

UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA, CIENCIA Y COMPUTACIÓN (STEAM)

GRADO EN INGENIERÍA EN SISTEMAS INDUSTRIALES

TRABAJO FIN DE GRADO

Planificación de Misiones Autónomas para Drones mediante la Integración de PX4 y Gazebo

Alumno: Alex José Llivisupa Palacios

Director: Nourdine Aliane

JULIO 2026

TÍTULO: Planificación de Misiones Autónomas para Drones mediante la Integración de PX4 y Gazebo

AUTOR: Alex José Llivisupa Palacios

DIRECTOR DEL PROYECTO: Nourdine Aliane

FECHA: Julio de 2026

RESUMEN

En el presente trabajo se lleva a cabo un estudio detallado para el diseño e implementación de un sistema de navegación autónoma para vehículos aéreos no tripulados (UAV) en infraestructuras industriales y fotovoltaicas. La elección de esta temática responde a la necesidad de automatizar inspecciones y vigilancia, buscando minimizar riesgos laborales y superar las limitaciones de la seguridad tradicional. La robótica aérea autónoma se posiciona como una solución tecnológica clave para la modernización operativa en la Industria 4.0.

Para ello, se realizará un análisis teórico y práctico que abarcará la integración de arquitecturas de software abierto como ROS 2, MAVSDK y el autopiloto PX4. Además, se desarrollarán algoritmos de planificación dinámica de trayectorias, Gemelos Digitales, sistemas de restricción espacial (*Geofencing*) y un protocolo de contingencias basado en el semiverseno para garantizar aterrizajes seguros. Todo ello validado en un entorno de simulación *Software-in-the-Loop* (SITL) mediante Gazebo, con el objetivo de determinar la estrategia más robusta para su futura implementación física.

Palabras clave: UAV, Navegación autónoma, Simulación SITL, PX4, ROS 2, Gemelos digitales.

ABSTRACT

In the present work, a detailed study is carried out for the design and implementation of an autonomous navigation system for unmanned aerial vehicles (UAVs) in industrial and photovoltaic infrastructures. The choice of this topic responds to the need to automate inspections and surveillance, aiming to minimize occupational risks and overcome the limitations of traditional security. Autonomous aerial robotics is positioned as a key technological solution for operational modernization in Industry 4.0.

To this end, a theoretical and practical analysis will be conducted, covering the integration of open-source software architectures such as ROS 2, MAVSDK, and the PX4 autopilot. Additionally, dynamic trajectory planning algorithms, Digital Twins, spatial restriction systems (Geofencing), and a contingency protocol based on the Haversine formula will be developed to ensure safe landings. All of this will be validated in a Software-in-the-Loop (SITL) simulation environment using Gazebo, aiming to determine the most robust strategy for its future physical implementation.

Key words: UAV, Autonomous navigation, SITL simulation, PX4, ROS 2, Digital twins.

AGRADECIMIENTOS

En primer lugar, deseo expresar mi más sincero agradecimiento a mi director de tesis, Nourdine Aliane, por su valiosa guía, paciencia y acertados consejos durante la realización de este proyecto. Su apoyo y dedicación han sido fundamentales para superar los retos técnicos y llevar a buen término este trabajo.

Asimismo, extiendo mi gratitud a mis colegas de clase y compañeros de camino (Daniel Romero, Yeyson Hofmann y Martín Peevski). Las tardes en el club del Formula, las jornadas de estudio y el apoyo mutuo en estos años de carrera han hecho de esta etapa una experiencia amena e inolvidable.

Además, quiero mencionar de forma muy especial a Alejandra Muñoz, una amiga a la que conozco de toda la vida, ya que su confianza incondicional, su escucha, sus grandes consejos y su aliento constante han supuesto un impulso extra de motivación para cada paso de este proceso.

Por último, y no menos importante, a mi familia; sobre todo a mis padres y a mi hermana. Sobran las palabras porque vosotros ya lo sabéis todo, así que, simplemente gracias por todo.

Índice

RESUMEN 3
ABSTRACT 4
Capítulo 1. Introducción 10
1.1 Contexto 10
1.2 Objetivos Planteados 11
1.2.1 Objetivo General 11
1.2.2 Objetivos Específicos 12
1.3 Estructura de la memoria 13
Capítulo 2. Estado de arte y Antecedentes Tecnológicos 15
2.1 Aplicaciones de los drones 15
2.1.1 Logística y gestión de inventarios 16
2.1.2 Seguridad y Vigilancia Autónoma 16
2.1.3 Inspección y Mantenimiento Industrial 17
2.1.4 Agricultura y Medio Ambiente 17
2.1.5 Topografía y Modelado 3D 18
2.1.6 Legislación y Espacio Aéreo 19
2.2 Autopilotos de electrónica abierta 20
2.2.1 Ardupilot/APM 21
2.2.2 Pixhawk/PX4 22
2.2.3 LibrePilot 23
2.2.4 iNAV 24
2.2.5 Paparazzi 25
2.2.6 BetaFlight 26
2.3 Herramientas de desarrollo abiertas 28
2.3.1 PX4 Software 29
2.3.2 Ground Control Stations (GCS) 29
2.3.3 Protocolo MAVLink 30
2.3.4 Integración con ROS 31
2.3.5 MAVSDK 32

2.3.6 Entornos SITL/HITL 33
Capítulo 3. Desarrollo e Implementación 35
3.1 Metodologías y Planificación 35
3.2 Implementación 40
3.2.1 Configuración y despliegue del entorno SITL 40
3.2.2 Tratamiento de coordenadas 42
3.2.3 Planificación y automatización de las misiones 42
3.3 Integración 44
3.3.1 Arquitectura Desacoplada y Flujo de Datos 44
3.3.2 Máquina de Estados del Vuelo 45
3.3.3 Asincronía y Tolerancia a Fallos 45
3.3.4 Implementación de Sistema de Gestión de Emergencias y Puntos de Contingencia (Rally Points) 46
Capítulo 4. Pruebas y Evaluación 50
4.1 Definición de las Pruebas 50
4.2 Análisis e Interpretación de los Resultados 51
4.2.1 Pruebas preliminares 51
4.2.2 Offboard Mode 54
4.2.3 Mission Mode 57
4.3 Discusión y Limitaciones: Dificultades 60
4.4 Posibles Mejoras 61
4.4.1 Implementación de Evasión de Obstáculos Dinámicos: 61
Capítulo 5. Conclusiones 63
5.1 Resumen del trabajo 63
5.2 Conclusiones sobre los Objetivos del Proyecto 64
5.3 Líneas Futuras de Trabajo 64
ANEXOS 66
ANEXO I. Script “main.py” 66
ANEXO II Script “navegación.py” 70
ANEXO III. Diseño Gemelos Digitales en 3D 72
BIBLIOGRAFÍA 74

Índice de Figuras

Figura 1. UAV en operación. (Fuente [11]). 15
Figura 2. Cronograma de planificación. (Fuente Propia). 39
Figura 3. Comando Scripts Bash. (Fuente Propia). 41
Figura 4. Comando QGroundControl. (Fuente Propia). 41
Figura 5.Tratamiento de coordenadas en Python. (Fuente Propia). 42
Figura 6. QGroundControl y Gazebo en funcionamiento. (Fuente Propia). 52
Figura 7. Conexión con el dron en Gazebo y QGroundControl. (Fuente Propia). 53
Figura 8. Vuelo estacionario del dron. (Fuente Propia). 53
Figura 9. UAV realizando misión en Panales Fotovoltaicos. (Fuente Propia). 54
Figura 10. Visualización de la ruta en QGroundControl. (Fuente Propia). 55
Figura 11. UAV realizando la misión en la nave industrial. (Fuente Propia). 56
Figura 12. Visualización de la misión en QGroundControl. (Fuente Propia). 56
Figura 13. Script para ejecutar protocolo de emergencia. (Fuente Propia). 57
Figura 14. Misión creada en QGroundControl para paneles. (Fuente Propia). 58
Figura 15. Misión creada en QGroundControl para la nave industrial. (Fuente Propia). 59
Figura 16. Definición de Rally Points en el mapa de la nave industrial. (Fuente Propia). 60
Figura 17. Definición Geofence interior en nave industrial. (Fuente Propia). 60
Figura 18. Diseño del Gemelo Digital de la Nave Industrial. (Fuente Propia). 72
Figura 19. Plano Ortogonal de la planta fotovoltaica en Blender. (Fuente Propia). 72
Figura 20. Panel Solar. (Fuente [12]). 73

Índice de Tablas

Tabla 1. Tabla de Autopilotos de código abierto..... 28
Tabla 2. Tabla de misiones/pruebas realizadas. 51

Capítulo 1. Introducción

Este primer capítulo constituye el punto inicial del Trabajo de Fin de Grado, donde se expondrá la problemática actual que presentan diversas áreas en materia de seguridad y supervisión, cuya solución escogida se encuentra en la implementación de robótica autónoma aérea.

Así mismo, se hablarán de temas como los antecedentes históricos de los UAVs (Unmanned Aerial Vehicle), cuáles serán los objetivos planteados en este proyecto, y cómo se estructurará la presente memoria.

1.1 Contexto

El uso de vehículos aéreos no tripulados (Unmanned Aerial Vehicle UAV) ha experimentado un gran crecimiento en diferentes y muy diversas áreas, como lo son las aplicaciones industriales o logísticas, en el ámbito de la agricultura, en el ámbito de inspección de infraestructuras, etc.

Gracias al avance en miniaturizar componentes electrónicos, al gran desarrollo en la capacidad de cálculo de los microprocesadores y al desarrollo de arquitecturas de software de código abierto, los UAV han pasado de ser exclusivos únicamente en el ámbito militar a poder convertirse en herramientas fundamentales en la Industria 4.0. Es por esto por lo que son ideales en operaciones de inspección y monitorización de infraestructura y seguridad automatizada.

Dentro de este panorama, las grandes naves industriales y complejos logísticos modernos presentan un alto desafío operativo. Estas instalaciones funcionan con diferentes microclimas dinámicos, donde conviven sistemas de almacenamiento, vehículos de guiado automático (AGV) y un alto número de personal humano. En este contexto, la tarea de supervisión y seguridad son relegadas a personal humano (seguratas) y a sistemas estáticos de circuito cerrado de televisión (CCTV), donde ambos cuentan con limitaciones. Para el primero, mantener un grupo de trabajadores encargados al patrullaje suele ser muy costoso y pueden contar con una lenta respuesta ante una emergencia, mientras que, para el segundo, las cámaras presentan puntos ciegos.

La integración de drones autónomos es una solución tecnológica óptima para realizar un patrullaje continuo, capaz de proporcionar información en tiempo real de la situación en la que sin importar el terreno donde se encuentre realizando la inspección. Sin embargo, este método presenta una gran limitante, la sobre-especialización del sistema de vuelo, es decir, la dependencia de un mapa evidencia una grave falta de escalabilidad.

Es por eso por lo que en este trabajo se propone el diseño, implementación y validación de un sistema de control autónomo dinámico, versátil y adaptable a múltiples ámbitos operativos evaluado a través de una arquitectura SITL (Software-in-the-Loop). Se diseñarán diversos Gemelos Digitales (Digital Twins) mediante un software de modelado 3D, los cuáles serán digitalizados en Gazebo Sim. Y mediante la integración del firmware de control de vuelo PX4 y el uso de programación en Python, se logrará que el UAV consiga moverse a través de los Gemelos Digitales sin problema alguno.

Este enfoque destaca por el nulo uso de hardware en todo el proceso. Al desarrollar y validar toda la lógica de control en un entorno puramente de software, la posterior transición al mundo real será un proceso directo y muy eficiente. Este puente inmediato entre el software y el hardware reduce los costes de experimentación, acelera los tiempos de despliegue y elimina el riesgo de sufrir accidentes aparatosos y costoso en las fases tempranas de desarrollo.

1.2 Objetivos Planteados

Para seguir un orden establecido a la hora de la realización de este proyecto y asegurar la viabilidad técnica de la propuesta, se define una serie de objetivos estructurados con un único fin, cada uno con un hito específico.

1.2.1 Objetivo General

El objetivo general del proyecto será implementar un marco programático robusto para la planificación autónoma de misiones de vehículos aéreos no tripulados (UAV), integrando el sistema de autopiloto PX4, el simulador Gazebo, ROS2 como Sistema Operativo Robótico, y la implementación de bibliotecas de control MAVSDK en Python.

1.2.2 Objetivos Específicos

Tras plantear el objetivo principal del proyecto, se han propuesto una serie de hitos que se han de conseguir escaladamente para la realización y validación de este trabajo.

1. Instalación y configuración del entorno. Para este hito, al ordenador con S.O (Sistema Operativo) Windows se le instalará una máquina virtual, la cual se le asignará Ubuntu 24.04 como S.O. Tras arrancar la máquina virtual, se procederá con la instalación de ROS2, el sistema operativo robótico elegido para esta tarea.
2. Instalación y configuración de Gazebo junto con el autopiloto PX4. En este segundo hito, se procederá con la instalación y programación de Gazebo, un simulador robótico de código abierto. Tras ello, se instalará el autopiloto PX4 para la comunicación telemétrica entre el dron simulado y la estación de control MAVLink.
3. Despliegue y configuración del entorno de simulación (SITL): Integrar el firmware de control de vuelo PX4 con el motor de físicas 3D Gazebo Sim, estableciendo una arquitectura Software-in-the-Loop fiable que evite el uso de hardware físico en las etapas iniciales.
4. Desarrollo del orquestador de vuelo en Python: Se implementará un núcleo de control asíncrono utilizando la API de MAVSDK para gestionar la telemetría, el envío de comandos MAVLink y el control de la aeronave en modo Offboard y Mission.
5. Prueba con éxito de operaciones básicas de vuelo (despegue, vuelo estacionario, realizar un cuadrado, aterrizaje, etc.) usando scripts de Python.
6. Planificación dinámica de trayectorias: En este paso se crearán algoritmos capaces de transformar coordenadas métricas locales al sistema global WGS84, generando patrones de vuelo automatizados adaptables a distintos escenarios, como inspecciones en serpentina (plantas fotovoltaicas) o patrullaje perimetral (naves industriales).
7. Crear mapas 3D (Gemelos Digitales) personalizados, integrando obstáculos para generar entornos de prueba específicos.
8. Implementación de un sistema avanzado de contingencias. Se desarrollará una capa de seguridad operativa que incluya barreras virtuales espaciales (Geofencing) y un algoritmo matemático basado en la fórmula del semiverseno

para calcular y ejecutar aterrizajes de emergencia en zonas seguras (Rally Points) ante interrupciones de la misión.

9. Planificación de trayectorias en Modo *"Mission.* En la interfaz de QGroundControl se realizará pruebas de trayectoria para poder hacer una comparativa entre el modo "*Offboard*" y el modo "*Mission*".
10. Desarrollar y validar algoritmos de planificación de trayectorias y evasión de obstáculos en el entorno simulado seguro. Este último hito tratará de las pruebas que se le realicen al dron en los diferentes Gemelos Digitales creados, analizando la respuesta de la aeronave al completar una misión, o por lo contrario, su respuesta ante un fallo durante el vuelo.

1.3 Estructura de la memoria

El presente documento se estructurará de la siguiente manera, donde se tratará de seguir un orden metodológico desde los fundamentos teóricos hasta la validación de las pruebas a realizar.

- Capítulo 1. Introducción. Este es el capítulo actual, donde se ha presentado el contexto tecnológico y la necesidad de crear sistemas de navegación autónoma versátiles para infraestructuras complejas. Además, se definen los objetivos generales y específicos, y se detalla la organización del documento.
- Capítulo 2. Estado del arte y antecedentes tecnológicos. En este apartado se redacta el marco teórico del proyecto. En primera instancia se analizarán las aplicaciones actuales de los UAVs en la industria y la legislación vigente. Por otro lado, se realizará una comparativa de los principales autopilotos de código abierto, justificando la elección de herramientas empleadas.
- Capítulo 3. Desarrollo e implementación. Este capítulo es el núcleo del documento, ya que en este se detallará la metodología aplicada, la arquitectura del software desarrollado, el tratamiento matemático de las coordenadas, y el diseño del orquestador de vuelo.
- Capítulo 4. Pruebas y evaluación. El capítulo entero tratará de todas las pruebas realizadas al dron, como puede ser una primera toma de contacto, un recorrido sencillo, hasta llegar a recoger un espacio real en concreto.

- Capítulo 5. Conclusiones. En este último apartado de la memoria, se recogerán todos los resultados obtenidos a lo largo del desarrollo, evaluando si se ha cumplido todos los objetivos iniciales, proponiendo futuras líneas de trabajo, investigación, desarrollo y mejora, así como una posible transición a HITL (Hardware-in.the-Loop) y/o la integración de visión artificial.

Capítulo 2. Estado de arte y Antecedentes Tecnológicos

Este capítulo servirá para crear una base teórica para entender la tecnología que se implementa en este proyecto. Primero se hará un repaso de la evolución de los drones y qué aplicaciones reales tienen hoy en día en los diferentes sectores. A continuación, el enfoque cambiará al análisis de las diferentes arquitecturas de autopilotos de código abierto, a los protocolos de comunicación, al uso de librerías como MAVSDK para el control del dron mediante scripts programados en Python, y de las herramientas para la simulación de código abierto.

Este bloque es fundamental ya que hará que el documento no se limite en ser un manual de instrucciones sobre cómo configurar un entorno para simular, sino que proporciona una visión profunda de toda la tecnología que hay detrás que sostiene el sistema y el porqué de las herramientas elegidas.

2.1 Aplicaciones de los drones

En los últimos años, los Vehículos Aéreos No Tripulados (UAV) han conseguido consagrarse como un pilar fundamental en la Industria 4.0, debido a que han sufrido una gran metamorfosis, dejando atrás el límite de solo poder capturar imágenes para transformarse en plataformas de captura de datos de alta precisión y herramientas operativas dinámicas. Esta transformación se debe a la evolución e integración de la Inteligencia Artificial (IA) a bordo del dron, la mejora de las baterías ya que proporciona una gran autonomía al igual que una reducción de peso considerable, la miniaturización de los sensores (LiDAR, cámaras, GPS, de IMU, etc.), y al desarrollo de las arquitecturas de software autónomo.



