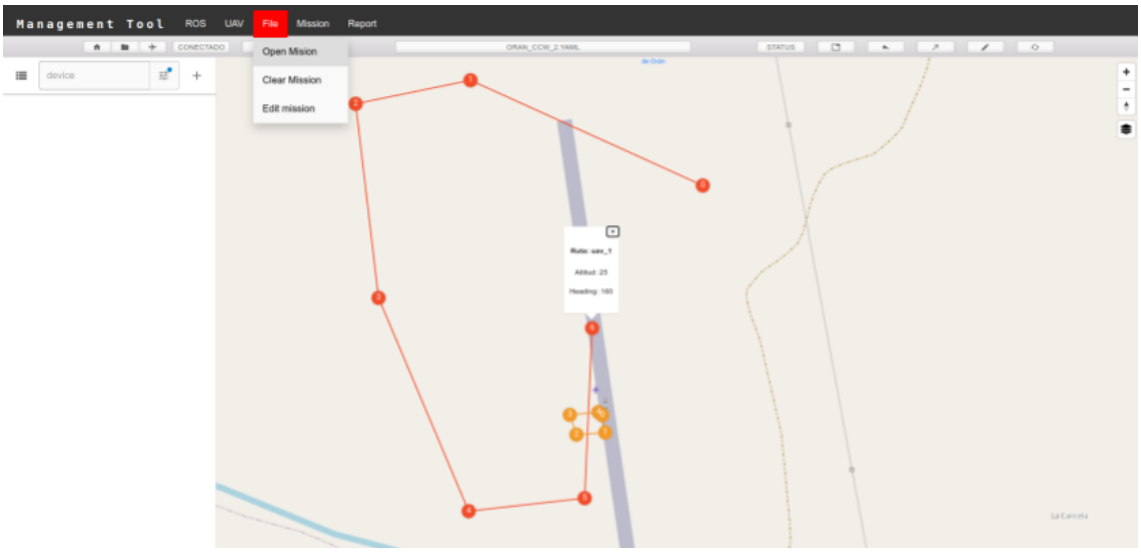

Figura 3.15 Abrir misión en la GCS (MUAV-GUI) .

3.7.6 Sincronización de Rosbag

Una vez acabadas las misiones de los UAVs, es necesario descargar los datos de los vuelos de los UAVs para procesar los datos. Esto se realiza de forma individual en el panel de detalle con el botón de sincronizar RosBag como se ve en la Figura 3.12.

3.8 Archivo de Misión

La asignación de tareas a cada UAV se realiza mediante un archivo de misión que es interpretado por la GCS para poder mostrarlo en la interfaz, cargarlo y comandarlo a cada UAV. Se ha desarrollado un formato de misión utilizando el lenguaje de señalización de datos YAML.

El archivo de misión debe tener un formato en el que uav_n indica el numero del UAV involucrado en la misión. Para asignar una tarea a un UAV se escribe uav_id donde id tiene que ser un numero entero mayor que cero. Yaml es un lenguaje en el que se tienen que respetar los espacios. Dentro de uav_id: se indica el número de waypoints de la tarea en c wp_n, y después se listan cada uno de los waypoints como wp_id donde id hace referencia al orden y tiene que ir enumerado en forma secuencial empezando en cero. Cada waypoint lleva asociado 4 items en el siguiente orden: latitud, longitud, altitud y orientación.

Se muestra un ejemplo de misión en el Código 3.6. Se corresponde con el archivo mission.yaml ejecutado en las pruebas de la Plaza del Agua situada junto al Edificio de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. Se tienen 2 UAVs: el primero con id uav_1, 4 waypoints, velocidad de 5 m/s y con aterrizaje por GNSS; el segundo tiene id uav_2 con con 5 waypoints, una velocidad de 5 m/s y acaba la misión en el último waypoint.

Código 3.6 Formato de mision, mision.yaml.

```
uav_n: 2
uav_1:
  wp_n: 4
  wp_0: [38.13921783,-3.17343016,40.37417603, 0]
```

```
  wp_1: [38.13920214,-3.17311366,40.78244019, 0]
  wp_2: [38.13963168,-3.17143225,43.22061157, 0]
  wp_3: [38.13907030,-3.17094522,41.67050171, 0]
  idle_vel: 5.00000000
  mode_landing: 5

uav_2:
  wp_n: 5
  wp_0: [38.13920996,-3.17380020,30.56164551, 0]
  wp_1: [38.13883219,-3.17445106,28.79190063, 0]
  wp_2: [38.13752204,-3.17381791,28.88183594, 0]
  wp_3: [38.13649470,-3.17239786,32.05810547, 0]
  wp_4: [38.13539110,-3.17102625,38.47814941, 0]
  idle_vel: 5.00000000
  mode_landing: 0
```

Además de los waypoints también se pueden usar parámetros de configuración como idle_vel, mode_landing, y yaw_mode, siendo estos dos últimos solo compatibles con aeronaves del fabricante DJI. El parámetro idle_vel indica la velocidad de vuelo en m/s y mode_landing el tipo de aterrizaje como se ve en la Tabla 3.3. Las opciones para yaw_mode se muestran en la Tabla 3.4. Estos parámetros de configuración se puede omitir y el UAV tomará los valores por defecto.

Tabla 3.3 Modos de orientación en DJI.

Modo	descripción
0	modo automático, el UAV se pone en dirección al punto siguiente
1	se bloquea con respecto a la orientación inicial
2	controlado por el radio control
3	orientación en función de los waypoints de la misión

Tabla 3.4 Modos de aterrizaje en DJI.

Modo	descripción
0	Sin aterrizaje autónomo
1	Acaba la misión aterriza en Home
2	Aterriza automáticamente en el ultimo punto
3	Acaba la misión y retorna al punto inicial

4 Configuración para la integración de aeronaves DJI

En este capitulo se detalla la configuración del computador abordo y del UAV necesarias para poder trabajar con las herramientas de desarrollo de DJI. Además se describirá el paquete que permite recibir comandos de la GCS en el DJI Matrice M210 RTK V2.

4.1 Configuración del entorno de desarrollo de DJI

Para poder comunicar el ordenador a bordo con el autopiloto de DJI se requiere utilizar las herramientas de desarrollo de DJI el OSDK y ROS-OSDK, para lo que es necesario configurar el dron y el ordenador a bordo.

4.1.1 Configuración del dron

Para configurar y habilitar la comunicación con el ordenador a bordo en Matrice 210 V2 RTK se utiliza el programa DJI Assistant 2, en la pestaña “Onboard SDK” donde se habilita “Enable API Control” y se cambia “Baud Rate” a 921600 como se muestra en la Figura 4.1.

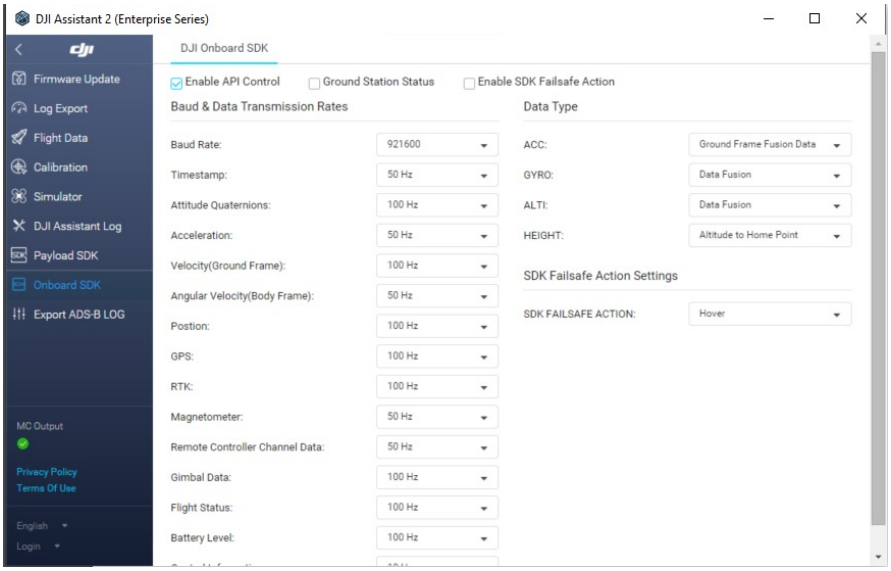

Figura 4.1 Configuraciones de la Onboard SDK con DJI Assistant 2..

Adicionalmente es necesario registrarse como desarrollador de DJI y crear un id de aplicación de OSDK y una clave que se utilizarán para que el computador abordo se comunique con el UAV. Para más información se puede revisar la documentación [4] de la configuración del entorno de desarrollo de DJI.

También es necesario en el mando del drone habilitar las opciones que se muestran en la Figura 4.2, “Enable Power Supply Port” habilita el puerto “External Power” para alimentar el ordenador a bordo y la opción “Enable OSDK Communication Port” habilita la transmisión serie del autopiloto del UAV.

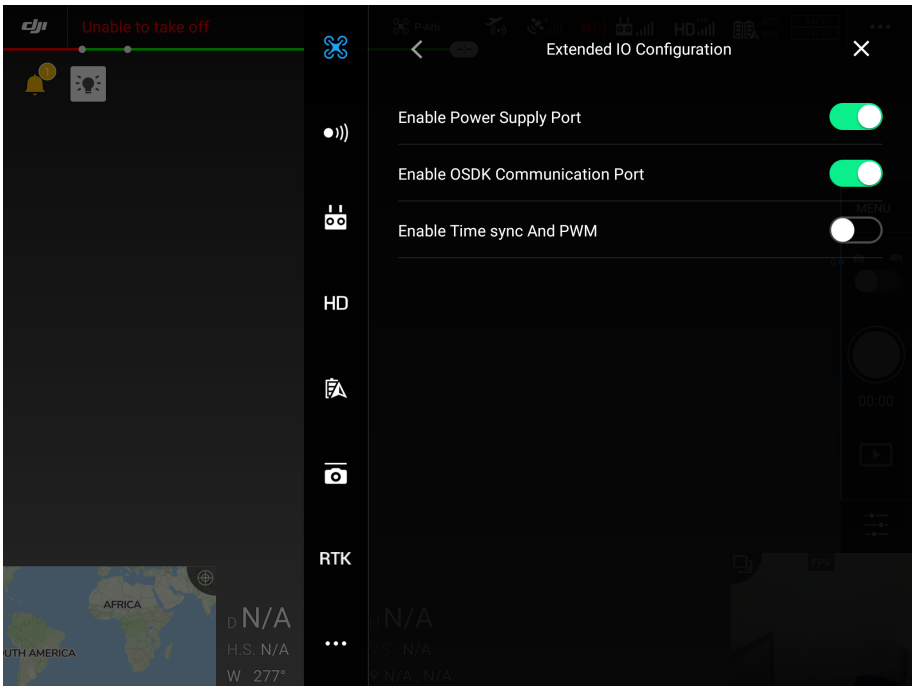

Figura 4.2 Configuración extendida de entradas y salidas DJI.

4.1.2 Conexión UAV y ordenador a bordo

La Figura 4.3 indica la conexión entre el ordenador a bordo y el Matrice 210 V2 RTK. El Matrice 210 posee 3 puertos para la conexión del ordenador abordo: un puerto para alimentación, un puerto de comunicación serie para conectar el ordenador a bordo con el autopiloto, y un puerto USB donde se levanta una conexión Ethernet para acceder a la información de la cámaras que se encuentren conectadas al UAV.

Para la conexión entre el UAV y el ordenador a bordo se utiliza un cable USB macho a USB macho para la conexión del puerto USB, y para la comunicación serie se usa un módulo de conversión serie a USB. Es importante que el “switch mode” en la Figura 4.3 se encuentre a la derecha para habilitar el envío de información por el puerto USB del dron.

4.1.3 Instalación de herramientas desarrollo en ordenador a bordo

El OSDK es una librería desarrollada en C++ que permite comunicar el ordenador a bordo con la plataforma de vuelo de DJI y el controlador de vuelo a través comunicación serie. El OSDK permite acceso a las características y funcionalidades, permitiendo al desarrollador automatizar el vuelo, el control de la cámara y el gimbal y monitorizar el estado del dron.

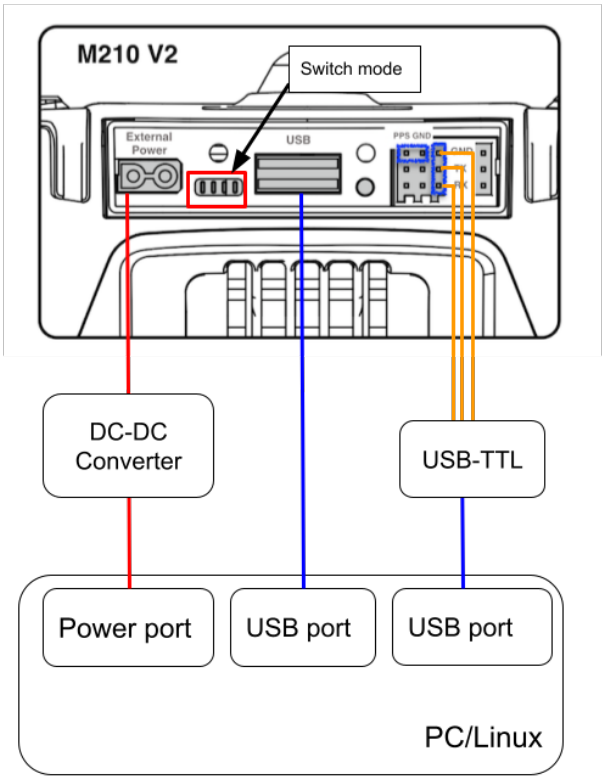

Figura 4.3 Conexión de Mochila DJI Matrice M210 .

Las principales funcionalidades del OSDK pueden observarse en la Figura 4.4 que se ha extraído de la referencia [5]. Todas estas funcionalidades las ofrece el OSDK y se pueden utilizar en ROS mediante el ROS-OSDK.

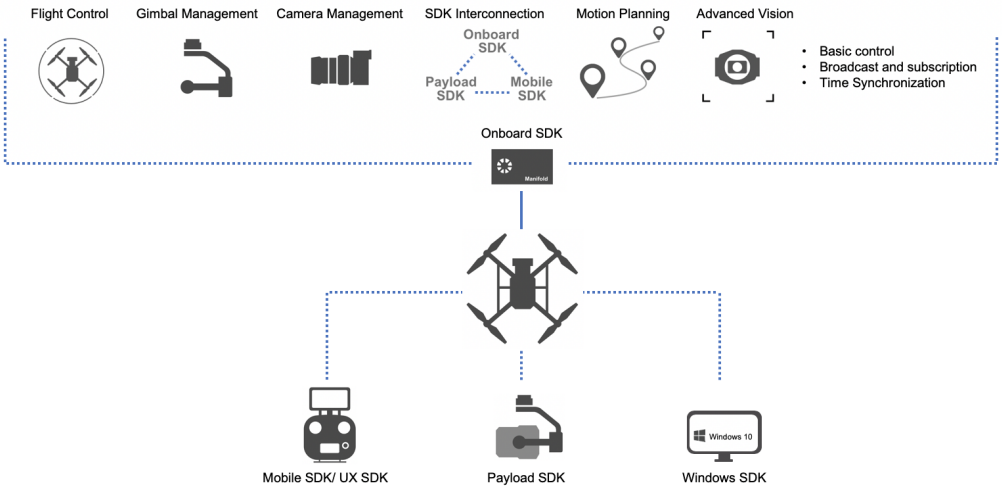

Figura 4.4 Funcionalidades de la Onboard-SDK .

Instalación de la OSDK

El Código 4.1 contiene los comandos de instalación de la OSDK de DJI. En la compilación se añade el argumento de DADVANCED_SENSING para que se compilen los archivos que permiten al ordenador abordo tener acceso a la información de las cámaras.

Código 4.1 Compilacion del OSDK.

```
git clone https://github.com/dji-sdk/Onboard-SDK.git
cd Onboard-SDK
mkdir build
cd build
cmake .. -DADVANCED_SENSING=ON
sudo make -j7 install
```

Para que la OSDK se pueda conectar con el UAV se necesita el archivo UserConfig.txt. En este archivo se coloca el id de aplicación de la OSDK y la clave para que se establezca la comunicación con el UAV. Este archivo se deberá colocar en la carpeta build/bin. En el Código 4.2 se copia la plantilla del archivo de configuración de usuario a la carpeta requerida.

Código 4.2 Archivo de configuracion.

```
cp ../sample/linux/common/UserConfig.txt bin/
```

Una vez realizados los pasos anteriores se puede comprobar que se ha configurado e instalado la OSDK correctamente mediante el Código 4.3.

Código 4.3 Prueba de funcionamiento de OSDK.

```
cd bin
./djiosdk-flightcontrol-sample UserConfig.txt
```

Cada que se encienda el UAV es necesario ejecutar el Código 4.4 para que el ordenador a bordo tenga control sobre el UAV.

Código 4.4 Configuracion de UAV.

```
/home/grvc/programming/Onboard-SDK/utility/bin/armv8/64-bit/
   M210ConfigTool --usb-port /dev/ttyACM0 --config-file /home/grvc/
   programming/Onboard-SDK/built/bin/UserConfig.txt --usb-connected-
   flight on
```

Instalación de ROS-OSDK

Para la instalación de ROS-OSDK se tiene que instalar las dependencia mediante el Código 4.5, y adicionalmente en el workspace de ROS se deberá colocar el paquete ROS-OSDK clonando el repositorio como se indica en el Código 4.6 y compilar el WorkSpace de ROS usando catkin_make.

Código 4.5 Instalacion de ROS-OSDK.

```
sudo apt install ros-melodic-nmea-comms
sudo apt-get install libavcodec-dev libswresample-dev
```

```
sudo apt install ffmpeg
sudo apt install libusb-1.0-0-dev
sudo apt install libsdl2-dev
sudo apt-get install libopencv-dev
```

Código 4.6 Instalacion de ROS-OSDK.

```
git clone https://github.com/dji-sdk/Onboard-SDK-ROS.git
```

Una vez compilado el workspace de ROS se modifica el archivo dji_vehicle_node.launch y se cambian los valores de App_id, key, Baudrate y puertos con los correspondientes al dron. Una vez realizado eso ya se podrá ejecutar el comando 4.7 que levantará los topics y servicios de ROS del ROS-OSDK.