Figura 1. UAV en operación. (Fuente [11]).

Los UAV han emergido como una tecnología verdaderamente transformadora, con aplicaciones que hoy en día abarcan una inmensa diversidad de campos. Actualmente, estas aeronaves operan no como simples plataformas de observación pasiva, sino como sistemas autónomos capaces de interactuar con entornos dinámicos y complejos. Para comprender el verdadero alcance de esta tecnología y la creciente necesidad de diseñar sistemas de navegación versátiles, a continuación, se clasificarán y se detallarán sus aplicaciones más destacadas en el estado del arte actual.

2.1.1 Logística y gestión de inventarios

En este sector, los UAV han sido adoptados como una herramienta para maximizar la eficiencia operativa, abarcando todos los procesos intralogístico. En el interior de las instalaciones, especialmente en zonas que carecen de señal satelital, los drones usan técnicas de Localización y Mapeo Simultáneo (SLAM) para poder navegar de forma autónoma por los pasillos de los grandes centros logísticos. Al estar equipados de gran tecnología, son capaces de realizar inventarios cíclicos masivos en altura a grandes velocidades inalcanzables para los operarios humanos, lo que elimina los riesgos de accidentes laborales y garantiza la sincronización del stock en tiempo real.

2.1.2 Seguridad y Vigilancia Autónoma

La incorporación de los UAV ha transformado los protocolos de seguridad física en entornos industriales y grandes infraestructuras, superando ampliamente las limitaciones de cobertura y movilidad propias de los sistemas tradicionales de CCTV fijo. En este ámbito destacan los sistemas autónomos conocidos *drone-in-a-box,* compuestos por estaciones base cerradas que resguarden, supervisen y recarguen el dron.

Un ejemplo de uso sería el caso de que se active una alerta perimetral, la estación se abre y el dron viaja hacia las coordenadas exactas del incidente, permitiendo evaluar la situación en cuestión de segundos. Para operar sin intervención humana, estas plataformas integran avanzados sistemas de analítica de vídeo basados en cámaras duales (sensores RGB y térmicos radiométricos), junto con redes neuronales convolucionales (CNN).

Gracias a esta capacidad de procesamiento a bordo, el dron es capaz de distinguir entre falsas alarmas y amenazas reales, realizar "*target tracking"* y transmitir vídeo cifrado en

tiempo real. Todo esto garantiza una respuesta inmediata y elimina por completo la exposición del personal de seguridad.

2.1.3 Inspección y Mantenimiento Industrial

Las tareas de supervisión estructural en grandes instalaciones industriales y de generación de energía han exigido tradicionalmente detener la producción, instalar complejos sistemas de andamios o exponer a los operarios a trabajos en altura con un elevado riesgo laboral. Hoy, los vehículos aéreos no tripulados han redefinido por completo estos procesos gracias a la aplicación de técnicas de inspección no destructiva.

Equipados con cargas útiles multimodales, los drones multirrotor pueden aproximarse con precisión milimétrica a infraestructuras críticas, desde palas de aerogeneradores y líneas de alta tensión hasta antorchas en plataformas petrolíferas. Todo ello se realiza sin comprometer la seguridad y sin necesidad de interrumpir la operación de la planta.

Durante estas misiones, la termografía permite detectar anomalías invisibles al ojo humano, como *hot spots* derivados de resistencias eléctricas defectuosas o microfisuras causadas por esfuerzos mecánicos. Más allá de la inspección inmediata, la recopilación masiva de datos y su posterior integración posibilitan la creación de Gemelos Digitales (*Digital Twins*). Estas réplicas tridimensionales de alta fidelidad sirven como base para modelos matemáticos que analizan la fatiga de materiales a lo largo del tiempo.

Gracias a ello, los equipos de ingeniería pueden evolucionar desde un mantenimiento reactivo hacia un enfoque predictivo, optimizando costes operativos y garantizando la integridad estructural de las instalaciones a largo plazo.

2.1.4 Agricultura y Medio Ambiente

La agricultura, la silvicultura y la gestión medioambiental han experimentado una modernización sustancial a partir de la incorporación de aeronaves no tripuladas, capaces de obtener datos telemétricos con una resolución espacial y temporal excepcional. Uno de los avances metodológicos más relevantes en este ámbito es la utilización de sensores multiespectrales e hiperespectrales embarcados, los cuales permiten registrar longitudes de onda que exceden ampliamente el espectro visible.

El procesamiento analítico de estas bandas espectrales posibilita la generación de cartografía agronómica de alta precisión, entre la que destacan los mapas basados en el

Índice de Vegetación de Diferencia Normalizada (NDVI). Dichos índices constituyen herramientas fundamentales para la detección temprana de estrés hídrico, deficiencias nutricionales no evidentes o la manifestación incipiente de plagas, fenómenos que suelen pasar desapercibidos mediante observación directa.

A partir de este diagnóstico detallado, se incorporan plataformas aéreas de mayor capacidad de carga equipadas con sistemas de pulverización inteligente. Estas aeronaves operan bajo el paradigma de la aplicación de tasa variable (VRA), mediante el cual la distribución de fertilizantes o productos fitosanitarios se realiza de forma altamente localizada, aplicando únicamente la dosis estrictamente necesaria en las coordenadas que lo requieren.

La integración entre la analítica aérea avanzada y la intervención robótica de precisión configura un modelo operativo que no solo incrementa la eficiencia productiva y reduce los costes asociados al manejo agronómico, sino que también minimiza de manera significativa la contaminación del suelo y de los recursos hídricos. En conjunto, este enfoque contribuye a consolidar sistemas de explotación agrícola más sostenibles y alineados con los principios de la gestión ambiental contemporánea.

2.1.5 Topografía y Modelado 3D

En las fases que conforman el ciclo de vida de la obra civil, la arquitectura y la actividad minera, los vehículos aéreos no tripulados han sustituido progresivamente a los métodos topográficos terrestres convencionales, produciendo una transformación profunda en la adquisición y el procesamiento de datos geoespaciales. Este cambio paradigmático se sustenta principalmente en el desarrollo de la fotogrametría digital y en la incorporación de sensores LiDAR (*Light Detection and Ranging*).

La fotogrametría permite generar modelos tridimensionales de alta fidelidad mediante el solapamiento secuencial de miles de imágenes aéreas de alta resolución. Por su parte, la tecnología LiDAR aporta una capacidad adicional de penetración a través de la cubierta vegetal denso, posibilitando la obtención de información precisa sobre la superficie terrestre subyacente. El procesamiento computacional de estos datos da lugar a nubes de puntos tridimensionales de elevada densidad y precisión centimétrica, a partir de las cuales se derivan Modelos Digitales (Digital Twins) de gran exactitud.

Estos productos geoespaciales resultan esenciales para la realización de cálculos volumétricos rigurosos en proyectos de movimiento de tierras y en explotaciones mineras a cielo abierto. Asimismo, en el ámbito de la construcción y bajo la metodología BIM (*Building Information Modeling*), los levantamientos aéreos permiten contrastar de forma continua el avance físico real de una obra respecto a su diseño estructural original. Esta capacidad de verificación sistemática optimiza la asignación de recursos, mejora la planificación operativa y facilita el diseño de nuevas infraestructuras y vías de comunicación con un nivel de detalle sin precedentes.

2.1.6 Legislación y Espacio Aéreo

La viabilidad comercial y la escalabilidad de las aplicaciones descritas dependen de manera directa de la existencia de un marco normativo robusto, coherente y armonizado a nivel europeo. En este sentido, la regulación de las aeronaves no tripuladas en la Unión Europea se articula principalmente a través del Reglamento (UE) 2018/1139, que establece las bases del sistema común de seguridad aérea, y de los reglamentos delegados y de ejecución que lo desarrollan: el Reglamento Delegado (UE) 2019/945, relativo a los requisitos de diseño y fabricación de los UAS y sus componentes, y el Reglamento de Ejecución (UE) 2019/947, que regula las normas y procedimientos aplicables a su operación. Estos textos constituyen el núcleo jurídico sobre el que la Agencia de Seguridad Aérea de la Unión Europea (EASA) define un enfoque regulatorio basado en el riesgo operacional.

En el ordenamiento español, este marco europeo se transpone y complementa mediante normativa estatal específica, destacando el Real Decreto 517/2024, que adapta el régimen jurídico nacional a los reglamentos europeos, regula las competencias de la Agencia Estatal de Seguridad Aérea (AESA) y establece disposiciones adicionales en materia de formación, certificación y operación de UAS en el espacio aéreo español.

Bajo este paradigma regulatorio, las operaciones con aeronaves no tripuladas se estructuran en tres categorías principales:

- **Categoría Abierta:** Comprende operaciones de bajo riesgo que no requieren autorización previa ni declaración operacional. El vuelo debe mantenerse dentro del alcance visual del piloto (VLOS), con límites estrictos de altitud (120 m AGL

según el Reglamento 2019/947) y prohibición de operar sobre concentraciones de personas. Las subcategorías A1, A2 y A3 establecen requisitos diferenciados en función del tipo de aeronave y la distancia respecto a terceros.

- **Categoría Específica**: Engloba operaciones de riesgo medio que superan las restricciones de la categoría abierta. Requiere una autorización operacional emitida por la autoridad competente (AESA en España) o una declaración operacional basada en escenarios estándar europeos (STS). La evaluación del riesgo se realiza mediante la metodología SORA (Specific Operations Risk Assessment), desarrollada por JARUS y adoptada por EASA como referencia metodológica. En esta categoría se incluyen los vuelos BVLOS, esenciales para aplicaciones industriales avanzadas y, en particular, para el sistema de navegación autónoma en infraestructuras propuesto en este proyecto.
- **Categoría Certificada**: Reservada para operaciones de alto riesgo y elevada complejidad, como los sistemas de movilidad aérea avanzada (VTOL), el transporte de mercancías peligrosas o los vuelos sobre grandes aglomeraciones de personas. Exige certificaciones de aeronavegabilidad, licencias de piloto y requisitos de seguridad equiparables a los de la aviación tripulada, conforme al Reglamento 2018/1139 y a los estándares de certificación de EASA.

Finalmente, para garantizar la integración segura y eficiente de un número creciente de aeronaves no tripuladas en el espacio aéreo compartido con la aviación comercial, la normativa europea establece la implantación progresiva del U-Space, definido en el Reglamento de Ejecución (UE) 2021/664, junto con sus reglamentos complementarios 2021/665 y 2021/666. Este ecosistema regulatorio introduce los servicios U-Space, como la identificación electrónica remota, la gestión dinámica del espacio aéreo y la resolución automatizada de conflictos, y sienta las bases para los sistemas de Gestión del Tráfico No Tripulado (UTM). Su despliegue constituye un requisito tecnológico esencial para habilitar la automatización industrial a gran escala y consolidar un entorno operativo seguro, eficiente y plenamente integrado en el sistema aeronáutico europeo.

2.2 Autopilotos de electrónica abierta

El avance de los sistemas autopilotos de código abierto ha desempeñado un papel decisivo en la evolución de la tecnología aplicada a los UAV. Se han impulsado numerosos proyectos, donde algunos ya se han quedado obsoletos con el paso de los años, como lo

pueden ser TauLabs, OpenPilot o MultiWii, mientras otros se han podido consolidar como los referentes del sector.

Destacan Ardupilot/APM, Pixhawk/PX4, LibrePilot, iNAV, Paparazzi y Betaflight. La naturaleza abierta de estos autopilotos les ha permitido incorporar mejoras continuas, optimizar el rendimiento de los controladores y adaptarlos a nuevas necesidades.

2.2.1 Ardupilot/APM

El desarrollo de Ardupilot constituye uno de los hitos más relevantes en la evolución de los sistemas de autopiloto de código abierto. Su nacimiento fue en 2007, de la mano de Jordi Muñoz y Chris Anderson, donde impulsaron el proyecto dentro de DIY Drones, aprovechando la filosofía modular del entorno Arduino. A partir de esta iniciativa, el sistema se ha ido ampliando hasta convertirse en la plataforma madura y adoptada en los UAV que es hoy en día.

En sus inicios, ArduPilot estaba limitado a los microcontroladores de 8 bits, hasta que con el pasar de los años y con su gran crecimiento y desarrollo, logró migrar hacia arquitecturas de 32 bits de mayor rendimiento (como por ejemplo el estándar Pixhawk y los STM32). Esta migración permitió incorporar avances como la capa de abstracción de hardware AP_HAL (mejorar estimación del estado) o el Filtro de Kalman Extendido (mejorar estabilidad del vehículo en cuestión).

En el ámbito académico y de investigación, Ardupilot ha adquirido una relevancia notable gracias a su integración con entornos de simulación SITL (*Software In The Loop*). Esta herramienta posibilita la validación exhaustiva de algoritmos de navegación, guiado y control en un entorno virtual reproducible, lo que reduce riesgos y costes antes de realizar pruebas en hardware real. Esta capacidad resulta especialmente valiosa en proyectos de ingeniería donde la experimentación directa puede implicar daños materiales o condiciones operativas complejas.

Actualmente, Ardupilot continúa evolucionando bajo el mantenimiento activo de su equipo central de desarrollo y se ha consolidado como una referencia en aplicaciones comerciales avanzadas, así como en proyectos de investigación punteros dentro del ámbito de los sistemas autónomos.

2.2.2 Pixhawk/PX4

El ecosistema PX4 representa uno de los hitos más importantes dentro de los sistemas de control de vuelo de código abierto. Sus orígenes datan de 2011, año en que Lorenz Meier, junto con su equipo de la ETH Zürich, puso en marcha una plataforma pensada inicialmente para impulsar la investigación en robótica avanzada y percepción artificial. Desde el principio, el proyecto se diseñó para explotar al máximo las arquitecturas de 32 bits, lo cual contribuyó también al nacimiento del estándar de hardware abierto Pixhawk. Gracias a la licencia BSD, su adopción en entornos industriales resultó mucho más sencilla, ya que permite integrar el núcleo del autopiloto en productos propietarios sin grandes restricciones legales. Hoy en día, el proyecto cuenta con el respaldo de la Dronecode Foundation y The Linux Foundation, factores que han reforzado su posición como referencia a nivel mundial.

Uno de los rasgos más característicos de PX4 es su diseño modular, sustentado en un sistema interno de comunicación llamado uORB, que funciona bajo un esquema de publicador-suscriptor. Este modelo separa la captura de datos de los algoritmos de control, lo que permite una respuesta del sistema rápida y resistente ante posibles fallos. Esta arquitectura facilita la adaptación a configuraciones experimentales complejas y el manejo de dinámicas de control demandantes, en particular en plataformas híbridas como los VTOL. Por otro lado, la estabilidad del vehículo se logra mediante algoritmos de estimación basados en Filtros de Kalman Extendido (EKF2 y EKF3), capaces de ofrecer una estimación precisa del estado del sistema incluso en condiciones operativas cambiantes.

En el ámbito académico, PX4 se ha posicionado como una herramienta clave para la investigación en autonomía y planificación de misiones. Su compatibilidad con motores de física como Gazebo hace posible crear entornos de simulación SITL de gran fidelidad, fundamentales para probar algoritmos de navegación, guiado y control sin necesidad de utilizar hardware real. A esto se suma el soporte nativo para el modo de vuelo Offboard, así como su conexión directa con ROS2 a través del protocolo MicroXRCE-DDS, lo que permite construir arquitecturas distribuidas donde tareas complejas de toma de decisiones, como evitar obstáculos o generar trayectorias en tiempo real, se transfieren a

computadoras de mayor capacidad a bordo. Esta combinación ha hecho del binomio Pixhawk/PX4 uno de los pilares tecnológicos más destacados de la robótica aérea actual.

2.2.3 LibrePilot

LibrePilot tomó forma en 2015 como continuación directa del proyecto OpenPilot, a raíz de una bifurcación motivada por una comunidad de desarrolladores que buscaba mantener un entorno de vuelo plenamente abierto y colaborativo. Su desarrollo estuvo muy ligado a controladoras icónicas de los primeros años del aeromodelismo moderno, entre ellas la CC3D y la familia Revolution, lo que le permitió partir de una base de código valorada por su claridad y robustez desde el punto de vista del diseño de software. A diferencia de otros autopilotos que recurren al protocolo MAVLink, LibrePilot basa su comunicación en UAVTalk, un protocolo de telemetría orientado a objetos reconocido por su eficiencia y por el orden con que organiza el intercambio de información entre el vehículo y la estación de control.

Internamente, el sistema gira en torno al concepto de UAVObjects, un modelo en el que cada parámetro, sensor o comando se representa como un objeto independiente. Esta filosofía modular simplifica la modificación de los controladores PID, la lectura de sensores y la adaptación del software a distintas configuraciones experimentales. Si bien LibrePilot ganó gran reconocimiento entre los drones acrobáticos y de carreras gracias a la suavidad de su estabilización manual, su alcance va mucho más allá del vuelo deportivo. El sistema incorpora algoritmos de navegación inercial y filtros matemáticos avanzados que, en combinación con receptores GPS y magnetómetros, hacen posible operar aeronaves de ala fija, helicópteros y multirrotores tanto en modos asistidos como en navegación por waypoints.

Desde el punto de vista académico, LibrePilot resulta especialmente interesante para el estudio del control electrónico de bajo nivel y de los sistemas de telecomunicación embarcados. Aunque proyectos como PX4 o Ardupilot lideran actualmente las aplicaciones autónomas más complejas, LibrePilot conserva un notable valor formativo gracias a su estación de control terrestre, LibrePilot GCS, que se distingue por una interfaz clara, visual y altamente configurable. Esta facilidad de uso, sumada a la transparencia de su código fuente, lo convierte en un entorno ideal para el análisis experimental de sistemas

de Guiado, Navegación y Control (GNC), así como para ajustar en tiempo real los algoritmos de estabilización mediante herramientas como TxPID. En definitiva, LibrePilot funciona como un puente entre el control manual avanzado y los sistemas autónomos, brindando un marco estructurado para la experimentación y la investigación en robótica aérea.