Código 4.7 Instalacion de ROS-OSDK.

```
roslaunch dji_osdk_ros dji_vehicle_node.launch
```

4.2 Funcionalidad del ordenador abordo

Para que el UAV realice las tareas de inspección y siga las trayectorias comandas por la GCS, en el ordenador embarcado se utilizan los paquetes multimaster, ROS-OSDK y onboard_dji. El paquete onboard_dji se encarga de consumir los servicios y topics de ROS-OSDK, y utilizar la clase FlightTaskControl del paquete OSDK para poder subir misiones al controlador de vuelo del Matrice M210 V2 RTK.

En la Figura 4.5 se puede visualizar la estructura de los procesos que se ejecutan en el ordenador abordo y en la GCS. El ordenador a bordo utiliza el paquete multimaster-fkie para que en la GCS se pueda acceder a los topics y servicios de ROS del UAV.

El nodo de misión es el encargado de recibir la misión de la GCS y enviarla al dron. Esto lo realiza mediante los servicios de ROS dji_control/configure_mission, dji_control/start_mission. Además este nodo levanta un rosbag service cada vez que inicia una misión con la finalidad de poder guardar los datos de la misión para poder reproducirlos posteriormente. rosbag service se ejecuta cuando se inicia la misión y termina cuando aterriza el UAV creando un archivo en la carpeta /home/user/bags con la información del vuelo. El nodo de misión también posee el servicio dji_control/send_bag que sirve para sincronizar la información de respaldo de cada misión entre el ordenador abordo y la GCS. Para ejecutar el nodo de misión es necesario tener el multimaster y el ROS-OSDK ejecutándose al mismo tiempo en el UAV. En la Sección 4.3 se indica como ejecutar el nodo de misión.

Código 4.8 Comando para ejecutar el nodo de misión.

```
roslaunch onboard_dji dji_mission.launch uav_id:=1
```

La máquina de estados gestiona los estados del UAV y de la misión. Obtiene información del autopilo mediante los topic del ROS-OSDK y se comunica con la máquina de estados de la GCS que se encarga de encapsular los estados de los UAVs para mostrarlos en la GCS.

Además se pueden levantar diferentes nodos adicionales según se requiera. En el caso del streaming de vídeo se puede levantar el nodo simple_vs para transmisión de vídeo a través de websocket o realizar el streaming con WebRTC como se indica en la Sección 3.3.1.

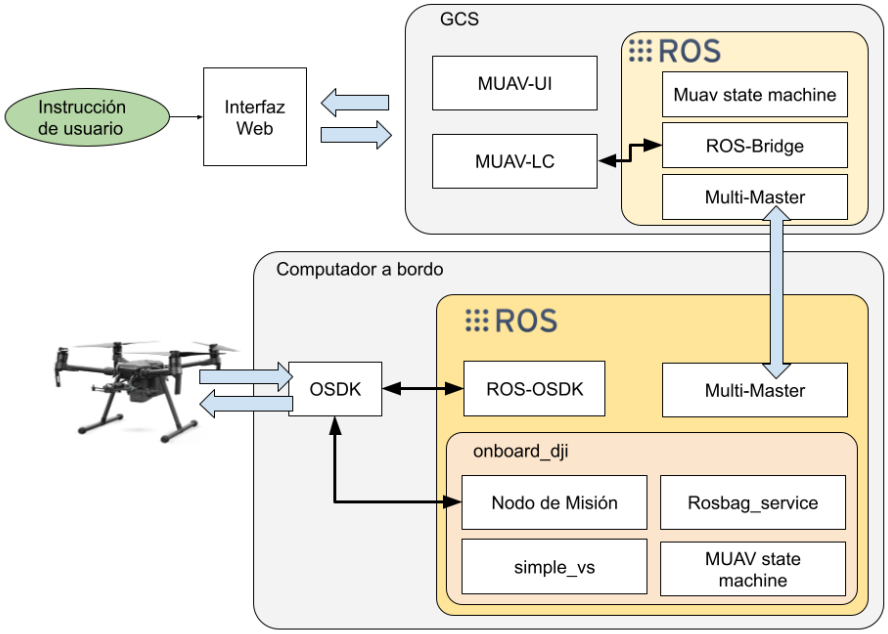

Figura 4.5 Estructura del ordenador a bordo y GCS.

4.3 Preparación para el vuelo

Para la preparación del vuelo es necesario conectarse por shh al UAV y ejecutar los los Codigos 4.9 y 4.10. Para la comunicación con la estación de control se utiliza el Código 4.9 que usa el paquete multimaster para que todos los topics y servicios que se levanten en el ordenador a bordo se vean reflejados en la GCS. Para configurar el paquete multimaster se pueden seguir las instrucciones indicadas en el Apéndice C.

Código 4.9 Comando multimaster.

```
roslaunch onboard_dji multimaster.launch
```

El Código 4.10 ejecuta el archivo atlas.launch que es el encargado de levantar los nodos mission_node y dji_vehicle_node dentro de un NameSpace. El nodo dji_vehicle_node publica información de telemetría del autopiloto en topics de ROS y crea servicios de ROS para configurar y comandar al UAV. El nodo mission_node levanta los servicios para recibir comandos de la GCS y controlar al UAV.

Código 4.10 Comando multimaster.

```
roslaunch onboard_dji atlas.launch uav_id:=1
```

En el caso de que también se requiera transmitir vídeo desde el UAV a la estación de control es necesario ejecutar el nodo simple_vs o WebRTC como se indica la Sección 3.3.1.

5 Pruebas y Validación

En este capitulo se explican las pruebas realizadas para la comprobación del funcionamiento de la herramienta desarrollada en este trabajo. Esta herramienta se probó con sistemas multi-UAV simulados y en vuelos reales.

5.1 Simulación

La simulación es una herramienta fundamental en el desarrollo de software, ya que permite validar nuevas características y funcionalidades que se están implementando en un sistema, disminuyendo el tiempo de desarrollo especialmente cuando las pruebas del sistema son complejas y pueden llevar mucho tiempo.

Es importante que cuando se prueben sistemas complejos se haga con condiciones muy cercanas a la realidad donde se integre el hardware y el software en la simulación, para lo cual existe técnicas como Hardware in the loop(HITL) y software in the loop (SITL). En SITL se trata de simular el comportamiento del hardware del sistema utilizando modelos de software, de forma que se puede probar el hardware en diferentes condiciones y sin tenerlo físicamente. En HITL se utiliza el hardware real junto a un entorno de simulación para probar la interacción del software con el hardware.

Los UAVs usados para el desarrollo de la GCS tienen distintas herramientas y funcionalidades para las pruebas y las simulaciones: el Matrice M210 V2 permite simulación HITL sobre el dron y el DeltaQuad posee un entorno para la simulación en software in the loop.

5.1.1 Sofware in the loop para el VTOL DeltaQuad

El DeltaQuad posee el autopiloto PX4 que provee la posibilidad de usar STIL con uno o varios vehículos en el entorno de simulación 3D Gazebo[14]. Se puede instalar de varias formas según la documentación de PX4[28]. Se recomienda utilizar Docker con el autopiloto fuera del contenedor como lo indica el Código 5.1, donde en “local_src” se coloca la dirección donde se clonó el repositorio de github de PX4-Autopillot.

Código 5.1 Comando para simular el VTOL.

```
docker run -it --rm \
-v <local_src>:/home/user/Firmware:rw \
-v /tmp/.X11-unix:/tmp/.X11-unix:ro \
-e DISPLAY=\${DISPLAY} \
-e LOCAL_USER_ID="\$(id -u)" \
--name=container_name px4io/px4-dev-ros-noetic /bin/bash
```

Una vez se tenga acceso al contenedor, en la carpeta del autopiloto se podrá ejecutar el Código 5.2 para simular el DeltaQuad. En una ventana aparece el entorno de simulación Gazebo donde se podrá ver el comportamiento del UAV como se indica en la Figura 5.1.

Código 5.2 Comando para simular el VTOL.

```
export PX4_HOME_LAT=38.13931349915096
export PX4_HOME_LON=-3.173436419425258
export PX4_HOME_ALT=445
make px4_sitl_default gazebo_standard_vtol
```

Es importante cambiar la latitud y la longitud para que correspondan al punto de despegue de la misión a ejecutar. Para levantar los nodos necesarios para que el UAV reciba comandos de la GCS se ejecuta el Código 5.3.

Código 5.3 Comando para conectar VTOL con los servicios de ROS y el nodo de misión.

```
roslaunch aerialcore_gui px4-simulation.launch uav_id:=1
```

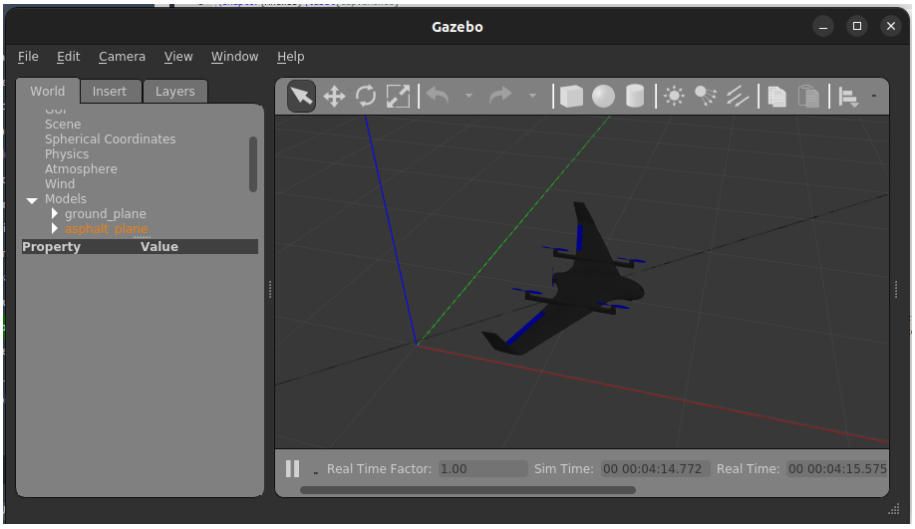

Figura 5.1 SITL DeltaQual PX4-Autopiloto .

5.1.2 Hardware in the loop con el DJI

El Matrice 210 V2 RTK permite realizar simulación HITL, por lo que se usa el UAV de la misma forma que en un vuelo real. Se ejecutan los mismos comandos solo que esta vez no se mueve el UAV, sino que el propio hardware simula su funcionamiento. Para colocar al UAV en modo simulación se ejecuta el Código 5.4 en el ordenador a bordo. Es importante cambiar la latitud y la longitud de forma que coincidan con los puntos de despegue de la misión a simular.

Código 5.4 Comando para colocar al DJI en modo Hardware in the loop.

```
/home/grvc/programming/Onboard-SDK/utility/bin/armv8/64-bit/
    M210ConfigTool --usb-port /dev/ttyACM0 --config-file /home/grvc/
    programming/Onboard-SDK/built/bin/UserConfig.txt --simulation on --
    latitude 37.53395183 --longitude -6.30202933 --usb-connected-flight
    on
```

Una vez ejecutado el Código 5.4 se pueden levantar los nodos de ROS necesarios para la comunicación con la estación de control. Esto se hace de la misma forma que cuando se prepara al UAV para realizar un vuelo real como se indica en la Sección 4.3.

5.2 Pruebas de vuelo reales

Una vez que la GCS se ha desarrollado y probado en simulación, se realizaron pruebas reales en diferentes entornos, con el objetivo final de realizar una misión multi-UAV de inspección de lineas eléctricas en el contexto del proyecto H2020 AERIAL-CORE.

El propósito de las pruebas reales consiste en que la herramienta desarrollada supervise y controle tres vehículos aéreos no tripulados con la tarea de inspeccionar líneas eléctricas. Cada uno de los UAVs recorre una línea eléctrica distinta, ejecutando su tarea de manera simultánea. Entre los UAV tenemos dos DJI Matrice M210 RTK (uno equipado con aterrizaje por visión y otro con aterrizaje por GNSS) y un DeltaQuad con aterrizaje por GNSS.

5.2.1 Pruebas preliminares

Las pruebas preliminares se realizaron en la Plaza del Agua (ver Figura 5.2) situada cerca de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. Este es un espacio aéreo controlado donde se realizaron varias pruebas con misiones de 4 ó 5 waypoints con un solo UAV para probar el funcionamiento de la interfaz y la comunicación con el UAV.

Figura 5.2 Plaza del Agua - ETSI de la Universidad de Sevilla .

También se realizaron pruebas con dos Matrice M210 RTK donde un UAV posee aterrizaje por GNSS y el otro realiza aterrizaje basado en visión artificial.

En la Figura 5.3 se puede visualizar la misión realizada por los dos Matrice 210 donde el UAV con aterrizaje por GNSS posee el namespace uav_1 y el que hace aterrizaje por visión tiene el namespace

uav_2. En esta configuración se logró monitorizar la misión y comandar de forma simultánea los UAVs, realizando de forma correcta el aterrizaje por GNSS y el aterrizaje por visión.

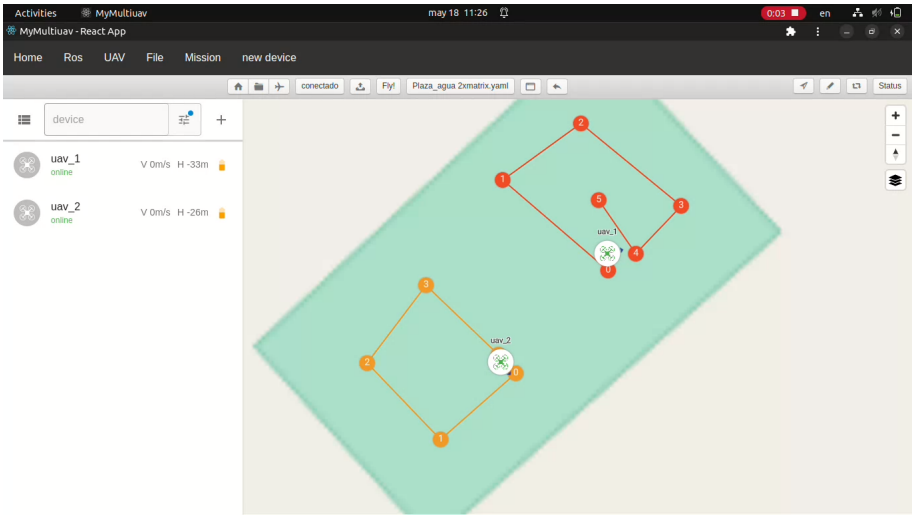

Figura 5.3 Pruebas de la GCS con dos Matrice M210 V2 en la Plaza del Agua .

Para comprobar el funcionamiento del DeltaQuad con la GCS se realizaron pruebas en la Hacienda Oran como se ve en la Figura 5.4 debido a que la aeronave VTOL necesita un mayor espacio para desarrollar una misión completa.

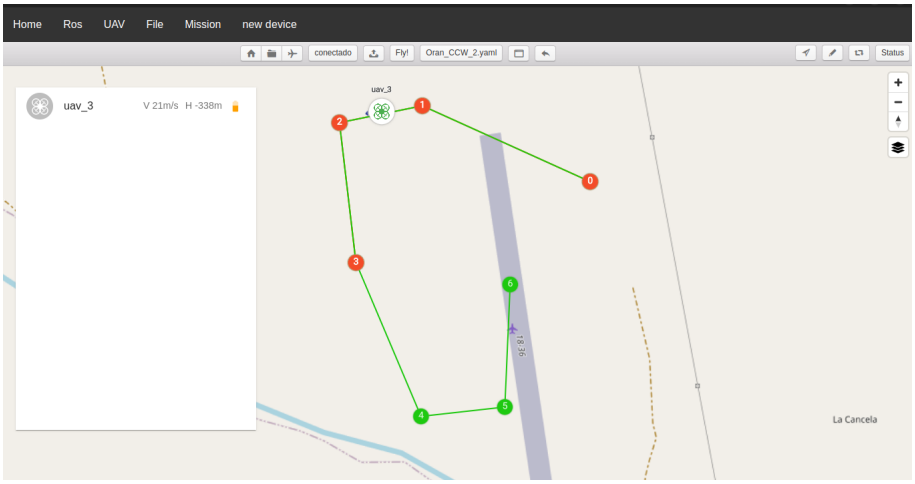

Figura 5.4 Pruebas de la GCS con DeltaQuad en la Hacienda Oran .

5.2.2 Pruebas en el campo de vuelo experimental ATLAS

Estas pruebas forman parte de la validación realizada en el proyecto H2020 AERIAL-CORE, donde se realizó la inspección de lineas eléctricas de Endesa ubicadas en el campo de vuelo experimental ATLAS. En estas pruebas de vuelo se utilizó la herramienta desarrollada para la supervisión y el control de vuelo de dos DJI Matrice M210 RTK, uno equipado con aterrizaje por visión y otro con aterrizaje por GNSS, y un DeltaQuad con aterrizaje por GNSS.

Antes de realizar la misión en conjunto con los tres UAVs, cada UAV realizó la misión de forma individual. Por ejemplo, en la Figura 5.5 se puede observar la misión llevada a cabo por el uav_2

que realizó el aterrizaje basado en GNSS. Durante esta operación, se pudo determinar que las comunicaciones permiten mantener una conexión de hasta aproximadamente 1,2 kilómetros en campo abierto y sin interferencias electromagnéticas externas. Una vez que el dron se aleja de este rango, la GCS deja de recibir datos e indica que el dron se encuentra en estado de desconexión. Sin embargo, una vez que el dron vuelve a estar dentro del rango de cobertura de comunicación, se reconecta automáticamente y se restablece la telemetría.