2.2.4 iNAV

El firmware INAV se ha posicionado como una de las propuestas más representativas dentro del campo de los controladores de vuelo de código abierto, particularmente en aplicaciones donde las limitaciones de tamaño, peso y consumo energético resultan determinantes para el diseño del sistema. Surgido a partir del proyecto Cleanflight, INAV se desarrolló con el objetivo de dotar a microcontroladores económicos de la familia STM32 de funciones avanzadas de navegación autónoma, conjugando la baja latencia propia del vuelo acrobático con la precisión necesaria en operaciones guiadas por GPS.

A diferencia de otros autopilotos que requieren sistemas operativos embebidos o arquitecturas de comunicación más complejas, INAV apuesta por la eficiencia computacional. Su arquitectura está diseñada para procesar de forma estable las señales de receptores GNSS, barómetros, magnetómetros y sensores de flujo óptico a través de interfaces estándar como I2C, SPI o UART, optimizando así el uso de la CPU y la memoria disponible. Esta ligereza operativa asegura que funciones críticas para la seguridad del vuelo permanezcan siempre disponibles, lo cual resulta esencial en misiones de largo alcance.

Uno de los rasgos más destacados del sistema es su comportamiento en aeronaves de ala fija, en especial en configuraciones aerodinámicamente inestables. Mediante un sistema de mezcla de actuadores flexible y la adaptación dinámica de los controladores PID, INAV consigue compensar de manera efectiva las perturbaciones atmosféricas. A esto se suma la incorporación nativa de un sistema OSD (On-Screen Display), que superpone información de navegación y telemetría directamente sobre el vídeo, permitiendo monitorizar en tiempo real el estado del filtro inercial.

En cuanto a la planificación y ejecución de misiones automatizadas, el ecosistema cuenta con INAV Configurator, una herramienta que funciona como estación de control y

entorno de configuración. Desde esta interfaz es posible diseñar rutas basadas en waypoints y ajustar los parámetros de guiado y control con un enfoque pensado para la experimentación. En conjunto, INAV se consolida como una infraestructura especialmente apta para el estudio y validación de técnicas de navegación y control en vehículos no tripulados ligeros, mostrando que es posible implementar autonomía avanzada de forma eficiente incluso en hardware con recursos limitados.

2.2.5 Paparazzi

El sistema Paparazzi, creado a principios de la década de 2000 en la Escuela Nacional de Aviación Civil de Francia (ENAC), representa uno de los proyectos de vuelo autónomo de código abierto más sólidos y longevos dentro del sector aeroespacial. A diferencia de otras plataformas surgidas del aeromodelismo o de iniciativas con orientación comercial, Paparazzi se concibió desde sus inicios como una infraestructura pensada para la investigación científica, dando prioridad a la fiabilidad, la seguridad y el rigor matemático del software por encima de la facilidad de uso.

Una de las particularidades más notables del proyecto es su enfoque centrado en la generación de código precompilado, en contraposición a la interpretación dinámica habitual en otros controladores de vuelo. La totalidad de la configuración del sistema, desde la definición del modelo de aeronave hasta la disposición de sensores, los parámetros de comunicación y la estructura de telemetría, se especifica mediante archivos XML. En la fase de compilación, dichas descripciones se traducen en código C optimizado y adaptado a cada plataforma específica. Este procedimiento reduce considerablemente el consumo de memoria, evita sobrecargas innecesarias y facilita la verificación formal del software, un elemento clave en sistemas críticos de navegación aérea.

El modelo de autonomía de Paparazzi también se distancia del enfoque convencional basado en secuencias de waypoints. En cambio, recurre a máquinas de estados finitos, lo que permite representar las misiones como conjuntos de comportamientos y transiciones lógicas. Esta estructura posibilita que la aeronave ejecute patrones de vuelo complejos, responda de forma determinista ante situaciones imprevistas y evalúe condiciones en bucle cerrado, sin depender de un planificador externo.

Respecto a la arquitectura de comunicaciones, Paparazzi se apoya en el bus de software Ivy, pensado para sistemas distribuidos asíncronos. Este planteamiento permite que múltiples agentes intercambien información de manera simultánea, lo que abre la puerta a capacidades avanzadas de coordinación multiagente y robótica de enjambres. La plataforma puede gestionar de forma descentralizada la interacción entre varias aeronaves dentro de un mismo espacio aéreo, convirtiéndose así en una herramienta de gran valor para la experimentación con sistemas cooperativos complejos.

2.2.6 BetaFlight

El firmware Betaflight se ha consolidado como una de las plataformas más influyentes en el ámbito del vuelo acrobático y de competición, adoptando un enfoque radicalmente diferente al de los sistemas centrados en la navegación autónoma. Surgió en 2015 como una evolución del código de Cleanflight, su desarrollo se ha orientado a maximizar el control de bajo nivel, dando prioridad a la reducción extrema de la latencia y a una respuesta inmediata del vehículo. Esta filosofía responde a las demandas del vuelo de alto rendimiento, donde la precisión y la capacidad de reacción deben mantenerse en escalas temporales del orden de microsegundos.

El rendimiento distintivo de Betaflight se apoya en un procesamiento avanzado de señales digitales y en la ejecución de lazos PID a frecuencias muy altas, llegando a tasas de actualización de hasta 8 kHz. Puesto que los multirrotores acrobáticos operan en entornos sometidos a vibraciones intensas, el sistema cuenta con un conjunto de filtros especializados orientados a reducir el ruido mecánico sin generar retrasos perjudiciales. Entre estas soluciones destacan los filtros Notch dinámicos, capaces de analizar el espectro de vibraciones en tiempo real y atenuar de forma selectiva las frecuencias vinculadas a resonancias estructurales o al funcionamiento de los motores.

En lo relativo a la interacción con los actuadores, Betaflight ha desempeñado un papel central en la transición hacia protocolos digitales de alta velocidad. Su temprana adopción del estándar DSHOT representó un punto de inflexión en la comunicación entre el controlador de vuelo y los ESC. Las versiones más recientes del firmware aprovechan la telemetría bidireccional de DSHOT para obtener lecturas precisas de las RPM de cada motor, datos que se emplean para ajustar dinámicamente los parámetros de filtrado

durante el vuelo. Este mecanismo genera un bucle cerrado de compensación del ruido electromecánico, lo que mejora la estabilidad y la capacidad de respuesta del sistema.

Desde el punto de vista académico, Betaflight resulta un entorno muy adecuado para el estudio de técnicas avanzadas de control y procesamiento de señales embebidas. Aunque carece de herramientas de planificación global como las que ofrecen PX4 o Ardupilot, proporciona un marco experimental valioso para analizar dinámicas no lineales, evaluar estrategias de estabilización agresiva y estudiar el comportamiento aerodinámico de aeronaves con relaciones empuje-peso extremas. Asimismo, su sistema de registro de datos (Blackbox logging) permite realizar análisis detallados tanto en el dominio del tiempo como en el de la frecuencia, lo que lo convierte en una herramienta de gran utilidad para la investigación en control de vuelo de alta precisión.

A continuación, se recogerá toda la información relevante de cada uno de los autopilotos en una tabla comparativa:

Autopiloto	Aplicación	Ventajas
Ardupilot	Industria, Investigación y versatilidad general.	Gran comunidad que lo avala, excelente madurez comercial
PX4	Investigación avanzada, autonomía y robótica distribuida.	Arquitectura modular, integración nativa con ROS2 y MAVSDK
LibrePilot	Transición entre control manual y asistido.	Protocolo UAVTalk, diseño basado en objetos, altamente configurable
Paparazzi	Investigación científica, sistemas multiagente y enjambres.	Generación de código precompilado seguro, máquinas de estado finito.
Inav	Sistema de recursos limitados, vuelos de largo alcance y alas fijas.	Eficiencia computacional en placas STM32, OSD nativo integrado.

BetaFlight	Vuelo acrobático, carreras, (FPV) y estabilización agresiva.	Procesamiento a muy alta frecuencia (8 kHz), filtros dinámicos Notch.

Tabla 1. Tabla de Autopilotos de código abierto.

Tras analizar las distintas alternativas, el ecosistema PX4 se perfila como la opción idónea para este proyecto por tres ventajas competitivas fundamentales.

En primer lugar, su arquitectura interna modular basada en mensajería uORB garantiza una alta tolerancia a fallos, y, en segundo lugar, puede llegar a ofrecer una integración nativa y transparente con entornos de simulación de alta fidelidad como Gazebo (SITL), permitiendo un desarrollo seguro. Así mismo, su robusta compatibilidad con MAVSDK y ROS 2 mediante el estándar MicroXRCE-DDS facilita enormemente la programación de misiones en modo Offboard, delegando el control de alto nivel a scripts en Python con una latencia mínima, requisito indispensable para el diseño de nuestro orquestador autónomo.

2.3 Herramientas de desarrollo abiertas

Después de revisar a fondo las arquitecturas de autopilotos de código abierto, toca definir el ecosistema de software a usar para desarrollar, probar y validar este proyecto. No estamos ante un trabajo trivial, ya que cualquier error en el modelado matemático o un fallo en la lógica de control puede traducirse en la pérdida o colisión del dron real, con todo lo que eso implica.

Para evitar ese riesgo de falla en un dron real, este proyecto se apoya en un simulador de alta fidelidad, combinando interfaces de programación que permiten abstraerse del hardware real.

2.3.1 PX4 Software

PX4 es el sistema de control de vuelo de código abierto que funciona sobre el hardware Pixhawk. Está escrito en C/C++ y se divide en dos partes fundamentales, la pila de control de vuelo y el middleware, que se encarga de las comunicaciones internas.

La pila de control de vuelo es el núcleo de PX4, ya que aquí se encuentran los algoritmos de navegación y control que se usan en distintos tipos de aeronaves, ya sean multirrotor, aviones de ala fija o plataformas VTOL. El sistema soporta varios modos de vuelo, como manual, altitude *"hold"* (mantenimiento de altitud) y position *"hold"* (mantenimiento de posición). Dicho de forma sencilla, la pila se encarga de controlar la posición, la velocidad y la actitud del vehículo (yaw, pitch, roll) a través de controladores PID independientes.

Por otro lado, PX4 usa un Filtro de Kalman Extendido (EKF) para fusionar los datos que llegan de la IMU, el GPS, la brújula, los sensores de velocidad del aire y el barómetro, todo esto con el objetivo de estimar el estado del vehículo en cada momento. El EKF cumple dos funciones clave, la primera es estimar variables que no se pueden medir directamente, como la posición y la velocidad; y lo segundo, es afinar la orientación angular reduciendo el ruido que meten los sensores.

2.3.2 Ground Control Stations (GCS)

Una Estación de Control Terrestre (GCS, Ground Control Station) es básicamente la interfaz que conecta al usuario con el UAV. A nivel físico, suele ser un ordenador o un dispositivo móvil que, a través de un enlace serie inalámbrico (o de una conexión de red, en el caso de entornos simulados), permite supervisar y controlar el sistema en ambas direcciones.

Desde esta plataforma, el operador puede diseñar y ejecutar misiones autónomas más o menos complejas: definir waypoints, establecer geofencing, configurar los failsafes y mandar comandos de vuelo directos cuando haga falta. A la vez, la GCS va recibiendo y procesando en tiempo real la telemetría del vehículo, mostrando datos clave como la altitud, la actitud, el estado de la batería, la calidad de la señal GNSS y los mensajes internos del sistema.

Toda esta comunicación bidireccional se lleva a cabo mediante MAVLink, el protocolo estándar que se usa en prácticamente toda la industria. En este proyecto, dentro del

ecosistema PX4, la estación de control utilizada es QGroundControl, que resulta especialmente útil por ser multiplataforma y por su compatibilidad nativa con simulaciones Software-in-the-Loop (SITL).

En este entorno de pruebas virtual, QGroundControl se conecta con la instancia simulada del autopiloto a través de enlaces UDP locales, lo que permite observar el comportamiento de los algoritmos de vuelo casi como si el dron estuviera volando de verdad. Y, gracias a que MAVLink está estandarizado, todo esto mantiene compatibilidad con otras soluciones muy usadas en el sector, como Mission Planner o MAVProxy, lo que da bastante flexibilidad a la hora de trabajar.

2.3.3 Protocolo MAVLink

La comunicación entre los distintos subsistemas de un dron necesita un canal que sea rápido, seguro y eficiente, casi sin margen de error. Aquí es donde entra MAVLink (Micro Air Vehicle Link), que se ha convertido en el protocolo de mensajería estándar dentro de la robótica aérea. Lo desarrolló Lorenz Meier en la ETH Zürich, y desde entonces ha evolucionado hasta llegar a su versión actual, MAVLink 2.0, lanzada en 2017, que trajo mejoras importantes en seguridad, extensibilidad y flexibilidad de red.

Técnicamente hablando, la eficiencia de MAVLink viene de su formato de paquetes binarios, muy compactos. La información se serializa antes de enviarse y se deserializa justo al recibirla, lo que reduce bastante la carga computacional y aprovecha mejor el ancho de banda disponible. Esto es clave en situaciones donde el espectro de radio está limitado o la señal no es la mejor. Los paquetes pueden variar entre 12 y 280 bytes en el caso de mensajes firmados. Para asegurar que los datos llegan bien, cada paquete lleva un *"checksum"* de 2 bytes y, opcionalmente, una firma criptográfica de 13 bytes que sirve para autenticar la comunicación y evitar intrusiones o manipulaciones.

Esta arquitectura permite una comunicación bidireccional y asíncrona bastante fiable. Por un lado, el vehículo manda constantemente telemetría a la Estación de Control Terrestre, informando de su estado, posición, velocidad y altitud. Y al mismo tiempo, el autopiloto va recibiendo comandos importantes, como órdenes de despegue, aterrizaje o la carga de misiones autónomas. Además, MAVLink no depende de un medio físico concreto: puede funcionar sobre radio sub-GHz (433, 868 o 915 MHz), telemetría serie, Wi-Fi, o incluso,

en controladoras más modernas como las basadas en Pixhawk 5X, mediante Ethernet usando TCP/IP o UDP.

Una de las cosas más interesantes del protocolo, sobre todo en sistemas distribuidos, es su sistema de enrutamiento interno. MAVLink no se queda en una simple conexión punto a punto, sino que permite montar topologías de red completas, donde cada dispositivo (autopiloto, cámaras, gimbals, antenas de seguimiento, Companion Computers...) tiene su propio identificador único (System ID y Component ID). Esto hace posible dirigir los mensajes solo al componente que corresponde, evitando saturar el canal y haciendo que todo el sistema funcione de forma más eficiente.

2.3.4 Integración con ROS

El ecosistema Pixhawk/PX4 saca todo su potencial cuando se combina con ROS (Robot Operating System), o más concretamente con su versión más reciente, ROS 2. Se ha convertido en el estándar de facto para programar robots, y ofrece un conjunto enorme de herramientas que facilitan el desarrollo, la simulación (SITL) y el despliegue de drones de forma rápida. Gracias a lo modular que es su arquitectura y al respaldo que tiene de la comunidad, ROS 2 es prácticamente la base ideal para construir sistemas UAV realmente autónomos.

Antiguamente, la integración de alto nivel se hacía con la primera versión de ROS junto con MAVROS, un paquete puente que básicamente traducía los mensajes de MAVLink a tópicos y servicios que ROS pudiera entender. Sin embargo, hoy en día el camino lógico (y el estándar para cualquier desarrollo nuevo) pasa por migrar a ROS 2. Y no es un cambio solo de nombre, supone un salto importante, ya que introduce una arquitectura que sí puede operar en tiempo real, mejora los protocolos de seguridad criptográfica y, sobre todo, implementa políticas de Calidad de Servicio (QoS) mucho más fiables. Estas políticas QoS son especialmente útiles en redes con varios drones (multi-vehicle Systems), ya que permiten priorizar el ancho de banda para que, por ejemplo, un comando crítico llegue antes que un simple paquete de telemetría que no es tan urgente.

A nivel de comunicaciones, ROS 2 elimina la latencia de tener que traducir todo por en medio, ya que se conecta de forma nativa con el firmware de PX4 a través de Micro XRCE-DDS. Este puente permite que ROS 2 se comunique directamente con uORB, que es el bus interno de mensajes del propio autopiloto. Así, el ordenador auxiliar puede leer

datos de los sensores o mandar comandos a los actuadores prácticamente con la misma rapidez y privilegios que el núcleo del autopiloto.

Pero más allá de gestionar la mensajería básica o integrar sensores (cámaras, IMUs, GPS, LiDAR...), lo que realmente hace potente a ROS 2 en este proyecto es su enorme repositorio de paquetes especializados. Ahí entran algoritmos de estimación de estado, el uso de Filtros de Kalman Extendidos (EKF) para fusionar sensores, y técnicas de SLAM que permiten navegar de forma autónoma en interiores o en zonas sin cobertura GPS. A esto se suman librerías de percepción como PCL (Point Cloud Library), muy útiles para procesar datos espaciales de cámaras de profundidad. Y para las fases de prueba y validación, ROS 2 ofrece herramientas de diagnóstico realmente útiles: RViz, que permite visualizar en tiempo real el modelo 3D del dron y todo lo que están detectando sus sensores, y Rosbag, que graba todo el tráfico de datos del vuelo para poder reproducirlo después, lo cual ayuda muchísimo a la hora de depurar errores en los algoritmos de navegación.