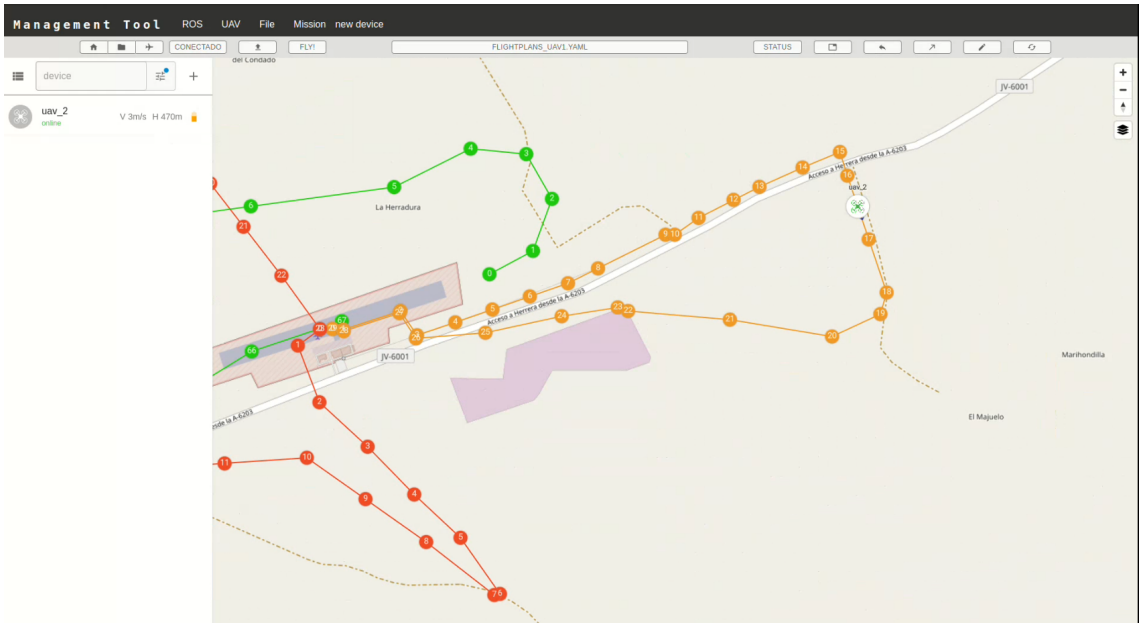

Figura 5.5 Prueba individual del DJI Matrice 210 en ATLAS .

La prueba de inspección multi-UAV de las lineas eléctrica se realizó con las 3 aeronaves de forma conjunta con una la ruta pre-establecida generada por un planificador de vuelo externo a la GCS. Al finalizar la misión, cada aeronave aterrizó de forma automática, realizando la misión completa en 15 minutos. Las rutas ejecutadas por los 3 UAVs se pueden observar en la Figura 5.6

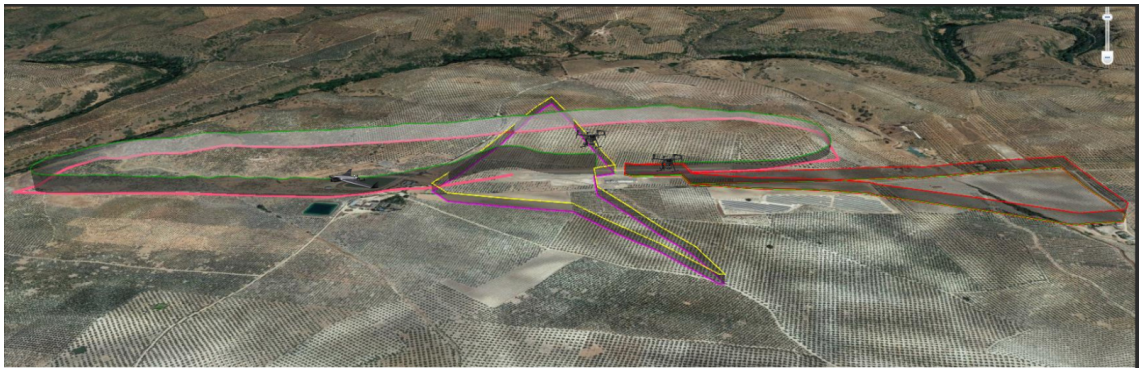

Figura 5.6 Rutas ejecutadas por los UAVs en la misión conjunta de inspección de lineas eléctricas. .

Las pruebas fueron satisfactorias ya que la flota de vehículos aéreos no tripulados llevó a cabo la inspección de varios kilómetros de líneas eléctricas de forma automática utilizándose la GCS para la monitorización y el control del vuelo de los 3 UAVs. En la Figura 5.7 el uav_1 es un DJI Matrice M210 RTK con aterrizaje por GNSS, el uav_2 es un DJI Matrice M210 RTK con aterrizaje por

visión, y el uav_3 es un DeltaQuad.

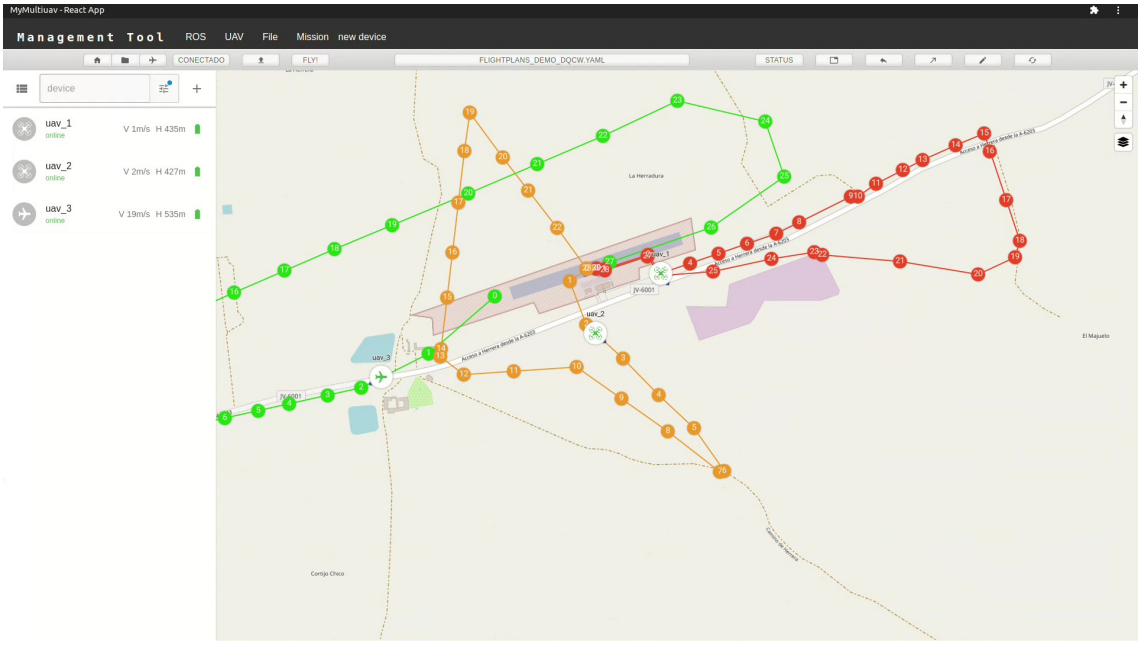

Figura 5.7 Pruebas de monitorización de la GCS para la inspección de lineas eléctricas .

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En conclusión, la herramienta de gestión para sistemas de inspección basados en múltiples drones desarrollada en este trabajo de TFM ha demostrado permitir monitorizar y gestionar múltiples UAVs a través de una interfaz intuitiva que permite visualizar en tiempo real la información de los drones.

El uso de tecnologías Web y el uso de una arquitectura cliente servidor ha permitido crear una interfaz multi usuario y multi ventana con capacidad de conexión con otras aplicaciones externas a través del uso de API y Websocket.

La herramienta desarrollada ha demostrado su gran utilidad en la implementación del sistema de inspección de lineas eléctricas basado en múltiples UAVs como parte de los proyectos AERIAL-CORE y RESISTO.

6.2 Trabajo Futuro

La herramienta y el sistema multi-UAV desarrollados cumplen con las funcionalidades deseadas, pero aun presentan mucho margen de mejora. Es necesario implementar nuevas funcionalidades que complementen las que se encuentran desarrolladas y ayuden a mejorar la robustez del sistema y la interacción con el usuario, por lo que se proponen las siguientes funcionalidades para desarrollo futuro:

- Supervisor en cada UAV que controle y levante los programas necesarios para la ejecución de las tareas de inspección.
- Creación y modificación de misiones en la GCS.
- Implementar algoritmos de planificación de misión en la GCS.
- Visualización de objetos de interés para el desarrollo de la misión en el mapa de la GCS, como las torres y líneas eléctricas para los proyectos AERIAL-CORE y RESISTO.
- Ventana de login y autenticación de usuario.
- Implementar la visualización de altura del terreno en la GCS.
- Implementación de página de alarmas para visualizar históricos de las alarmas de la misión.
- Implementación de alarmas georeferenciadas y con información adicional para una correcta interpretación actual del estado del dron.

- Visualización de múltiples cámaras al mismo tiempo.

Agradecimientos

Este trabajo ha sido financiado por el proyecto nacional RESISTO (2021/C005/00144188) de los fondos FEDER (Fondo Europeo de Desarrollo Regional) del Ministerio de Asuntos Económicos y Transformación Digital y por el proyecto europeo AERIAL-CORE (H2020-2019-871479).

Apéndice A
Documentación sobre API

A.1 API GCS (Ground control Station MultiUAV) v2.3.1

High level API description for Ros communication with UAV.Some usefull link: - Repositorio
Base URLs:

- https://localhost:4000/api/

Email: Support
Devices
Management of UAV-devices

A.1.1 get__devices

`GET /devices`
Fetch a list of Devices
Return the list of devices
Parameters

Name	In	Type	Required	Description
token	header	string	false	Token de autenticacion

Example responses

200 Response


```
[
  {
    "id": 0,
    "name": "string",
    "status": "string",
    "lastUpdate": "2019-08-24T14:15:22Z",
    "category": "string"
  }
]
```


Responses

Status	Meaning	Description	Schema
200	OK	(ok) list of devices	Inline
400	Bad Request	No permission	None

Response Schema
Status Code **200**

Name	Type	Required	Restrictions	Description
anonymous	[Device]	false	none	none
» id	integer	false	none	none
» name	string	false	none	none
» status	string	false	none	none
» lastUpdate	string(date-time)	false	none	in IS0 8601 format. eg. `1963-11-22T18:30:00Z`
» category	string	false	none	none

This operation does not require authentication

A.1.2 post__devices

`POST /devices`

Create a Device

Body parameter

```
{
  "uav_id": "string",
  "uav_type": "string"
}
```

Parameters

Name	In	Type	Required	Description
token	header	string	false	Token de autenticacion
body	body	BodyAddUAV	true	none

Example responses

200 Response

```
{
  "uav_id": "string",
  "uav_type": "string"
}
```

Responses

Status	Meaning	Description	Schema
200	OK	OK	BodyAddUAV

This operation does not require authentication

A.1.3 delete_*devices*{id}

`DELETE /devices/{id}`

Delete a Device

Parameters

Name	In	Type	Required	Description
id	path	integer	true	none

Example responses

Responses

Status	Meaning	Description	Schema
204	No Content	No Content	None

Response Schema

This operation does not require authentication

Positions

Obtains raw data from the UAV

A.1.4 get__postitions

`GET /postitions`

Fetches a list of Positions

We strongly recommend using Websocket instead of periodically polling positions endpoint. Without any params, it returns a list of last known positions for all the user's Devices. *from* and *to* fields are not required with *id*.

Example responses

200 Response


```
[
  {
    "deviceId": 0,
    "deviceTime": "2019-08-24T14:15:22Z",
    "latitude": 0,
    "longitude": 0,
    "altitude": 0,
    "speed": 0,
    "course": 0,
    "attributes": {
      "protocol": "string",
      "mission_state": "string",
      "wp_reached": 0,
      "uav_state": "string",
      "landed_state": "string",
      "alarm": "string"
    }
  }
]
```

Responses

Status	Meaning	Description	Schema
200	OK	OK	Inline

Response Schema
Status Code **200**

Name	Type	Required	Restrictions	Description
anonymous	[Position]	false	none	none
» deviceId	integer	false	none	none
» deviceTime	string(date-time)	false	none	in IS0 8601 format. eg. `1963-11-22T18:30:00Z`
» latitude	number	false	none	none
» longitude	number	false	none	none
» altitude	number	false	none	none
» speed	number	false	none	in knots
» course	number	false	none	none
» attributes	object	false	none	none
»» protocol	string	false	none	none
»» mission_state	string	false	none	none
»» wp_reached	integer	false	none	none
»» uav_state	string	false	none	none
»» landed_state	string	false	none	none
»» alarm	string	false	none	none

This operation does not require authentication
Missions
Management the mission

A.1.5 post_*loadmission*{id}

`POST /loadmission/{id}`
Load mission to devices
without any parameter,load the mission all UAV's
Parameters

Name	In	Type	Required	Description
id	path	integer	false	none

Example responses

Responses

Status	Meaning	Description	Schema
204	No Content	No Content	None

Response Schema
This operation does not require authentication

A.1.6 post_*commandmission*{id}

`POST /commandmission/{id}`

Command to exec mission

without any parameter,load the mission all UAV's

Parameters

Name	In	Type	Required	Description
id	path	integer	false	none

Example responses

Responses

Status	Meaning	Description	Schema
204	No Content	No Content	None

Response Schema

This operation does not require authentication

Commands

Send command the UAV

A.1.7 post__comands_send

`POST /comands/send`

Send command to devices

Send command to devices

Body parameter

```
{
  "deviceId": 0,
  "attributes": {}
}
```

Parameters

Name	In	Type	Required	Description
body	body	Command	true	none

Example responses

200 Response

```
{
  "deviceId": 0,
  "attributes": {}
}
```

Responses

Status	Meaning	Description	Schema
200	OK	Command sent	Command
400	Bad Request	Command no match	None

Response Schema
This operation does not require authentication

A.2 Schemas

A.2.1 BodyAddUAV

```
{
  "uav_id": "string",
  "uav_type": "string"
}
```

Properties

Name	Type	Required	Restrictions	Description
uav_id	string	false	none	none
uav_type	string	false	none	none

A.2.2 Device

```
{
  "id": 0,
  "name": "string",
  "status": "string",
  "lastUpdate": "2019-08-24T14:15:22Z",
  "category": "string"
}
```

Properties

Name	Type	Required	Restrictions	Description
id	integer	false	none	none
name	string	false	none	none
status	string	false	none	none
lastUpdate	string(date-time)	false	none	in IS0 8601 format. eg. `1963-11-22T18:30:00Z`
category	string	false	none	none

A.2.3 Position

```
{
  "deviceId": 0,
  "deviceTime": "2019-08-24T14:15:22Z",
  "latitude": 0,
  "longitude": 0,
```

```
  "altitude": 0,
  "speed": 0,
  "course": 0,
  "attributes": {
    "protocol": "string",
    "mission_state": "string",
    "wp_reached": 0,
    "uav_state": "string",
    "landed_state": "string",
    "alarm": "string"
  }
}
```


Properties

Name	Type	Required	Restrictions	Description
deviceId	integer	false	none	none
deviceTime	string(date-time)	false	none	in IS0 8601 format. eg. `1963-11-22T18:30:00Z`
latitude	number	false	none	none
longitude	number	false	none	none
altitude	number	false	none	none
speed	number	false	none	in knots
course	number	false	none	none
attributes	object	false	none	none
» protocol	string	false	none	none
» mission_state	string	false	none	none
» wp_reached	integer	false	none	none
» uav_state	string	false	none	none
» landed_state	string	false	none	none
» alarm	string	false	none	none

A.2.4 Command


```
{
  "deviceId": 0,
  "attributes": {}
}
```


Properties

Name	Type	Required	Restrictions	Description
deviceId	integer	false	none	none
attributes	object	false	none	none

A.2.5 Notificación


```
{
  "id": 0,
  "type": "string",
  "attributes": {}
}
```

Properties

Name	Type	Required	Restrictions	Description
id	integer	false	none	none
type	string	false	none	none
attributes	object	false	none	none

Apéndice B
Documentación sobre Web-Socket

B.1 GCS Websockets API for multiUAV 1.8.0 documentation

In addition to the REST API, the GCS has a WebSocket endpoint for live updates.

This only consists in the fact that when a connection is established, the server sends UPDATE messages to the client consecutively to update the list of devices and their telemetry.

B.1.1 Table of Contents

B.1.2 Servers

`public` **Server**

- URL: `localhost:4000/api/socket`
- Protocol: `wss`

Public server available without authentication

B.1.3 Operations

SUB / Operation

- Operation ID: `sendMessage`

Messages that you receive from the websocket

Message `update`

array with serverstate, device and positions

Element of the array contains a Json type object of devices, positions. If a message does not contain any object of a certain type, the key will not be included in the JSON structure. In most cases, a message contains only one type of objects.