2.3.5 MAVSDK

Para que la plataforma sea realmente autónoma, hace falta conectar de alguna forma las rutinas de estabilización de bajo nivel del autopiloto con la lógica de decisión de alto nivel. Y para esto, el ecosistema PX4 se apoya en MAVSDK (Micro Air Vehicle Software Development Kit). Lanzada en 2018, esta API se ha convertido en la herramienta oficial, más actualizada y recomendada para gestionar misiones y controlar vehículos aéreos basados en MAVLink. A diferencia de otras librerías que ya están un poco abandonadas, MAVSDK sigue en desarrollo activo, lo cual es una garantía de que es una solución sólida, escalable y que va a seguir funcionando a futuro.

A nivel de arquitectura, el núcleo de MAVSDK está escrito en C++, lo que asegura buen rendimiento y poca latencia en las comunicaciones. Pero lo más interesante es lo flexible que es, ya que ofrece "*wrappers*" para varios lenguajes de alto nivel, como Python, Swift, Rust o Java. Hay un detalle técnico que merece la pena mencionar: mientras que la librería nativa en C++ permite conectarse a la vez con varios UAVs, las versiones en lenguajes como Python están pensadas, por diseño, para controlar un solo vehículo por instancia, lo cual simplifica bastante el desarrollo cuando solo se necesita controlar un dron.

El punto clave de MAVSDK es su capacidad para operar en modo Offboard. En este modo, el control de la aeronave pasa de la emisora del piloto o de la GCS a un script que se ejecuta de forma autónoma, normalmente desde un Companion Computer. Gracias al modo Offboard, el código puede enviar comandos MAVLink directamente al autopiloto PX4, marcando "*waypoints*" de posición, velocidad o actitud en tiempo real.

Hoy en día, MAVSDK se usa bastante para ejecutar operaciones complejas, donde varios estudios demuestran de su utilidad en tareas exigentes, como el seguimiento de objetivos en movimiento, la evasión autónoma de obstáculos o misiones de vigilancia con actualización constante de waypoints. Además, su baja latencia es clave para algoritmos de aterrizaje de precisión en entornos dinámicos, donde el Companion Computer va ajustando la trayectoria de descenso según lo que detecta un sistema de visión artificial.

2.3.6 Entornos SITL/HITL

Dentro del ecosistema de herramientas para robótica aérea, poder probar y validar algoritmos antes de volar con el dron de verdad es prácticamente un requisito innegociable. El ciclo de vida del software robótico sigue el modelo en "V" de la ingeniería de sistemas, donde la simulación actúa como un mecanismo de seguridad que reduce riesgos y evita accidentes con el hardware. En este contexto, la industria ha estandarizado dos arquitecturas de simulación principales, SITL y HITL, que permiten reproducir el comportamiento del dron en condiciones controladas y con distintos niveles de fidelidad.

En el caso de SITL, todo el sistema corre de forma virtual en un mismo entorno computacional: el firmware del controlador de vuelo funciona como un proceso independiente que interactúa en tiempo real con un simulador físico y gráfico, que le va inyectando datos cinemáticos y ambientales por la red local mientras el controlador responde con comandos de navegación. Esta arquitectura es muy flexible, ya que permite acelerar la simulación, meter al dron en condiciones meteorológicas extremas o añadir ruido sintético a los sensores. Es el entorno ideal para depurar la lógica de alto nivel en Python y validar mecanismos de seguridad como el "geofencing", sin arriesgar ningún componente físico.

HITL, en cambio, representa un nivel de fidelidad superior y actúa como paso intermedio entre la simulación pura y el vuelo real. Aquí el simulador sigue generando el mundo

virtual y calculando las dinámicas de vuelo, pero el firmware corre directamente sobre el hardware físico del autopiloto, que procesa los datos del simulador como si vinieran de sus propios sensores y genera señales eléctricas reales que vuelven al entorno virtual. Esto permite comprobar que el microcontrolador tiene suficiente capacidad de cómputo, que no hay latencias críticas en la comunicación y que el sistema embebido se comporta de forma estable antes de dar el salto a un dron real.

Combinar estas dos metodologías de forma progresiva reduce bastante los costes y riesgos del desarrollo. Y al apoyarse en herramientas como MAVSDK y PX4, se tiene un marco estandarizado que facilita una evaluación iterativa, segura y escalable del comportamiento de sistemas autónomos en robótica aérea.

Capítulo 3. Desarrollo e Implementación

Una vez tenemos claro el marco teórico y las herramientas de código abierto que se usan en el estado del arte de la robótica aérea, este capítulo entra ya en la parte aplicada y central del TFG. La idea de esta sección es explicar, de forma ordenada y lo más detallada posible, todo el proceso de diseño, configuración y programación del sistema de navegación autónoma que se ha desarrollado.

Diseñar sistemas ciberfísicos tan complejos como un dron requiere ir pasando, poco a poco, de la idea conceptual a algo que realmente funcione en código. En vez de lanzarse directamente al hardware real desde el principio, todo el proceso experimental se ha planteado usando un banco de pruebas Software-in-the-Loop (SITL). Esto permite trabajar en un entorno virtual 3D bastante fiel a la realidad física, perfecto para validar la lógica de control, estudiar cómo se comporta el vehículo en el espacio y comprobar que las comunicaciones funcionan bien, todo esto sin poner en riesgo el dron de verdad.

3.1 Metodologías y Planificación

El desarrollo de este trabajo se ha llevado a cabo mediante una estrategia metodológica basada en una implementación secuencial por fases, orientada a garantizar la estabilidad del sistema antes de abordar la programación de los módulos de vuelo autónomo. Dado que todo el proyecto se ejecuta en un entorno de simulación, se adoptó un enfoque de prototipado lineal, lo que permitió avanzar de manera controlada en la integración de los distintos componentes del ecosistema software.

Las tareas de ingeniería se centraron en tres ejes fundamentales: la correcta integración de subsistemas y dependencias, el ajuste preciso de las coordenadas de navegación en el espacio simulado, y la depuración progresiva de los scripts de automatización, con el fin de adaptar el comportamiento del vehículo a las condiciones del entorno virtual.

Para asegurar el correcto funcionamiento de todas las herramientas de código abierto empleadas, la planificación y estructura del proyecto se organizó en seis fases operativas consecutivas, cada una orientada a consolidar un bloque funcional antes de avanzar al siguiente:

- Fase 1. Infraestructura y Sistema Operativo: donde se elegirá la máquina virtual de trabajo y el S.O anfitrión para desarrollar el trabajo.

- Fase 2. Instalación de las herramientas de código abierto: se explicará que aplicaciones y softwares se han instalado para la realización del proyecto.
- Fase 3. Entorno de programación y librerías de alto nivel: se escogerá Python como software de programación y sus correspondientes librerías.
- Fase 4. Ejecución de pruebas y simulaciones virtuales:
- Fase 5. Análisis de resultados y validación
- Fase 6. Redacción de la memoria

Fase 1. Infraestructura y Sistema Operativo (S.O)

La primera parte del proyecto consistió en montar un entorno de trabajo completamente aislado, para evitar problemas con las dependencias del sistema anfitrión y asegurar que todo el proceso fuera fácilmente reproducible.

Para esto se optó por crear una máquina virtual con VMware, instalando ahí Ubuntu 24.04 LTS, que se eligió por su estabilidad y porque es, sin duda, el ecosistema más usado para desarrollar herramientas de robótica aérea.

Y no es solo por comodidad, Ubuntu tiene un kernel sólido y muy bien documentado, además de buena compatibilidad nativa con las librerías y compiladores que se usan normalmente en sistemas de navegación autónoma. Esto es importante a la hora de ejecutar procesos en tiempo real y mantener un comportamiento estable durante la simulación y la compilación del software.

Fase 2. Instalación de las herramientas de código abierto

Con el sistema operativo ya listo, el siguiente paso fue ir instalando los componentes principales de la arquitectura del proyecto. Lo primero fue instalar ROS 2, que actúa como columna vertebral de toda la comunicación entre nodos y permite usar el puente Micro XRCE-DDS, necesario más adelante para comunicarse con el autopiloto de forma ligera y eficiente.

Una vez montada la parte de comunicaciones, se descargó, compiló e integró el firmware oficial del autopiloto PX4 junto con Gazebo, el motor de físicas 3D. Esta combinación es la que permite tener un entorno de simulación que se comporta de forma muy parecida a un vuelo real.

Todo esto terminó con la puesta en marcha del banco de pruebas SITL (Software-in-the-Loop), un entorno que ejecuta el código de vuelo real, pero recibiendo datos sintéticos

generados por el simulador. Gracias a esto, el autopiloto puede funcionar como si estuviera conectado a sensores físicos de verdad, recibiendo información simulada sobre gravedad, colisiones, fuerzas aerodinámicas, etc. Así se puede comprobar el comportamiento del sistema con bastante fidelidad, sin arriesgar nada de hardware real.
Más adelante, se tuvo que instalar un programa fuera de la máquina virtual para la creación del modelo 3D de los Gemelos Digitales (Digital Twins), Blender 5.1. Llegaba un momento en el que tener todo abierto y tratar de modelar el mapeado en Gazebo no funcionaba, ya que se colapsaba la máquina virtual por falta de recurso que ofrece el ordenador en el cual se trabajaba.

Fase 3. Entorno de programación y librerías de alto nivel.

En esta fase se preparó la capa de abstracción necesaria para poder controlar la aeronave mediante código. Para esto se montó un entorno de trabajo basado en Python 3, creando un entorno virtual dentro de Ubuntu. Esto es algo bastante habitual en ingeniería de software, y sirve para aislar las dependencias y evitar que las librerías del sistema choquen con las que necesita el proyecto.
Dentro de este entorno se instalaron MAVSDK y los componentes relacionados con el protocolo MAVLink, que son la base para comunicarse con el autopiloto a un nivel más alto porque es la que proporciona la API necesaria para no tener que lidiar con los mensajes en bruto, y permite, más adelante, ejecutar rutinas de vuelo en modo Offboard, donde el control del dron pasa completamente a manos de un programa externo.

Fase 4. Ejecución de pruebas y simulaciones virtuales

Una vez que la infraestructura de comunicaciones estaba lista y se comprobó que todos los componentes interactuaban bien entre sí, llegó el momento de centrarse en desarrollar los algoritmos y validar el software de control. Esta fue, en realidad, el núcleo experimental del proyecto, porque aquí es donde se ve cómo se comporta el sistema en condiciones simuladas de vuelo.
Para hacer las pruebas, se ejecutaron a la vez tres elementos principales:

- El entorno 3D de Gazebo, que genera la física del escenario
- La instancia virtual del autopiloto PX4, conectada internamente por puertos UDP
- Los scripts en Python encargados de enviar comandos y recibir telemetría a través de la API.

Con esto se pudieron recrear varios escenarios de vuelo, ajustando poco a poco los algoritmos de control y comprobando que todo se mantenía estable.

Además, se usó QGroundControl como herramienta de monitorización, lo que permitió ver en tiempo real variables importantes como la actitud, la posición, la velocidad o el estado de los sensores.

Gracias a poder supervisar todo esto en directo, se fueron ajustando las consignas de navegación, como los *"waypoints"*, perfiles de velocidad, tiempos de espera, transiciones entre fases de la misión, etc. para que el dron ejecutara sus trayectorias de forma autónoma y según lo planeado. Este proceso de ir probando, corrigiendo y volviendo a validar fue el que permitió pulir el comportamiento del sistema y asegurarse de que la lógica de vuelo respondía bien a lo que pasaba en el entorno simulado.

Fase 5. Análisis de resultados y validación

La última fase del proyecto se centró en analizar a fondo los datos sacados durante las sesiones de simulación, con la idea de comprobar de forma empírica que todo el sistema funcionaba como debía. Aquí no solo se revisó que las trayectorias programadas se ejecutaran correctamente, sino también que el dron respondiera de forma coherente a las órdenes que llegaban desde el script en Python, que en la práctica hacía de computador auxiliar controlando el vuelo en modo Offboard.

Para evaluar esto, se revisó con detalle la telemetría registrada, comparando la respuesta real del vehículo simulado con lo que se esperaba en cada fase de la misión. Se comprobó que las consignas de posición, velocidad y orientación se cumplían dentro de los márgenes esperados, y que las transiciones entre estados (despegue, navegación, espera, retorno o aterrizaje) se hacían de forma estable y sin sorpresas.

Una parte importante de esta fase fue comprobar la estabilidad del enlace MAVLink, que al final es la columna vertebral de toda la comunicación entre el autopiloto y el sistema de control externo. Se evaluó cómo se comportaba con distintas cargas de mensajes y durante maniobras más críticas, para asegurarse de que no hubiera pérdida de paquetes ni retrasos que pudieran comprometer la seguridad o el control del vuelo.

En general, esta fase permitió confirmar que todos los componentes (el simulador físico, el autopiloto PX4, la API de control y la estación de monitorización) funcionaban juntos cumpliendo con los requisitos de fiabilidad planteados al principio del proyecto. La validación empírica demostró que el sistema diseñado era capaz de ejecutar misiones

autónomas de forma consistente, lo cual es un paso clave antes de pensar en una futura migración a hardware real.

Fase 6. Redacción de la memoria

Como cierre de proyecto se fue elaborando el documento de la memoria de este TFG. Esta parte no fue solo escribir el texto final, sino que implicó organizar, depurar y dar forma a todos los elementos que componen la memoria de manera ordenada.

Primero se consolidó el marco teórico, juntando de forma coherente los fundamentos de navegación autónoma, los principios de los sistemas de control y la comparativa de los autopilotos analizados. Después se justificaron las decisiones tecnológicas que se fueron tomando a lo largo del desarrollo, explicando cada elección en función de los requisitos del proyecto y de sus limitaciones.

En paralelo, se documentó bien la metodología seguida, detallando las fases, los criterios de diseño, las herramientas usadas y cómo se validó todo. También se describió la arquitectura del código desarrollado, explicando cómo funcionan los distintos módulos, cómo interactúan entre ellos y cómo fluyen los datos dentro del sistema.

Por último, se redactaron las conclusiones a partir de lo que se obtuvo en la simulación, relacionándolo con los objetivos planteados al inicio del proyecto. Con esto se cerró el ciclo de desarrollo, dando una visión completa de todo el trabajo realizado y dejando abierta la puerta a posibles mejoras o futuras ampliaciones del proyecto.

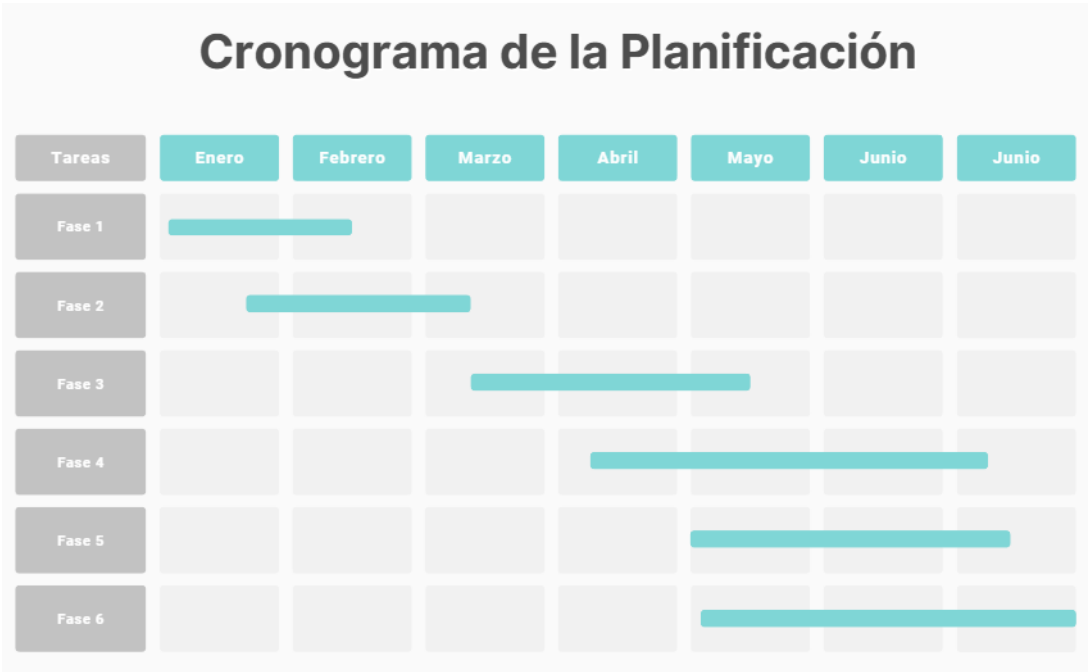


Figura 2. Cronograma de planificación. (Fuente Propia).

3.2 Implementación

En este apartado se explica la arquitectura de software y la lógica que hace posible que el UAV tenga capacidades de navegación autónoma. Una vez montada toda la infraestructura técnica explicada en la metodología, la fase de implementación se centró en conseguir un flujo continuo y coherente de información entre el controlador de vuelo, los algoritmos en Python y la interfaz gráfica usada para supervisar todo el sistema.

El desarrollo se organizó alrededor de una lógica única, cuyo objetivo era que el dron pudiera interpretar su entorno, procesar los datos de navegación y ejecutar maniobras autónomas de forma estable y predecible. Para esto, se configuró con cuidado el entorno virtual que sirve de base para la simulación, asegurando que los sensores simulados, los modelos aerodinámicos y los procesos internos del autopiloto estuvieran bien sincronizados. Encima de esta capa se construyó el módulo encargado de procesar matemáticamente las coordenadas espaciales, que es el que transforma y normaliza la información de posición y orientación, permitiendo que el dron sepa con precisión dónde está en el espacio tridimensional.

A partir de esta representación espacial, se desarrolló la lógica de navegación que define las trayectorias y el comportamiento autónomo del dron. Esta parte se apoya en sistemas visuales y en la interpretación de los datos de telemetría, lo que permite definir rutas, puntos de referencia y secuencias de misión adaptadas al escenario simulado. La forma en que interactúan estos componentes da lugar a una arquitectura sólida que permite ejecutar misiones autónomas con bastante fiabilidad.

3.2.1 Configuración y despliegue del entorno SITL

El primer paso de la implementación fue poner en marcha y configurar la arquitectura de simulación. Para no tener que escribir los mismos comandos a mano cada vez que se entra al Ubuntu 24.04 LTS, se creó varios scripts en bash llamados ./nave_industrial.sf, y ./paneles_solares.sf. Al ejecutarlo, el sistema compila y arranca a la vez el firmware PX4 en modo SITL junto con el entorno 3D de Gazebo, cargando automáticamente el modelo del quadrotor mediante px4_sitl gz_x500_depth.

Vale la pena mencionar que estos scripts se diseñaron para ser modulares y parametrizables. Como el proyecto necesita probar el dron en distintos escenarios virtuales, las coordenadas iniciales no son fijas: se pueden cambiar directamente en el script para mover el punto de referencia (0, 0, 0) según lo que necesite cada prueba.

Además, el script se encarga también de abrir automáticamente la interfaz de la estación de control ejecutando "./QGroundControl", dejando lista la telemetría y el enlace de datos local desde el principio.

Para mantener las dependencias bien aisladas, se activa el entorno virtual de Python3, que permite cargar solo las librerías MAVSDK y MAVLink necesarias. De esta forma, se pueden ejecutar los scripts de navegación autónoma y enviar comandos de control en un entorno limpio y sin interferencias con los paquetes globales del sistema.

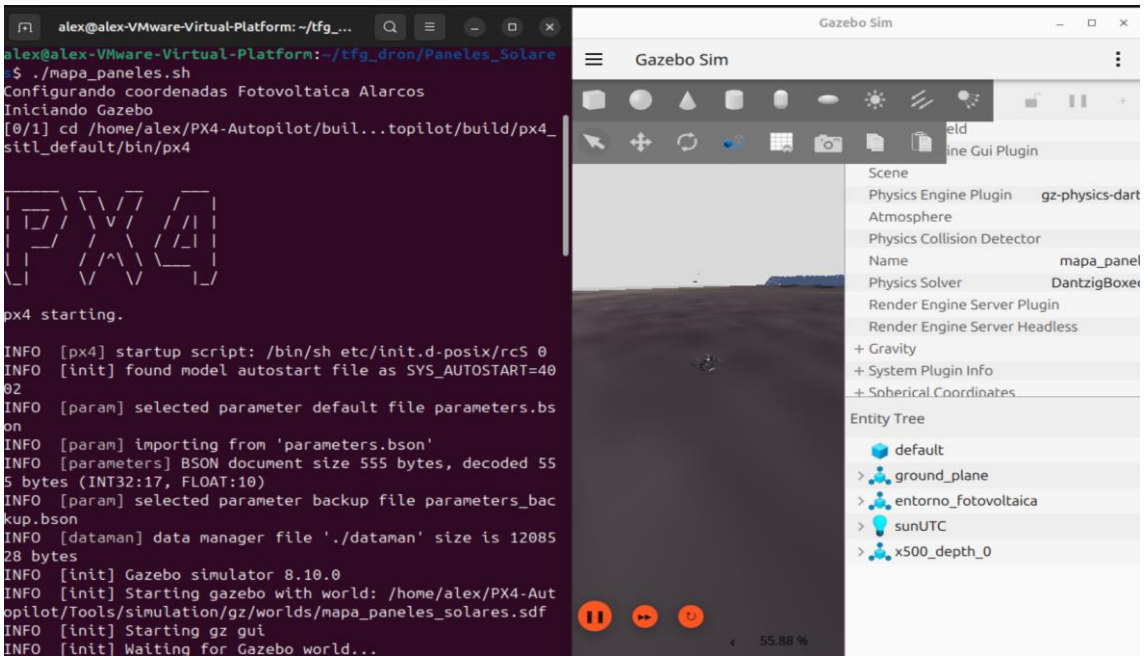


Figura 3. Comando Scripts Bash. (Fuente Propia).

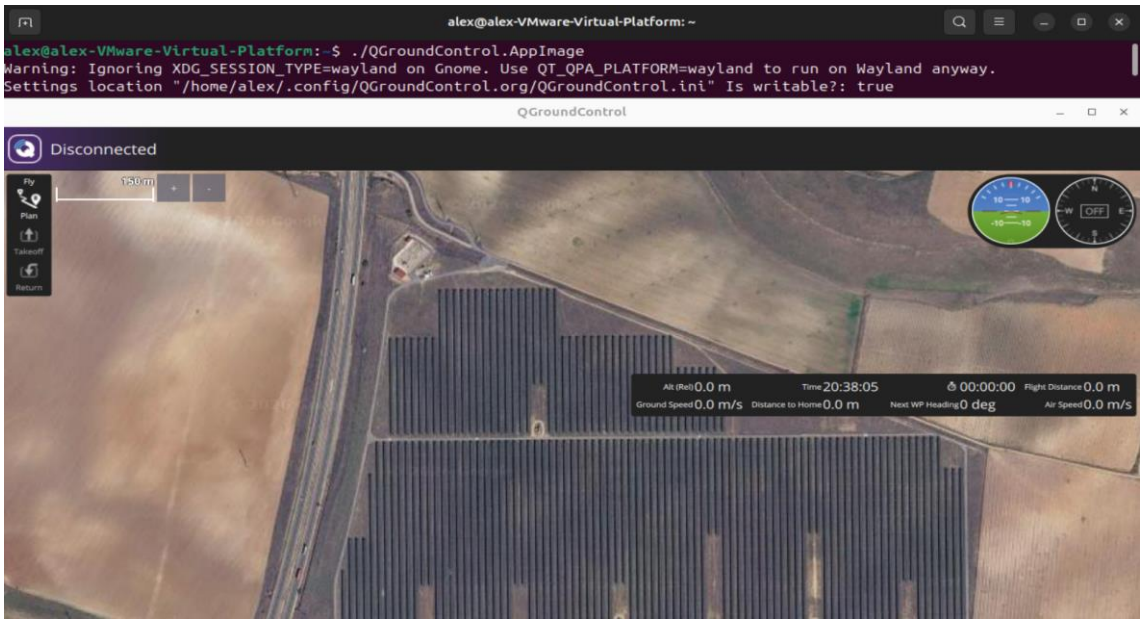


Figura 4. Comando QGroundControl. (Fuente Propia).

3.2.2 Tratamiento de coordenadas

Uno de los retos más importantes a nivel de software durante el desarrollo de la lógica de vuelo fue tener que lidiar con distintos sistemas de referencia. El problema es que planificar las trayectorias es mucho más intuitivo en un plano cartesiano 2D local, expresando los desplazamientos en metros hacia el norte y el este, pero el autopiloto PX4 y los sistemas GPS necesitan que las rutas se transmitan en coordenadas geográficas WGS84, es decir, en latitud y longitud. Para resolver esta incompatibilidad y que el código siguiera siendo escalable, se implementó dentro del módulo principal de navegación una función específica de transformación de coordenadas llamada mover_en_metros, cuyo objetivo es calcular una nueva posición GPS a partir de un vector de desplazamiento en metros (x, y) aplicado sobre un punto de origen conocido, que corresponde a la posición Home registrada al inicio de la misión.

```python
class PilotoAutonomoPaneles:

    def __init__(self, lat_base, lon_base):
        # Guardamos la posición que el mapa detectó automáticamente
        self.lat_base = lat_base
        self.lon_base = lon_base
    def mover_en_metros(self, metros_norte, metros_este):
        """Calcula una nueva coordenada GPS sumando metros exactos a la base."""
        r_tierra = 6378137.0
        d_lat = metros_norte / r_tierra
        d_lon = metros_este / (r_tierra * math.cos(math.pi * self.lat_base / 180.0))

        nueva_lat = self.lat_base + (d_lat * 180.0 / math.pi)
        nueva_lon = self.lon_base + (d_lon * 180.0 / math.pi)
        return nueva_lat, nueva_lon
```

Figura 5.Tratamiento de coordenadas en Python. (Fuente Propia).

3.2.3 Planificación y automatización de las misiones

Una vez resuelto el problema de la transformación de coordenadas, el siguiente paso fue diseñar el motor lógico encargado de planificar las trayectorias de vuelo. La idea era demostrar la versatilidad y robustez de la arquitectura desarrollada programando y validando dos perfiles de misión completamente distintos, ambos representativos de casos de uso reales que se piden en la industria. Aprovechando la función de desplazamiento relativo del apartado anterior, se automatizó la generación de waypoints para dos escenarios diferenciados: la inspección de una planta fotovoltaica y la vigilancia perimetral de una nave industrial.

En el primer caso, el de la inspección de paneles solares, el objetivo de esta misión es obtener información termográfica o imágenes visuales de las placas solares, lo que exige que el dron recorra toda el área siguiendo un patrón en serpentina o meandro, conocido en robótica como "lawnmower", para cubrir la superficie de forma completa y sistemática. Para implementar esta trayectoria, se desarrolló un algoritmo generador de rutas basado en waypoints relativos, de manera que la planificación pudiera adaptarse a la geometría real del terreno sin tener que programar cada punto GPS a mano.

El diseño del sistema incluye un desplazamiento inicial que coloca al dron en la primera fila de paneles antes de empezar el recorrido, y construye la trayectoria de forma explícita mediante una lista de puntos de paso que delimitan con precisión el inicio y el final de cada pasada sobre las placas. Esto evita tratar el área como un bloque continuo y asegura que el barrido sea exacto. Además, la lógica interna del algoritmo gestiona automáticamente la alternancia del sentido longitudinal del vuelo, y, al terminar cada tramo recto, introduce un "*waypoint*" intermedio que aplica una separación lateral constante hacia el Este, moviendo el dron a la siguiente fila antes de invertir el rumbo y continuar con el patrón de inspección.

En el segundo caso, el de la nave industrial, el tipo de vuelo cambia completamente, porque ya no se trata de cubrir un área plana sino de hacer un circuito de patrullaje alrededor de una estructura tridimensional. A diferencia de la planta fotovoltaica, la nave industrial plantea un tipo de vuelo completamente distinto ya que no hace falta cubrir un área plana con pasadas sucesivas, sino ejecutar un circuito de patrullaje alrededor de una estructura tridimensional. Para esta misión se reutilizó el mismo núcleo de cálculo de coordenadas del caso anterior, adaptando la lógica geométrica para generar un recorrido cerrado con puntos estratégicos alrededor de las fachadas de la nave. Esto demuestra lo flexible que es la arquitectura desarrollada, ya que con el mismo núcleo se pueden abordar misiones muy distintas. La planificación se centró en definir rutas basadas en polígonos que permiten ajustar parámetros clave en cada vértice, como el yaw o guiñada, para que la cámara del dron apunte en todo momento hacia la estructura o hacia las posibles vías de acceso, garantizando una vigilancia continua y coherente con los objetivos de la misión.

Independientemente del tipo de misión, el proceso de automatización siempre termina igual para mantener el código estandarizado: una vez generada la matriz de coordenadas relativas, el sistema itera sobre ella para crear objetos MissionItem de la API de MAVSDK, inyectando los catorce parámetros que exige el protocolo, entre los que están la velocidad de crucero, la altitud relativa de seguridad y los radios de aceptación de los waypoints. Finalmente, todo se empaqueta en un MissionPlan y, mediante funciones asíncronas de Python, el sistema limpia la memoria del autopiloto y le transfiere la nueva ruta generada dinámicamente. Al lanzar la orden de inicio de misión, el control pasa completamente al firmware PX4, y el dron despega, navega, ejecuta su cometido y vuelve solo a la base de forma totalmente autónoma.

3.3 Integración

Una vez terminados por separado el módulo de transformación de coordenadas y el de planificación geométrica, la fase de integración consistió en unir ambos componentes dentro de una arquitectura de software centralizada. El objetivo es establecer un flujo de comunicación bidireccional, estable y fiable entre la lógica de alto nivel del script en Python y el firmware de bajo nivel de PX4, para que el ciclo de vida de las misiones autónomas se ejecutara correctamente. Para esto se diseñó un orquestador principal llamado ***main.py***, construido sobre la API de MAVSDK, que actúa como núcleo coordinador de toda la operación. Este orquestador es el que permite que los distintos módulos del sistema trabajen de forma sincronizada, integrando en un único flujo la planificación, el envío de comandos y la supervisión del estado del vehículo.

3.3.1 Arquitectura Desacoplada y Flujo de Datos

Siguiendo los principios fundamentales de la ingeniería de software, la integración del sistema se llevó a cabo mediante un enfoque estrictamente modular, en el que el orquestador principal actúa como un componente de coordinación y no como un elemento de cálculo, ya que carece por completo de lógica matemática y se limita a gestionar el flujo de información entre los distintos módulos funcionales. En este esquema, el proceso integrado comienza con la captura del entorno operativo: el orquestador obtiene en tiempo real la telemetría del dron y, a partir de ella, registra dinámicamente la coordenada GPS de origen (*Home*), que constituye el punto de referencia inicial sobre el cual se construye toda la misión autónoma.

Una vez adquirida esta información, el sistema procede a inyectarla durante la creación de la clase externa de navegación (*PilotoAutonomo*), estableciendo así un vínculo directo entre el mundo físico y el espacio matemático en el que se procesan los desplazamientos relativos, lo que garantiza la coherencia entre los datos sensoriales y la lógica geométrica. Cuando el módulo de navegación finaliza el cálculo de la matriz de *waypoints*, dicha estructura es devuelta al orquestador, que asume la responsabilidad de transferirla al autopiloto mediante el protocolo **MAVLink**, cargándola en su memoria interna para su posterior ejecución. De este modo, el orquestador se convierte en el eje vertebrador del sistema, gestionando la adquisición de datos, la inyección de dependencias y la transferencia final de la misión, y asegurando una arquitectura limpia, escalable y plenamente alineada con las buenas prácticas del diseño modular.

3.3.2 Máquina de Estados del Vuelo

Para que todo sea seguro, la integración sigue una máquina de estados secuencial que el sistema debe respetar sí o sí:

- Fase de Conexión y Sincronización: el sistema conecta a través del puerto udp://:14540. Antes de permitir cualquier movimiento, el orquestador hace un chequeo de salud (Health Check) y se queda bloqueado hasta confirmar que la posición global es válida y que hay una buena triangulación de satélites.
- Fase de Despliegue: una vez pasados los chequeos y transferido el plan de vuelo, el sistema manda los comandos de armado de motores (Arm) y autorización de despegue. A partir de ahí empieza la patrulla autónoma.
- Monitorización y Retorno: Mientras el dron está volando, el sistema mantiene un bucle de monitorización activa que va informando por terminal de cómo va la misión (waypoint actual frente al total). Cuando llega al último objetivo, se activa automáticamente el protocolo de vuelta a casa (RTL - Return To Launch).

3.3.3 Asincronía y Tolerancia a Fallos

Dada la naturaleza de las comunicaciones robóticas, el orquestador principal se programó usando la librería “asyncio” de Python. La ejecución asíncrona permite que el programa gestione varios flujos de datos a la vez sin bloquear el procesador. Gracias a esto, el sistema puede estar mandando órdenes de control al autopiloto mientras, al mismo

tiempo, escucha la telemetría del dron y monitoriza el progreso de la misión, lo que garantiza una operación fluida y sin latencias críticas en la comunicación.

3.3.4 Implementación de Sistema de Gestión de Emergencias y Puntos de Contingencia (Rally Points)

La robustez de un sistema de vuelo autónomo no se limita a la capacidad de ejecutar trayectorias predefinidas en condiciones ideales, sino que debe medirse por su resiliencia ante interrupciones críticas y fallos imprevistos en la estación de control; con este criterio, y con el objetivo de dotar a la aeronave de un comportamiento tolerante a fallos, se incorporó en el hilo principal de ejecución (main.py) un bloque de gestión de excepciones diseñado para interceptar las interrupciones manuales del sistema, simuladas mediante la señal de teclado *KeyboardInterrupt*, de modo que cualquier evento de este tipo sea detectado de forma inmediata y determinante por la lógica de control.

Al producirse una interrupción abrupta de la misión, el mecanismo implementado evita que la aeronave continúe su trayectoria de manera descontrolada redirigiendo de forma inmediata el flujo de ejecución hacia una corrutina de contingencia denominada *emergencia_rally_points(),* la cual centraliza las acciones de seguridad necesarias para recuperar un estado operativo seguro; esta estrategia garantiza que, ante una pérdida de control desde la estación o una intervención manual, el sistema ejecute una secuencia predefinida de medidas protectoras sin depender de la intervención humana, preservando así la integridad del vehículo y del entorno.

```
if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        # Al pulsar Ctrl+C, activamos el sistema híbrido de rescate
        asyncio.run(emergencia_rally_points())
```

Para garantizar una toma de decisiones contextualizada, dicha subrutina de emergencia establece de forma autónoma una nueva conexión con el piloto automático y recupera en

tiempo real tanto las coordenadas del punto de despegue original (*Home*) como la posición satelital exacta del dron en el instante preciso en que se produjo la anomalía, de modo que la información empleada para la resolución del fallo refleja el estado operativo más reciente. A continuación, el sistema genera dinámicamente una matriz de zonas seguras de aterrizaje que combina la base principal con una serie de puntos de contingencia, los "*rally points*", estratégicamente distribuidos en el perímetro del área operativa, proporcionando así un conjunto priorizado de alternativas de recuperación que la corrutina emergencia_rally_points() puede ejecutar de forma determinista para restablecer un estado operativo seguro.

```
# Definimos las zonas seguras INCLUYENDO la Base Principal
    puntos_de_aterrizaje = [
        {"nombre": "Base Principal (RTL)", "lat": lat_home, "lon": lon_home, "es_base": True},
        {"nombre": "Zona Segura A", "lat": 40.437076, "lon": -3.562491, "es_base": False},
        {"nombre": "Zona Segura B", "lat": 40.436076, "lon": -3.564669, "es_base": False}
    ]
```

Para dotar al vehículo de autonomía decisional, el algoritmo debe determinar cuál de estas áreas designadas representa la opción más segura y eficiente para el aterrizaje, dado que las coordenadas se obtienen en formato esférico (latitud y longitud), el cálculo de la proximidad espacial se resuelve implementando computacionalmente la fórmula del semiverseno (Haversine). Esta aproximación matemática permite calcular con precisión la distancia ortodrómica d entre dos puntos sobre la superficie terrestre, mitigando la distorsión que generaría un cálculo euclidiano plano. La formulación matemática aplicada es la siguiente:

$$a = \sin^2\left(\frac{\Delta \text{lat}}{2}\right) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2\left(\frac{\Delta \text{lon}}{2}\right)$$

$$d = 2R \cdot \arcsin(\sqrt{a})$$

Ecuación 1. Fórmulas del Semiverseno. (Generado por IA).