Payload

Name	Type	Description	Value	Constraints	Notes
devices	array(object)	-	-	-	-
devices.id	integer	-	-	-	-
devices.name	string	-	-	-	-
devices.status	string	-	-	-	-
devices.lastUpdate	string	in IS0 8601 format. eg. `1963-11-22T18:30:00Z`	-	format (`date-time`)	-
devices.category	string	-	-	-	-
position	array(object)	-	-	-	-
position.deviceId	integer	-	-	-	-
position.deviceTime	string	in IS0 8601 format. eg. `1963-11-22T18:30:00Z`	-	format (`date-time`)	-
position.latitude	number	-	-	-	-
position.longitude	number	-	-	-	-
position.altitude	number	-	-	-	-
position.speed	number	in m/s	-	-	-
position.course	number	-	-	-	-
position.attributes	object	-	-	-	**additional properties are allowed**
position.attributes.-protocol	string	-	-	-	-
position.attributes.-mission_state	string	-	-	-	-
position.attributes.-wp_reached	integer	-	-	-	-
position.attributes.-uav_state	string	-	-	-	-
position.attributes.-landed_state	string	-	-	-	-
position.attributes.-alarm	string	-	-	-	-
RosState	boolean	-	-	-	-

Examples of payload *(generated)*


```
{
  "devices": [
    {
      "id": 0,
      "name": "string",
      "status": "string",
      "lastUpdate": "2019-08-24T14:15:22Z",
      "category": "string"
    }
  ],
  "position": [
    {
```

```
      "deviceId": 0,
      "deviceTime": "2019-08-24T14:15:22Z",
      "latitude": 0,
      "longitude": 0,
      "altitude": 0,
      "speed": 0,
      "course": 0,
      "attributes": {
        "protocol": "string",
        "mission_state": "string",
        "wp_reached": 0,
        "uav_state": "string",
        "landed_state": "string",
        "alarm": "string"
      }
    }
  ],
  "RosState": true
}
```

B.2 Schemas

B.2.1 Update

```
{
  "devices": array(Object),
  "position": array(Object),
  "RosState": "Bolean",
}
```

Properties

Name	Type	Required	Restrictions	Description
devices	array(Device)	false	none	List of added devices
position	array(Position)	false	none	list of telemetry of addad devices
RosState	bool	false	none	State of Ros conexion

B.2.2 Device

```
{
  "id": 0,
  "name": "string",
  "status": "string",
  "lastUpdate": "2019-08-24T14:15:22Z",
  "category": "string"
}
```

Properties

Name	Type	Required	Restrictions	Description
id	integer	false	none	none
name	string	false	none	none
status	string	false	none	none
lastUpdate	string(date-time)	false	none	in IS0 8601 format. eg. `1963-11-22T18:30:00Z`
category	string	false	none	none

B.2.3 Position

```
{
  "deviceId": 0,
  "deviceTime": "2019-08-24T14:15:22Z",
  "latitude": 0,
  "longitude": 0,
  "altitude": 0,
  "speed": 0,
  "course": 0,
  "attributes": {
    "protocol": "string",
    "mission_state": "string",
    "wp_reached": 0,
    "uav_state": "string",
    "landed_state": "string",
    "alarm": "string"
  }
}
```

Properties

Name	Type	Required	Restrictions	Description
deviceId	integer	false	none	none
deviceTime	string(date-time)	false	none	in IS0 8601 format. eg. `1963-11-22T18:30:00Z`
latitude	number	false	none	none
longitude	number	false	none	none
altitude	number	false	none	none
speed	number	false	none	in knots
course	number	false	none	none
attributes	object	false	none	none
» protocol	string	false	none	none
» mission_state	string	false	none	none
» wp_reached	integer	false	none	none
» uav_state	string	false	none	none
» landed_state	string	false	none	none
» alarm	string	false	none	none

Apéndice C
Configuración multimaster

Multimaste-Fkie es un paquete de ROS que permite conectar distintos ordenadores que se encuentran en la misma red, de forma que todos los nodos, topics y servicios se pueden comunicar como si se estuvieran ejecutando en el mismo ordenador.

Para la instalación del paquete se utilizó el siguiente comando:

Código C.1 Instalacion de multimaster en ROS melodic.

```
sudo apt-get install ros-melodic-multimaster-fkie
```

Para que este paquete funcione correctamente es necesario indicarle las IP de todas las máquinas a las que se busca establecer conexión. Esto tiene que realizarse en el archivo /etc/hosts donde se coloca el nombre del host y la dirección IP de la máquina como se muestra en el Código C.2. Además en los launch en los que se usa el paquete de ROS se tiene que colocar el nombre del host de la máquina a la que se va a conectar como se ve en el Código C.3 y C.4.

Código C.2 Intalacion de Multimaster.

```
sudo apt-get install ros-melodic-multimaster-fkie
```

Código C.3 Ejemlo de configuracion de archivo multimaster.launch.

```
<launch>
  <node name="master_discovery" pkg="master_discovery_fkie" type="
      master_discovery">
    <rosparam param="robot_hosts">[ubuntu18-Lenovo,ubuntu-black]</
        rosparam>
    <rosparam param="static_hosts">[ubuntu18-Lenovo,ubuntu-black]</
        rosparam>
    <rosparam param="send_mcast">False</rosparam>
    <rosparam param="listen_mcast">False</rosparam>
  </node>
  <node name="master_sync" pkg="master_sync_fkie" type="master_sync">
    <!--rosparam param="sync_hosts">[ubuntu18-Lenovo,ubuntu-black,arpapc
        ]</rosparam-->
  </node>
</launch>
```

Código C.4 Ejemlo de configuracion de archivo conect_uas.launch.

```
<launch>
    <node name="master_discovery" pkg="master_discovery_fkie" type="
        master_discovery">
        <rosparam param="robot_hosts">[ubuntu18-Lenovo, ubuntu-black,
            grvc-Xavier-NX,deltapi]</rosparam>
        <rosparam param="static_hosts">[ubuntu18-Lenovo, ubuntu-black,
            grvc-Xavier-NX,deltapi]</rosparam>
        <rosparam param="send_mcast">False</rosparam>
        <rosparam param="listen_mcast">False</rosparam>
    </node>
    <node name="master_sync" pkg="master_sync_fkie" type="master_sync">
        <!--rosparam param="sync_hosts">[ubuntu18-Lenovo, ubuntu-black,
            grvc-Xavier-NX]</rosparam-->
    </node>

    <include file="$(find rosbridge_server)/launch/rosbridge_websocket.
        launch"/>
</launch>
```

Código C.5 Configuracion de archivo /etc/hosts de GCS.

```
127.0.0.1      localhost
127.0.1.1      ubuntu18-Lenovo
# multimaster
10.42.0.42      ubuntu-black  #Matrice 210
10.42.0.41      deltapi       #DeltaQuad
10.42.0.99      grvc-Xavier-NX #Mtrice 210 aterrizaje autonomo
# The following lines are desirable for IPv6 capable hosts
::1    ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Apéndice D
Archivo de misión para las pruebas en ATLAS

El plan de vuelo de este archivo fue desarrollado mediante un software de planificación para la inspección de 3 lineas eléctricas, en la que participaron dos Matrice M210 y un DeltaQuad Pro.

D.1 Archivo de misión para las pruebas en ATLAS

Código D.1 Misión para las pruebas en ATLAS.

```
  frame_id: /gps

uav_n: 3

uav_1:          #M210_vision.   Base height: 445 [m].   Inspection
    height: 40 [m].
  wp_n: 30
  wp_0: [38.13921783,-3.17343016,40.37417603, 0]
  wp_1: [38.13920214,-3.17311366,40.78244019, 0]
  wp_2: [38.13963168,-3.17143225,43.22061157, 0]
  wp_3: [38.13907030,-3.17094522,41.67050171, 0]
  wp_4: [38.13936766,-3.16981891,42.97579956, 0]
  wp_5: [38.13966542,-3.16872551,45.44439697, 0]
  wp_6: [38.13996741,-3.16762333,47.45291138, 0]
  wp_7: [38.14027152,-3.16650115,49.60070801, 0]
  wp_8: [38.14061535,-3.16561737,48.66287231, 0]
  wp_9: [38.14139306,-3.16363102,43.91320801, 0]
  wp_10: [38.14139308,-3.16335690,43.60974121, 0]
  wp_11: [38.14178140,-3.16265602,41.66143799, 0]
  wp_12: [38.14219718,-3.16162751,44.28015137, 0]
  wp_13: [38.14249850,-3.16086486,46.79437256, 0]
  wp_14: [38.14294384,-3.15959487,49.76364136, 0]
  wp_15: [38.14330295,-3.15849550,53.81039429, 0]
  wp_16: [38.14275817,-3.15827878,52.79119873, 0]
  wp_17: [38.14128533,-3.15764764,53.66253662, 0]
```