, donde R representa el radio ecuatorial de la Tierra (aproximadamente 6371 km), y Δlat y Δlon denotan las diferencias angulares entre la posición actual del dron y el punto de evaluación.

Por último, una vez identificada la coordenada de salvaguarda más cercana, el sistema de control ejecuta un árbol de decisiones bastante restrictivo.

Si el cálculo determina que el punto de partida original sigue siendo la mejor opción, la aeronave lanza directamente el comando nativo Return To Launch (RTL). En cambio, si un Rally Point secundario resulta más conveniente, el dron interrumpe de inmediato su plan de vuelo inyectando un comando de retención (hold), lo que neutraliza la inercia del vehículo y lo estabiliza en el aire.

Una vez cancelada la misión activa, el autopiloto recalcula un vector directo hacia las coordenadas de rescate y, tras esperar un margen de tiempo prudencial para completar el desplazamiento, inicia una maniobra de descenso vertical automatizado (*land()*), garantizando en todo momento la integridad del equipo y la seguridad del entorno.

```
if punto_mas_cercano["es_base"]:
        print("[EMERGENCIA] Activando protocolo nativo RTL...")
        await drone_rescate.action.return_to_launch()
    else:
        print("[EMERGENCIA] Desviando trayectoria. Abortando misión activa...")
        # Forzamos al dron a ponerse en modo HOLD para cancelar la misión
```

```
        await drone_rescate.action.hold()

        await asyncio.sleep(5)

        print(f"[EMERGENCIA] Redirigiendo a zona segura alternativa:
{punto_mas_cercano['nombre']}...")

        await
drone_rescate.action.goto_location(punto_mas_cercano["lat"],
punto_mas_cercano["lon"], float('nan'), 0)

        # Margen de tiempo para asegurar la llegada al punto de
aterrizaje

        await asyncio.sleep(120)

        print("[EMERGENCIA] Posición alcanzada. Iniciando descenso
vertical de seguridad.")

        await drone_rescate.action.land()
```

Capítulo 4. Pruebas y Evaluación

4.1 Definición de las Pruebas

La validación integral de la arquitectura de software desarrollada exige la construcción de un marco de experimentación progresivo y estructurado dentro del entorno de simulación SITL, cuya finalidad es verificar tanto la estabilidad de las comunicaciones subyacentes como la fiabilidad de los algoritmos de vuelo autónomo bajo condiciones operativas diversas, de manera que el sistema pueda evaluarse con un rigor metodológico acorde a los estándares de la ingeniería de sistemas. Con esta finalidad, el proceso de validación se ha articulado en tres etapas consecutivas de complejidad creciente que permiten avanzar desde la comprobación de los subsistemas fundamentales hasta la auditoría de comportamientos críticos en pleno vuelo, asegurando que cada fase consolide los resultados de la anterior y contribuya a una evaluación global coherente.

En primer lugar, la validación se orienta a verificar la infraestructura base y la cinemática elemental del dron, comenzando por el despliegue sincronizado del motor físico Gazebo, el firmware **PX4** y la estación terrestre QGroundControl, con el objetivo de confirmar la estabilidad del flujo bidireccional de telemetría y la correcta representación tridimensional del gemelo digital. Una vez asegurada la integridad del enlace de comunicaciones, se ejecuta una orden automatizada de despegue y vuelo estacionario que permite comprobar que los controladores internos del autopiloto mantienen la aeronave estabilizada frente a perturbaciones simuladas

Superada la fase inicial, la validación avanza hacia la evaluación dinámica del sistema operando bajo el paradigma **Offboard**, en el que el orquestador desarrollado en Python inyecta consignas de posición en tiempo real para guiar al UAV a través de los patrones de inspección en serpentina sobre la planta fotovoltaica y del patrullaje perimetral tridimensional alrededor de la nave logística.

Finalmente, la validación termina con la auditoría de la autonomía total delegada al autopiloto mediante el paradigma **Mission**, proceso que implica la carga previa en la memoria interna del sistema de las matrices de *waypoints* asociadas a ambos escenarios operativos para verificar el cumplimiento estricto de las trayectorias predefinidas sin

intervención externa. El objetivo de esta etapa consiste en simular una interrupción crítica de los sistemas en pleno vuelo automático con el fin de evaluar la eficacia de la subrutina de emergencia, cuyo criterio de éxito se fundamenta en la detección inmediata del evento anómalo, la interrupción de la misión activa, el cálculo matemático en tiempo real de las distancias ortodrómicas y, como resultado final, el aterrizaje autónomo de la aeronave en el *Rally Point* de contingencia más adecuado. Con ello se certifica que la arquitectura desarrollada no solo es capaz de ejecutar misiones complejas, sino también de responder de manera segura y determinista ante situaciones críticas, consolidando así la fiabilidad global del sistema.

ID	Descripción
P-01	Conectividad SITL y despliegue del gemelo digital. Sincronización entre QGroundControl, PX4 y Gazebo.
P-02	Conexión con el dron
P-03	Estabilización y vuelo estacionario
MO-01	Ejecución misión en modo Offboard de la planta fotovoltaica (Python)
MO-02	Ejecución misión en modo Offboard de la nave industrial (Python)
MO-03	Inyección de fallos y aborto de misión (Modo Offboard)
MM-01	Ejecución misión en modo "*Mission*" de la planta fotovoltaica (Python)
MM-02	Ejecución misión en modo "*Mission*" de la nave industrial (Python)
MM-03	Inyección de fallos y aborto de misión (*Mission Mode*)

Tabla 2. Tabla de misiones/pruebas realizadas.

4.2 Análisis e Interpretación de los Resultados

En esta sección se expondrán y evaluarán los resultados obtenidos tras la ejecución del plan de validación. Los análisis se han estructurado en tres bloques, siendo las siguientes:

- Verificación de la infraestructura (P-X)
- Evaluación en Modo Offboard (MO-XX)
- Evaluación en Modo Mission (MM-XX)

4.2.1 Pruebas preliminares

Prueba P-01. Conectividad SITL

El objetivo de esta prueba inicial fue certificar la correcta sincronización de la arquitectura de simulación, ejecutando los scripts de inicialización se confirmó el arranque simultáneo del firmware PX4 y la renderización del entorno tridimensional en Gazebo. Como se puede observar en la Figura XX, la estación terrestre QGroundControl estableció un enlace telemétrico estable, demostrando que el ecosistema Software-in-the-Loop es capaz de sostener una comunicación bidireccional sin latencias críticas antes de iniciar cualquier operación.

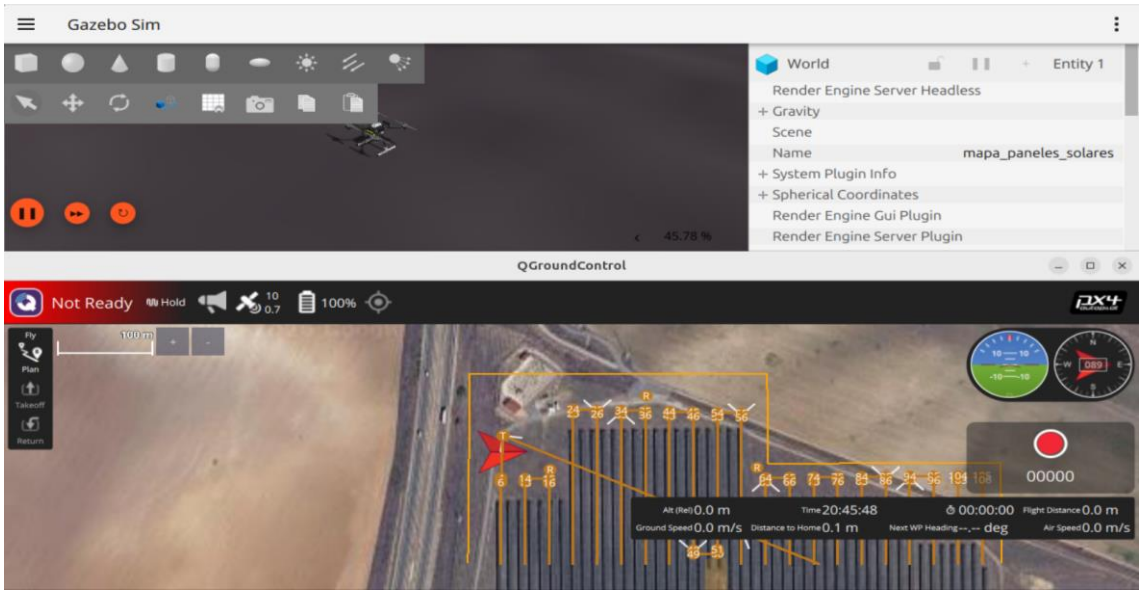


Figura 6. QGroundControl y Gazebo en funcionamiento. (Fuente Propia).

Prueba P-02. Conexión con el dron

Una vez garantizado el despliegue del entorno virtual, se procedió a validar el puente de comunicación entre la lógica de control de alto nivel y el autopiloto.

Mediante la ejecución del orquestador programado en Python y la librería MAVSDK, el sistema logró conectarse exitosamente al vehículo, superando los chequeos de salud pre-vuelo y adquiriendo la señal GPS simulada necesaria para fijar la coordenada inercial de origen (Home/RTL).

Este hito confirma que la capa de software externa dispone de los privilegios necesarios para comandar la aeronave.

```
INFO  [gz_bridge] world: mapa_paneles_solares, model: x500_depth_0
INFO  [init] Setting camera to follow x500_depth_0
INFO  [init] Camera follow offset set to -2.0, -2.0, 2.0
INFO  [lockstep_scheduler] setting initial absolute time to 3016000 us
INFO  [commander] LED: open /dev/led0 failed (22)
WARN  [health_and_arming_checks] Preflight Fail: ekf2 missing data
WARN  [health_and_arming_checks] Preflight Fail: system power unavailable
WARN  [health_and_arming_checks] Preflight Fail: No connection to the GCS
INFO  [tone_alarm] home set
INFO  [uxrce_dds_client] init UDP agent IP:127.0.0.1, port:8888
INFO  [mavlink] mode: Normal, data rate: 4000000 B/s on udp port 18570 remote port 14550
INFO  [mavlink] mode: Onboard, data rate: 4000000 B/s on udp port 14580 remote port 14540
INFO  [mavlink] mode: Onboard, data rate: 4000 B/s on udp port 14280 remote port 14030
INFO  [mavlink] partner IP: 127.0.0.1
INFO  [mavlink] mode: Gimbal, data rate: 400000 B/s on udp port 13030 remote port 13280
INFO  [logger] logger started (mode=all)
INFO  [logger] Start file log (type: full)
INFO  [logger] [logger] ./log/2026-07-02/18_45_12.ulg
INFO  [logger] Opened full log file: ./log/2026-07-02/18_45_12.ulg
INFO  [mavlink] MAVLink only on localhost (set param MAV_{i}_BROADCAST = 1 to enable network)
INFO  [mavlink] MAVLink only on localhost (set param MAV_{i}_BROADCAST = 1 to enable network)
INFO  [px4] Startup script returned successfully
pxh> INFO  [tone_alarm] notify positive
WARN  [health_and_arming_checks] Preflight Fail: heading estimate not stable
```

Figura 7. Conexión con el dron en Gazebo y QGroundControl. (Fuente Propia).

Prueba P-03. Estabilización y vuelo estacionario

La última prueba de esta fase preliminar evalúa la respuesta cinemática del dron ante un primer despegue automatizado. Tras enviar el comando de armado de motores y elevación vertical mediante la API de MAVSDK, el UAV alcanzó la altitud de seguridad estipulada y mantuvo un estado de vuelo estacionario de unos 10 segundos con un éxito rotundo. El comportamiento observado evidenció una correcta interpretación de las instrucciones de alto nivel por parte del firmware de PX4 permitiendo que el dron se mantuviera posicionada y estabilizada en el entorno tridimensional de forma consistente.

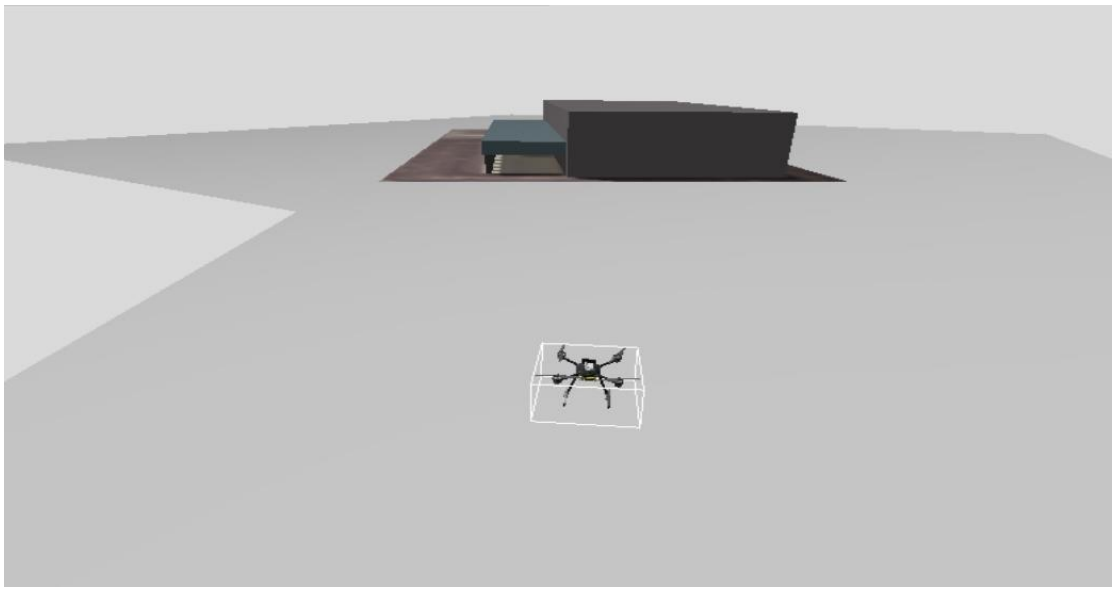


Figura 8. Vuelo estacionario del dron. (Fuente Propia).

4.2.2 Offboard Mode

Misión MO-01. Inspección planta fotovoltaica

Para este primer escenario de prueba, se ha desarrollado un script en Python para que el dron recorra de manera autónoma una planta fotovoltaica. El proceso se inicia mediante la ejecución de un lanzador .sh, momento en el cual se verifica cómo el dron se sitúa de forma precisa en las coordenadas asignadas a través de QGroundControl, reflejándose simultáneamente su posición sobre el gemelo digital en el entorno de simulación de Gazebo. Una vez ejecutado el script de control, el dron procede al armado automático de sus motores y comienza la navegación de la zona siguiendo la secuencia de *waypoints* previamente delimitados. El vuelo se mantiene a una altitud constante de entre 4 y 5 metros para simular de forma realista las condiciones de una inspección visual de los paneles solares. Toda la misión transcurre sin ninguna incidencia; el dron barre la totalidad del mapa previsto y, al finalizar su ruta, ejecuta exitosamente el comando RTL (*Return to Launch*) para regresar de forma segura a su punto de origen.

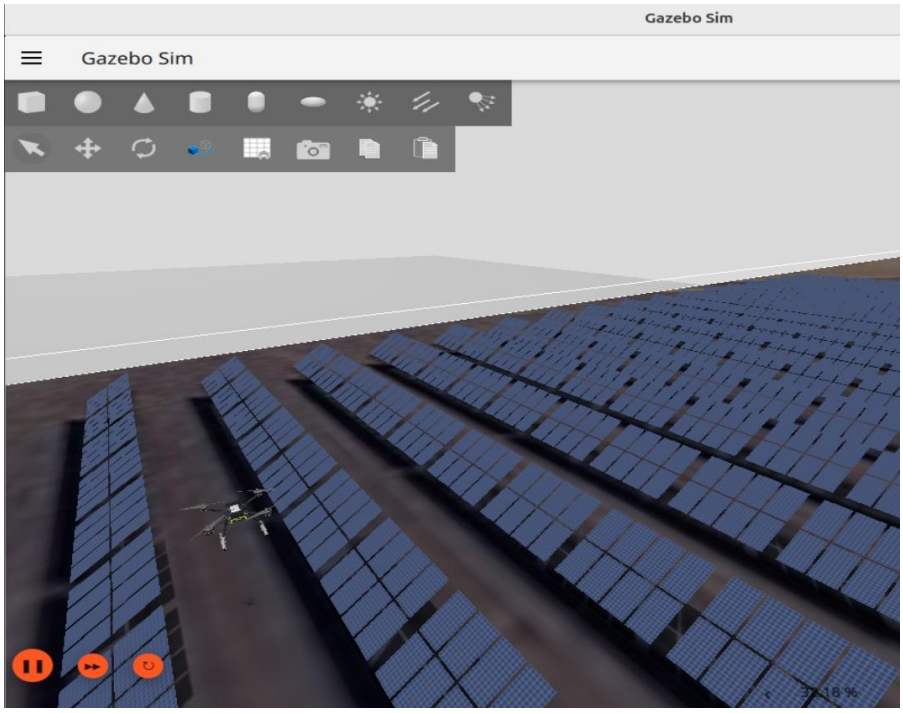


Figura 9. UAV realizando misión en Panales Fotovoltaicos. (Fuente Propia).

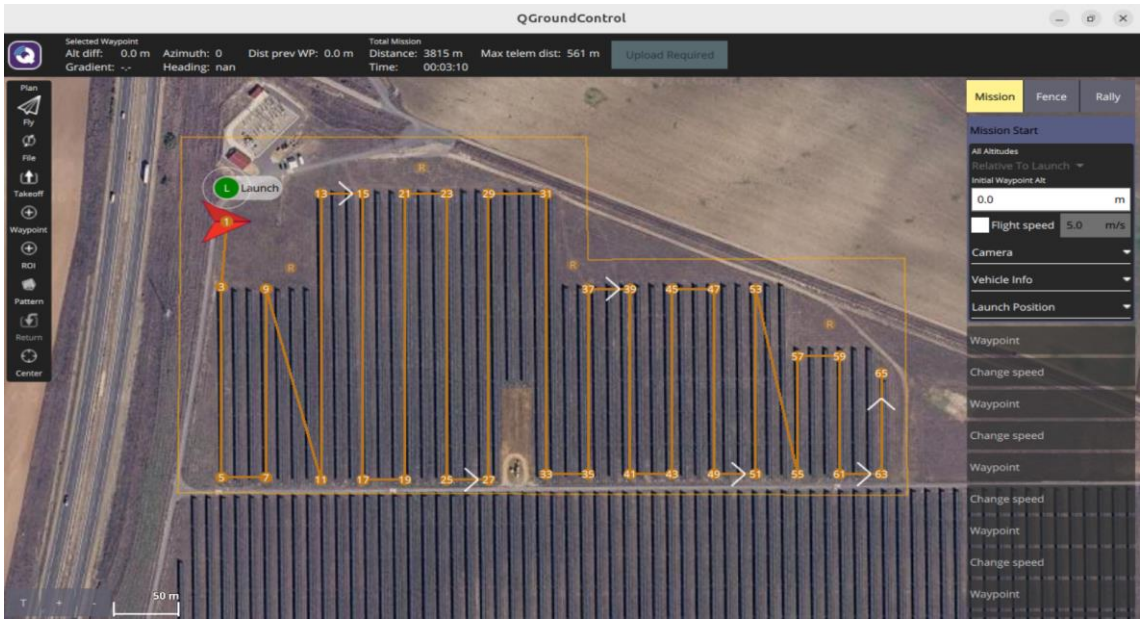


Figura 10. Visualización de la ruta en QGroundControl. (Fuente Propia).

Misión MO-02. Vigilancia de nave industrial

El segundo escenario de prueba emplea la misma arquitectura de control y sincronización entre QGroundControl y Gazebo que el caso anterior, pero se adapta a un entorno industrial situado en Coslada. En esta ocasión, la navegación autónoma sobre los *waypoints* establecidos se ejecuta manteniendo una altitud de vuelo superior, situada en un rango de 25 a 30 metros. Este incremento en la altura resulta indispensable para garantizar un franqueo seguro de los distintos edificios estructurales presentes en la zona y, al mismo tiempo, proporcionar un campo de visión lo suficientemente amplio como para llevar a cabo de forma efectiva las labores de vigilancia perimetral de la nave. Al igual que en la primera prueba, la aeronave completa la ruta delimitada de manera totalmente satisfactoria.

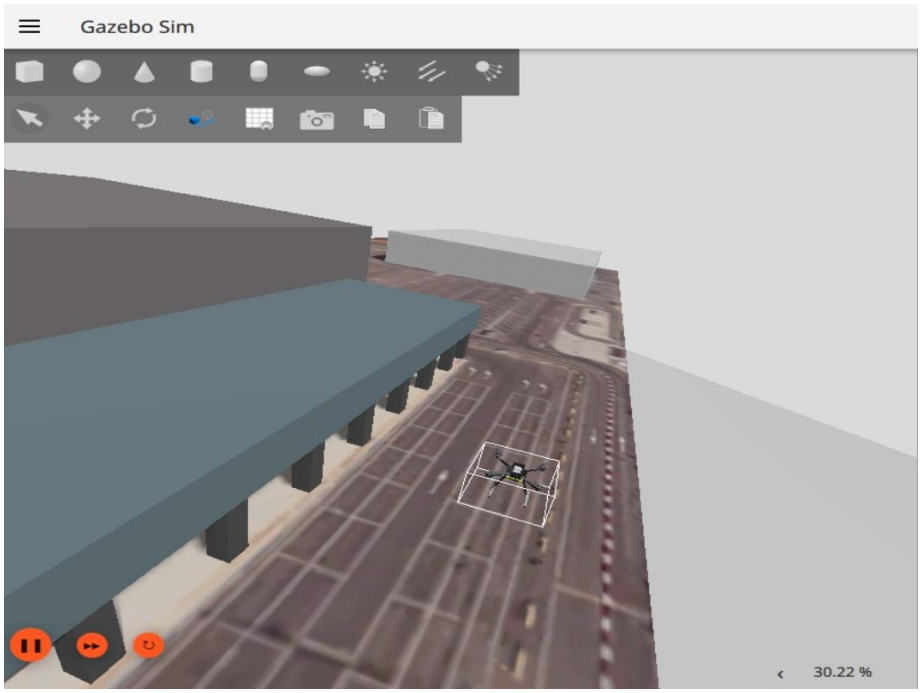


Figura 11. UAV realizando la misión en la nave industrial. (Fuente Propia).

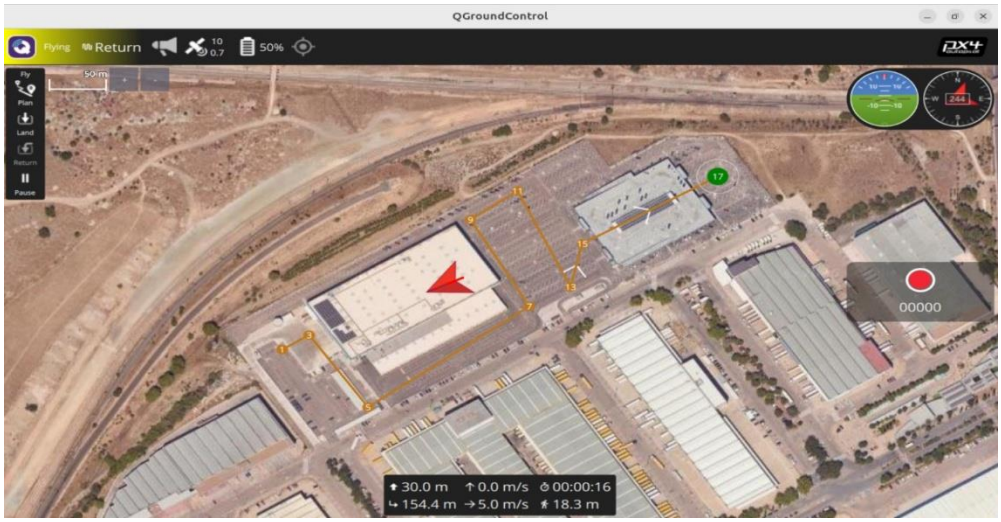


Figura 12. Visualización de la misión en QGroundControl. (Fuente Propia).

Misión MO-03. Sistemas de restricción espacial y protocolos de emergencia

Para esta última prueba, la validación se centra en la programación específica implementada en el script para gestionar contingencias y delimitar el espacio aéreo.

En primer lugar, se constata el correcto funcionamiento del sistema de *Geofencing*, comprobando que el dron es capaz de evadir activamente la navegación por zonas de exclusión delimitadas (mediante un Geofence interno) y de mantenerse operando

estrictamente dentro de un área permitida (mediante un Geofence externo). En segundo lugar, se evalúa el protocolo de fallos a través de la configuración de *Rally Points* (RP), para demostrar que, en caso de sufrir una falla simulada durante el vuelo, la aeronave calcula de manera autónoma cuál es el punto de encuentro más cercano a su posición actual, lo que le permite ejecutar un aterrizaje de emergencia rápido y seguro para minimizar cualquier riesgo operativo.

```python
async def emergencia_rally_points():
    print("\n[EMERGENCIA] Ejecución interrumpida por el usuario (Ctrl+C).")
    print("[EMERGENCIA] Conectando sistema de rescate...")
    drone_rescate = System()
    await drone_rescate.connect(system_address="udp://:14540")

    # 1. Recuperamos la posición HOME (Base) fijada al inicio
    lat_home, lon_home = 0.0, 0.0
    async for home in drone_rescate.telemetry.home():
        lat_home = home.latitude_deg
        lon_home = home.longitude_deg
        break

    # 2. Definimos las zonas seguras INCLUYENDO la Base Principal
    puntos_de_aterrizaje = [
        {"nombre": "Base Principal (RTL)", "lat": lat_home, "lon": lon_home, "es_base": True},
        {"nombre": "Zona Segura A", "lat": 40.436304, "lon": -3.564287, "es_base": False},
        {"nombre": "Zona Segura B", "lat": 40.437287, "lon": -3.562662, "es_base": False}
    ]

    # 3. Obtenemos la posición actual exacta del dron al momento del corte
    lat_actual, lon_actual = 0.0, 0.0
    async for position in drone_rescate.telemetry.position():
        lat_actual = position.latitude_deg
        lon_actual = position.longitude_deg
        break
```

```python
print(f"[EMERGENCIA] El punto más cercano es {punto_mas_cercano['nombre']} a {distancia_minima:.1f} metros.")

if punto_mas_cercano["es_base"]:
    print("[EMERGENCIA] Activando protocolo nativo RTL...")
    await drone_rescate.action.return_to_launch()
else:
    print("[EMERGENCIA] Desviando trayectoria. Abortando misión activa...")

    # Forzamos al dron a ponerse en modo HOLD (frena en seco y cancela la misión)
    await drone_rescate.action.hold()
    await asyncio.sleep(5) # Le damos cinco segundos para procesar el cambio de modo

    print(f"[EMERGENCIA] Redirigiendo a zona segura alternativa: {punto_mas_cercano['nombre']}...")

    # Ahora que está libre de la misión, sí que obedecerá el goto_location
    await drone_rescate.action.goto_location(punto_mas_cercano["lat"], punto_mas_cercano["lon"], float('nan'), 0)

    # Le damos un margen de tiempo realista para que llegue volando al punto de aterrizaje
    print("[EMERGENCIA] En ruta hacia la zona segura. Esperando aproximación...")
    await asyncio.sleep(30)

    print("[EMERGENCIA] Posición alcanzada. Iniciando descenso vertical de seguridad.")
    await drone_rescate.action.land()
```

Figura 13. Script para ejecutar protocolo de emergencia. (Fuente Propia).

4.2.3 Mission Mode

Misión MM-01. Inspección de planta fotovoltaica (Alarcos, Ciudad Real)

Para este primer escenario de prueba, la planificación de la misión para que el dron recorra la planta fotovoltaica se realiza directamente desde la interfaz de QGroundControl.

El proceso se inicia mediante la ejecución del lanzador .sh, situando al dron de forma precisa en las coordenadas asignadas en QGroundControl, esta vez sin necesidad de estar

abriendo el gemelo digital en Gazebo. Una vez cargada y enviada la misión, la aeronave procede al armado de sus motores y comienza la navegación de la zona siguiendo la secuencia de *waypoints* trazada en el software. El vuelo se mantiene a una altitud constante de entre 4 y 5 metros para simular de forma realista las condiciones de una inspección visual de los paneles solares. Toda la misión transcurre sin ninguna incidencia; el dron barre la totalidad del mapa previsto y, al finalizar su ruta, ejecuta exitosamente el comando RTL (*Return to Launch*) configurado al final de la misión para regresar de forma segura a su punto de origen.

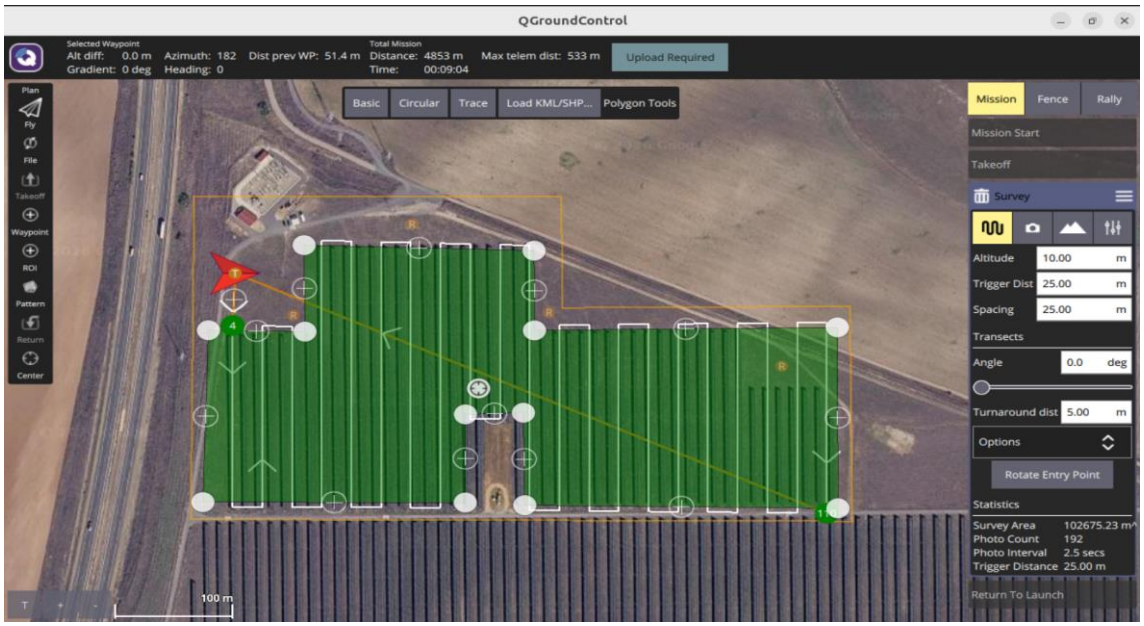


Figura 14. Misión creada en QGroundControl para paneles. (Fuente Propia).

Misión MM-02. Vigilancia de nave industrial (Coslada)

El segundo escenario de prueba emplea la misma metodología de planificación anterior, adaptándola al entorno industrial situado en Coslada. En esta ocasión, la misión se configura a una altitud de vuelo de 25-30 metros. Al igual que en la primera prueba, la aeronave completa la ruta planificada en la interfaz de manera totalmente satisfactoria.

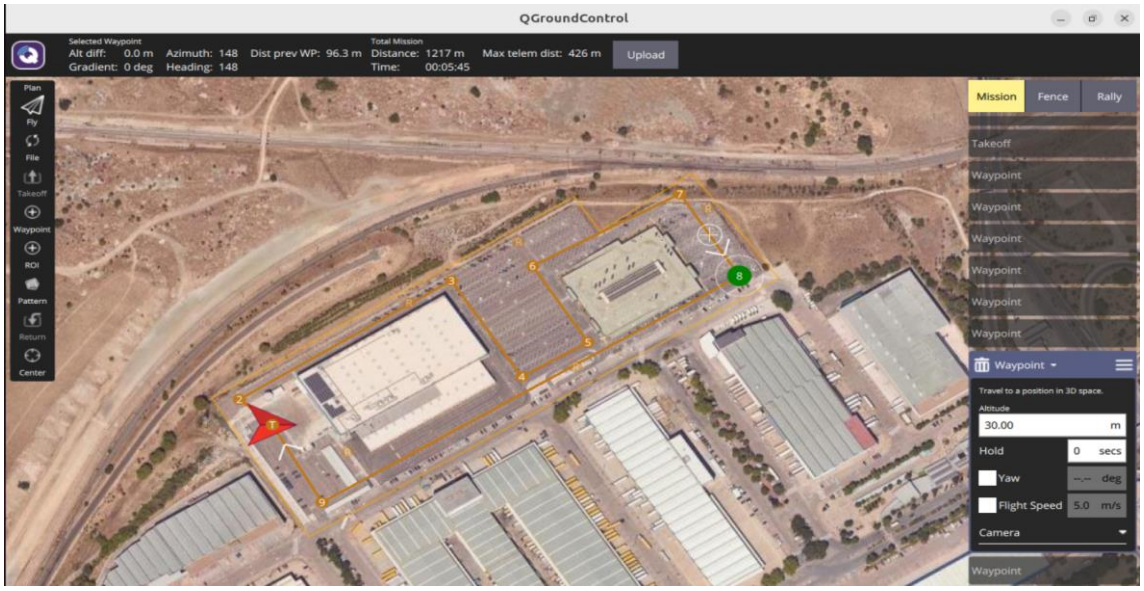


Figura 15. Misión creada en QGroundControl para la nave industrial. (Fuente Propia).

Misión MM-03. Sistemas de restricción espacial y protocolos de emergencia

Para esta última prueba, la validación se centra en la parametrización de las restricciones espaciales y la gestión de contingencias, configuradas íntegramente a través de las herramientas de QGroundControl. Se comprueba el correcto funcionamiento del sistema de *Geofencing*, asegurando que el dron respeta las geometrías dibujadas en el mapa de la estación base, capaz de evadir activamente la navegación por zonas de exclusión (Geofence poligonal interno) y de mantenerse operando estrictamente dentro del área permitida (Geofence externo). En segundo lugar, se evalúa el protocolo de fallos a través de la asignación de *Rally Points* (RP) en la planificación de la misión.

El sistema demuestra que, en caso de sufrir una falla simulada durante el vuelo, la aeronave utiliza los parámetros cargados desde QGroundControl para calcular de manera autónoma cuál es el punto de encuentro más cercano a su posición actual, ejecutando un aterrizaje de emergencia rápido y seguro para minimizar cualquier riesgo operativo en la simulación.