```
  wp_18: [38.14005724,-3.15711986,56.79684448, 0]
  wp_19: [38.13955934,-3.15730315,57.27792358, 0]
  wp_20: [38.13904045,-3.15872780,54.36978149, 0]
  wp_21: [38.13943114,-3.16173435,49.15197754, 0]
  wp_22: [38.13962990,-3.16473112,46.70565796, 0]
  wp_23: [38.13971237,-3.16503641,47.04357910, 0]
  wp_24: [38.13951026,-3.16668523,48.33630371, 0]
  wp_25: [38.13912870,-3.16893216,44.52386475, 0]
  wp_26: [38.13899872,-3.17097718,41.54800415, 0]
  wp_27: [38.13958144,-3.17146385,43.18173218, 0]
  wp_28: [38.13916280,-3.17310175,40.78244019, 0]
  wp_29: [38.13921783,-3.17343016,40.37417603, 0]
  idle_vel: 5.00000000
  mode_landing: 5


uav_2:          #M210_GPS.    Base height: 444 [m].    Inspection height:
    30 [m].
  wp_n: 24
  wp_0: [38.13920996,-3.17380020,30.56164551, 0]
  wp_1: [38.13883219,-3.17445106,28.79190063, 0]
  wp_2: [38.13752204,-3.17381791,28.88183594, 0]
  wp_3: [38.13649470,-3.17239786,32.05810547, 0]
  wp_4: [38.13539110,-3.17102625,38.47814941, 0]
  wp_5: [38.13439197,-3.16966449,47.13433838, 0]
  wp_6: [38.13309497,-3.16850259,53.68536377, 0]
  wp_7: [38.13306973,-3.16866934,53.60992432, 0]
  wp_8: [38.13429298,-3.17067477,46.06329346, 0]
  wp_9: [38.13528475,-3.17246040,35.08819580, 0]
  wp_10: [38.13623962,-3.17418834,29.49960327, 0]
  wp_11: [38.13610862,-3.17660810,24.33303833, 0]
  wp_12: [38.13600996,-3.17853313,22.61614990, 0]
  wp_13: [38.13655765,-3.17944091,20.53286743, 0]
  wp_14: [38.13678012,-3.17941107,20.06158447, 0]
  wp_15: [38.13833814,-3.17917240,19.40795898, 0]
  wp_16: [38.13970221,-3.17897006,27.40673828, 0]
  wp_17: [38.14121218,-3.17873939,43.10690308, 0]
  wp_18: [38.14277047,-3.17850636,59.87918091, 0]
  wp_19: [38.14393246,-3.17829846,63.67245483, 0]
  wp_20: [38.14257739,-3.17703214,60.81005859, 0]
  wp_21: [38.14157520,-3.17605153,49.59866333, 0]
  wp_22: [38.14044692,-3.17493549,35.24044800, 0]
  wp_23: [38.13920996,-3.17380020,30.56164551, 0]
  idle_vel: 5.00000000
  mode_landing: 2


uav_3:          #DeltaQuad.    Base height: 446 [m].    Inspection height:
     60 [m].
  wp_n: 26
```

```
wp_0: [38.14044314,-3.16904476,65]
wp_1: [38.14200043,-3.16620024,71]
wp_2: [38.14366707,-3.16692363,86]
wp_3: [38.14429096,-3.17030084,95]
wp_4: [38.14241550,-3.17359703,89]
wp_5: [38.14004481,-3.17554871,60]
wp_6: [38.13836366,-3.17735995,50]
wp_7: [38.13665264,-3.17986797,47]
wp_8: [38.13560932,-3.18249816,38]
wp_9: [38.13538168,-3.18378303,36]
wp_10: [38.13512893,-3.18524567,32]
wp_11: [38.13491353,-3.18644925,32]
wp_12: [38.13470202,-3.18774142,30]
wp_13: [38.13448064,-3.18895443,24]
wp_14: [38.13428002,-3.19014457,21]
wp_15: [38.13403412,-3.19124836,20]
wp_16: [38.13366533,-3.19278480,20]
wp_17: [38.13340951,-3.19393168,19]
wp_18: [38.13492663,-3.19481948,16]
wp_19: [38.13643429,-3.19418826,18]
wp_20: [38.13729090,-3.19176827,33]
wp_21: [38.13624999,-3.18921138,26]
wp_22: [38.13628671,-3.18566458,31]
wp_23: [38.13686652,-3.18315326,50]
wp_24: [38.13786242,-3.17902237,40]
wp_25: [38.13892789,-3.17482060,25]
idle_vel: 15.000000
mode_landing: 2
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Andrey B, *Environment: signaling, stun and turn servers*, https://support.medialooks.com/hc/en-us/articles/360000213312-%D0%95nvironment-signaling-STUN-and-TURN-servers, Accessed: 2023-06-01.

[2] Hyeoncheol Baik and Jorge Valenzuela, *Unmanned aircraft system path planning for visually inspecting electric transmission towers*, Journal of Intelligent & Robotic Systems **95** (2019), no. 3, 1097–1111.

[3] Chuang Deng, Shengwei Wang, Zhi Huang, Zhongfu Tan, and Junyong Liu, *Unmanned aerial vehicles for power line inspection: A cooperative way in platforms and communications.*, J. Commun. **9** (2014), no. 9, 687–692.

[4] DJI, *Configuración de entorno de desarrollo*, https://developer.dji.com/onboard-sdk/documentation/development-workflow/environment-setup.html, Accessed: 2023-06-15.

[5] _____, *Dji onboard sdk*, https://github.com/dji-sdk/Onboard-SDK, Accessed: 2023-06-15.

[6] _____, *Guía de usuario del dji m210 v2 rtk*, https://dl.djicdn.com/downloads/m200_v2/20200608/M210_V2_M210_RTK_V2_Quick_Start_Guide_v1.4_multi-.pdf, Accessed: 2023-06-15.

[7] e distribución, *Proyecto resisto*, https://www.edistribucion.com/es/innovacion-nuevas-tecnologias/Proyecto_RESISTO.html, 2022, Accessed: 2023-06-15.

[8] Unión Europea, *Proyecto aerial-core*, https://aerial-core.eu/, 2019, Accessed: 2023-06-15.

[9] _____, *Proyecto omicron*, https://www.omicronproject.eu/, 2022, Accessed: 2023-06-15.

[10] OpenJS Fundation, *Nodejs*, https://openjsf.org/, Accessed: 2023-06-01.

[11] Tong He, Yihui Zeng, and Zhuangli Hu, *Research of multi-rotor uavs detailed autonomous inspection technology of transmission lines based on route planning*, IEEE Access **7** (2019), 114955–114965.

[12] Nicolai Iversen, Aljaž Kramberger, Oscar Bowen Schofield, and Emad Ebeid, *Pneumatic-mechanical systems in uavs: Autonomous power line sensor unit deployment*, 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 548–554.

[13] Nicolai Iversen, Oscar Bowen Schofield, Linda Cousin, Naeem Ayoub, Gerd Vom Bögel, and Emad Ebeid, *Design, integration and implementation of an intelligent and self-recharging drone system for autonomous power line inspection*, 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2021, pp. 4168–4175.

[14] Nathan Koenig and Andrew Howard, *Gazebo*, *https://gazebosim.org/home*, 2004-2011, Accessed: 2023-06-15.

[15] Jonathan Mace, *Rosbridge suit*, *http://wiki.ros.org/rosbridge_suite*, Accessed: 2023-06-15.

[16] I. Maza, F. Caballero, J. Capitan, J.R. Martinez de Dios, and A. Ollero, *A distributed architecture for a robotic platform with aerial sensor transportation and self-deployment capabilities*, Journal of Field Robotics **28** (2011), no. 3, 303–328.

[17] Ivan Maza, Anibal Ollero, Enrique Casado, and David Scarlatti, *Classification of multi-UAV architectures*, pp. 953–975, Springer Netherlands, 2015 (English).

[18] Meta, *React*, *https://react.dev/*, Accessed: 2023-06-15.

[19] Jose A. Millan-Romera, Hector Perez-Leon, Alejandro Castillejo-Calle, Ivan Maza, and Anibal Ollero, *Ros-magna, a ros-based framework for the definition and management of multi-uas cooperative missions*, 2019 International Conference on Unmanned Aircraft Systems (ICUAS), 2019, pp. 1477–1486.

[20] Anibal Ollero, Guillermo Heredia, Antonio Franchi, Gianluca Antonelli, Konstantin Kondak, Alberto Sanfeliu, Antidio Viguria, J. Ramiro Martinez-de Dios, Francesco Pierri, Juan Cortes, Angel Santamaria-Navarro, Miguel Angel Trujillo Soto, Ribin Balachandran, Juan Andrade-Cetto, and Angel Rodriguez, *The aeroarms project: Aerial robots with advanced manipulation capabilities for inspection and maintenance*, IEEE Robotics & Automation Magazine **25** (2018), no. 4, 12–23.

[21] robotwebtools, *robotwebtools*, *http://robotwebtools.org/*, Accessed: 2023-06-15.

[22] Alejandro Suarez, Antonio Enrique Jimenez-Cano, Victor Manuel Vega, Guillermo Heredia, Angel Rodriguez-Castaño, and Anibal Ollero, *Design of a lightweight dual arm system for aerial manipulation*, Mechatronics **50** (2018), 30–44.

[23] Docker Team, *Docker*, *https://www.docker.com/*, Accessed: 2023-07-01.

[24] GStreamer Team, *Gstreamer*, *https://gstreamer.freedesktop.org/*, Accessed: 2023-06-01.

[25] Vertical Technologies, *Documentación de la aeronave deltaquad pro*, *https://docs.deltaquad.com/deltaquad-operation-manual/*, Accessed: 2023-06-15.

[26] Justin Uberti Peter Thatcher, *Webrtc*, *https://webrtc.org/*, Accessed: 2023-06-01.

[27] Alexander Tiderko, *Paquete multimaster*, *http://wiki.ros.org/multimaster_fkie*, Accessed: 2023-06-15.

[28] Computer Vision and Geometry Lab of ETH Zurich, *Guía de simulación para el firmware px4*, *https://docs.px4.io/main/en/simulation/*, 2012, Accessed: 2023-06-15.