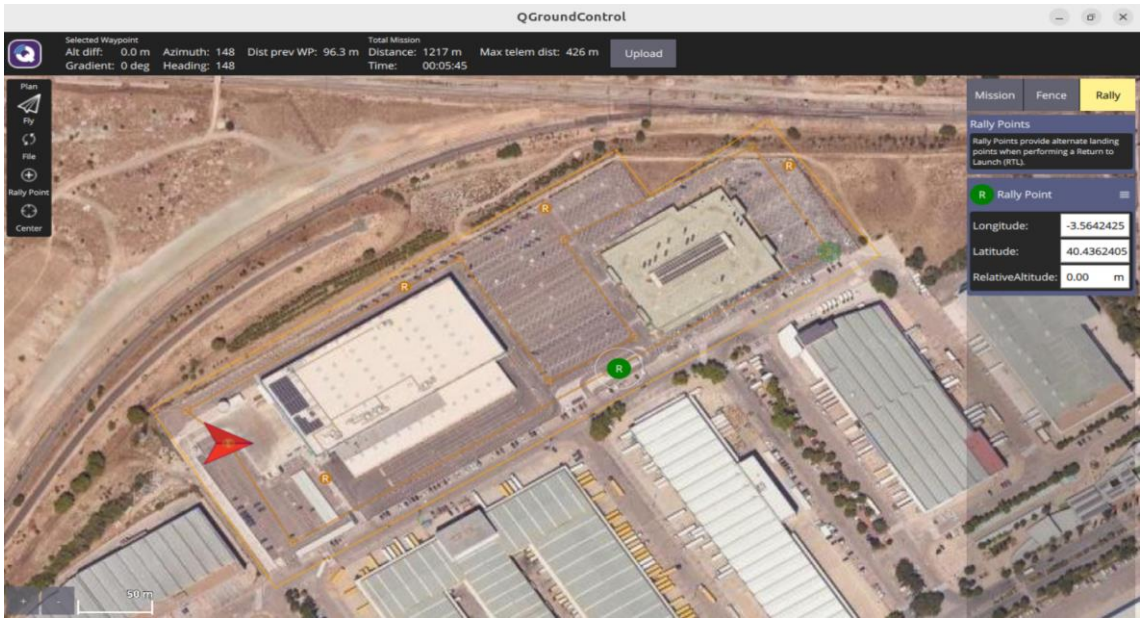


Figura 16. Definición de Rally Points en el mapa de la nave industrial. (Fuente Propia).

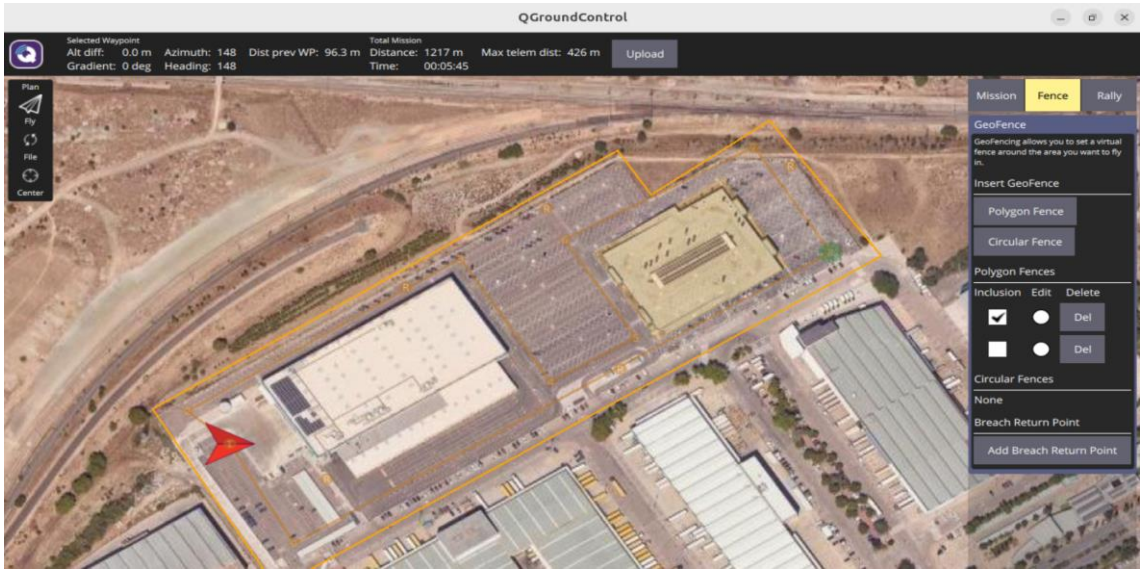


Figura 17. Definición Geofence interior en nave industrial. (Fuente Propia).

Para el mapa de los paneles solares, tal y como se puede apreciar en la Figura 14, la zona verde en la ruta es la zona habilitada para volar para el dron. Además, aquellos "*waypoints*" que salen con la nomenclatura de R, son los denominados *Rally Points* (RP).

4.3 Discusión y Limitaciones: Dificultades

Durante el desarrollo de este proyecto, una de las dificultades técnicas más relevantes se encontró en la creación y el modelado del entorno tridimensional dentro de **Gazebo**, ya

que el editor nativo del simulador presenta una complejidad considerable y, además, las limitaciones de rendimiento inherentes al trabajo dentro de una máquina virtual penalizan de forma notable la fluidez del proceso de diseño. Ante esta situación, se decidió trasladar el flujo de trabajo de modelado fuera del entorno virtualizado y emplear **Blender** como herramienta externa para la construcción, optimización y texturizado del gemelo digital, lo que permitió trabajar con mayor precisión y estabilidad. Una vez completado el diseño, el entorno fue exportado e integrado nuevamente en el ecosistema de simulación, garantizando la compatibilidad con los recursos del motor físico y con la estructura interna del mundo simulado.

Paralelamente, otro de los retos más complejos y críticos del proyecto fue la alineación exacta entre las coordenadas locales de la simulación y las coordenadas geográficas reales del mapa, proceso que exigió una calibración meticulosa para conseguir que el origen del escenario virtual coincidiera con precisión milimétrica respecto a la cartografía real. Lograr que la posición inicial del dron fuera idéntica tanto en QGroundControl como sobre el terreno tridimensional de Gazebo implicó un ajuste cuidadoso de los parámetros de georreferenciación, así como la verificación continua de la correspondencia entre los sistemas de referencia, dando lugar a un proceso laborioso pero imprescindible para garantizar la coherencia espacial del entorno simulado y la validez de las pruebas de navegación autónoma.

4.4 Posibles Mejoras

A partir de los resultados obtenidos en las simulaciones y las limitaciones técnicas identificadas, se propone la siguiente línea de trabajo futuro para optimizar y expandir las capacidades del sistema desarrollado.

4.4.1 Implementación de Evasión de Obstáculos Dinámicos:

La navegación basada en *waypoints* y en restricciones espaciales predefinidas opera, en su estado actual, sobre geometrías estáticas que no varían durante la misión, lo que implica que el dron ejecuta rutas planificadas con antelación sin capacidad de reinterpretar el entorno cuando este cambia. Una evolución significativa del sistema consistiría en integrar sensores adicionales, como cámaras de profundidad estereoscópicas o unidades **LiDAR**, junto con algoritmos avanzados de **visión artificial**,

de modo que la aeronave pudiera no solo seguir la trayectoria programada, sino también percibir el entorno en tiempo real y reaccionar ante elementos no mapeados.

La incorporación de estos sensores permitiría que el dron detectara obstáculos móviles o inesperados y recalculara dinámicamente su trayectoria mediante un proceso continuo de análisis geométrico, evitando colisiones y adaptando su comportamiento a las condiciones cambiantes del escenario operativo. Con ello, la aeronave pasaría de ejecutar misiones estrictamente deterministas a desplegar una autonomía reactiva mucho más robusta e inteligente, capaz de combinar la planificación previa con la interpretación instantánea del entorno, lo que representa un avance esencial hacia sistemas de navegación verdaderamente adaptativos y seguros.

Capítulo 5. Conclusiones

5.1 Resumen del trabajo

El presente Trabajo de Fin de Grado ha tenido como eje central el diseño, desarrollo e implementación de una arquitectura de software orientada a la planificación y ejecución de misiones autónomas en vehículos aéreos no tripulados (UAVs), respondiendo así a la creciente demanda industrial de sistemas capaces de automatizar tareas de inspección y monitorización en entornos complejos. Para ello, se construyó un ecosistema modular basado íntegramente en tecnologías de código abierto, articulado en torno al lenguaje **Python** y a la API **MAVSDK** como interfaz de comunicación con el firmware de control de vuelo **PX4**, lo que permitió establecer una plataforma flexible, escalable y fácilmente extensible.

A lo largo del desarrollo se abordó el problema de la navegación autónoma desde una perspectiva rigurosamente matemática y computacional. En primer lugar, se implementó un motor de transformación de coordenadas capaz de traducir trayectorias definidas en un plano métrico bidimensional al sistema global WGS84, requerido por los módulos de navegación del autopiloto PX4. Este componente se integró posteriormente en un orquestador principal asíncrono que gestiona el ciclo completo de la misión, delegando en PX4 la ejecución de los comandos de vuelo y garantizando la coherencia entre el espacio matemático de planificación y el entorno físico o simulado en el que opera el dron.

La validación del sistema se llevó a cabo mediante simulaciones **SITL** en el entorno tridimensional Gazebo, lo que permitió reproducir condiciones realistas sin comprometer la integridad del hardware. En este marco se modelaron dos escenarios industriales representativos: una planta fotovoltaica y una nave logística. Cuyo fin era el de demostrar que la arquitectura desarrollada es capaz de adaptarse a misiones con requisitos operativos muy distintos, desde patrones de barrido sistemático hasta circuitos de vigilancia perimetral. Los resultados obtenidos evidencian que el sistema no solo cumple con los objetivos planteados, sino que constituye una base sólida para futuras ampliaciones, ya sea mediante la incorporación de sensores adicionales, la integración de algoritmos de visión artificial o la migración hacia plataformas físicas en entornos reales.

5.2 Conclusiones sobre los Objetivos del Proyecto

A partir de los resultados obtenidos y de las pruebas de validación ejecutadas en los distintos módulos operativos, puede afirmarse que los objetivos principales definidos al inicio del proyecto se han alcanzado de manera plenamente satisfactoria. Se ha logrado construir e integrar un entorno de simulación robusto orientado a la planificación de trayectorias y al control de vuelo autónomo, consolidando la interoperabilidad entre arquitecturas complejas basadas en **ROS2** y **PX4**, y demostrando que ambos ecosistemas pueden cooperar de forma coherente dentro de un flujo de trabajo unificado.

La viabilidad operativa del sistema queda demostrada al validar casos de uso representativos, como la inspección de plantas fotovoltaicas y la vigilancia perimetral de naves industriales, en los que el dron ha sido capaz de ejecutar misiones complejas siguiendo secuencias precisas de *waypoints*, manteniendo las altitudes y parámetros de navegación requeridos tanto en rutas automatizadas mediante *scripts* como en trayectorias planificadas desde la estación base. Esta consistencia confirma que la arquitectura propuesta es adecuada para escenarios reales y que su comportamiento es estable incluso ante variaciones en la forma de definir la misión.

Del mismo modo, la implementación y verificación de los sistemas de seguridad críticos, especialmente la definición de zonas de exclusión mediante *geofencing* y el cálculo de aterrizajes de emergencia a través de *rally points*, garantiza que el diseño responde de manera fiable ante contingencias y fallos del sistema. A pesar de los retos técnicos asociados a la creación y georreferenciación de los gemelos digitales, la metodología adoptada ha permitido establecer una plataforma de pruebas sólida y funcional, que constituye una base tecnológica excelente para futuras migraciones hacia hardware físico y para la incorporación de capacidades avanzadas como **visión artificial** o **evasión dinámica**.

5.3 Líneas Futuras de Trabajo

Aunque el sistema desarrollado constituye una base operativa completa y funcional, la naturaleza modular y escalable de la arquitectura diseñada abre la puerta a múltiples líneas de mejora y expansión que podrían abordarse en trabajos futuros.

La primera de ellas, y quizás la más inmediata, sería la validación del sistema en un entorno HITL (Hardware-in-the-Loop), desplegando el código sobre un autopiloto físico como Pixhawk para comprobar que el comportamiento observado en simulación se mantiene cuando el firmware corre sobre hardware real. Este paso intermedio permitiría evaluar aspectos que SITL no puede reproducir fielmente, como las latencias reales del microcontrolador o las interferencias electromagnéticas, y serviría como puente hacia la realización de pruebas de vuelo real con un dron físico, donde entrarían en juego factores adicionales como el viento, las turbulencias o la precisión del GPS en exteriores.

Una segunda línea de evolución especialmente relevante para los casos de uso planteados sería la integración de visión artificial en el sistema de navegación. Equipar la aeronave con una cámara RGB o termográfica y procesar el flujo de vídeo en tiempo real mediante librerías como OpenCV o modelos de detección basados en redes neuronales como YOLO permitiría no solo sobrevolar los paneles solares de forma sistemática, sino detectar automáticamente anomalías como roturas físicas o puntos calientes indicativos de fallos en las células fotovoltaicas, añadiendo así una capa de inteligencia perceptiva al sistema que lo acercaría a una solución de inspección industrial real.

Por último, una mejora significativa en términos de robustez operativa sería la incorporación de evasión dinámica de obstáculos. El sistema actual basa su navegación en waypoints precalculados y zonas de exclusión estáticas mediante geofencing, lo que lo hace vulnerable ante la aparición de elementos imprevistos en la trayectoria, como maquinaria en movimiento o estructuras temporales. Integrar sensores LiDAR o cámaras de profundidad que alimenten un algoritmo de replanificación en tiempo real supondría un salto cualitativo importante, dotando al sistema de una capacidad de adaptación dinámica al entorno que es imprescindible en escenarios industriales reales.

ANEXOS

ANEXO I. Script “main.py”

```
import math

import asyncio

from mavsdk import System

from navegacion_paneles import PilotoAutonomoPaneles


async def main():

    drone = System()

    print("[CENTRAL] Conectando con los sistemas del dron...")

    await drone.connect(system_address="udp://:14540")


    print("[CENTRAL] Sincronizando telemetría y satélites GPS...")

    async for state in drone.core.connection_state():

        if state.is_connected:

            break


    async for health in drone.telemetry.health():

        if health.is_global_position_ok and
health.is_home_position_ok:

            print("[CENTRAL] Todos los sistemas de navegación
listos.")

            break


    # ============================================================

    # CAPTURA AUTOMÁTICA DE COORDENADAS

    # ============================================================

    print("[CENTRAL] Detectando posición HOME fijada por el
simulador...")

    async for home in drone.telemetry.home():

        lat_base = home.latitude_deg

        lon_base = home.longitude_deg
```

```
        print("[CENTRAL] Base detectada automáticamente -> Lat:
{lat_base}, Lon: {lon_base}")

        break


    # Inicializamos al piloto pasándole las coordenadas capturadas en
tiempo real

    piloto = PilotoAutonomoPaneles(lat_base=lat_base,
lon_base=lon_base)


    print("[CENTRAL] Transfiriendo plan de vuelo al Piloto
Automático...")

    # CORRECCIÓN: Llamamos a la nueva función de barrido dinámico

    await piloto.cargar_ruta_paneles(drone)


    print("[CENTRAL] Activando motores...")

    await drone.action.arm()


    print("[CENTRAL] Orden de despegue ejecutada. Iniciando patrulla
autónoma.")

    await drone.mission.start_mission()


    async for progress in drone.mission.mission_progress():

        print(f"[MONITOR] Pasando por punto de vigilancia:
{progress.current} de {progress.total}")

        if progress.current == progress.total:

            print("[CENTRAL] Patrulla completada con éxito.")

            break


    print("[CENTRAL] Regresando a la estación de carga...")

    await drone.action.return_to_launch()

    # Añadimos 2 segundos de pausa para asegurar que el dron reciba

    # la orden de Return To Launch antes de que Python cierre el
programa

    await asyncio.sleep(2)

    print("[CENTRAL] Desconectando sistemas. Fin del programa.")
```

```
async def emergencia_rally_points():

    print("\n[EMERGENCIA] Ejecución interrumpida por el usuario
(Ctrl+C).")

    print("[EMERGENCIA] Conectando sistema de rescate...")

    drone_rescate = System()

    await drone_rescate.connect(system_address="udp://:14540")


    # 1. Recuperamos la posición HOME (Base) fijada al inicio

    lat_home, lon_home = 0.0, 0.0

    async for home in drone_rescate.telemetry.home():

        lat_home = home.latitude_deg

        lon_home = home.longitude_deg

        break


    # 3. Obtenemos la posición actual exacta del dron al momento del
corte

    lat_actual, lon_actual = 0.0, 0.0

    async for position in drone_rescate.telemetry.position():

        lat_actual = position.latitude_deg

        lon_actual = position.longitude_deg

        break


    print(f"[EMERGENCIA] Posición de fallo: Lat {lat_actual:.5f}, Lon
{lon_actual:.5f}")


    # 5. Ejecutamos la maniobra evasiva según lo que haya decidido la
matemática

    print(f"[EMERGENCIA] El punto más cercano es
{punto_mas_cercano['nombre']} a {distancia_minima:.1f} metros.")


    if punto_mas_cercano["es_base"]:

        print("[EMERGENCIA] Activando protocolo nativo RTL...")

        await drone_rescate.action.return_to_launch()

    else:
```

```
        print("[EMERGENCIA] Desviando trayectoria. Abortando misión
activa...")


        # Forzamos al dron a ponerse en modo HOLD (frena en seco y
cancela la misión)

        await drone_rescate.action.hold()

        await asyncio.sleep(5) # Le damos cinco segundos para procesar
el cambio de modo


        print(f"[EMERGENCIA] Redirigiendo a zona segura alternativa:
{punto_mas_cercano['nombre']}...")


        # Ahora que está libre de la misión, sí que obedecerá el
goto_location

        await
drone_rescate.action.goto_location(punto_mas_cercano["lat"],
punto_mas_cercano["lon"], float('nan'), 0)


        # Le damos un margen de tiempo realista para que llegue
volando al punto de aterrizaje

        print("[EMERGENCIA] En ruta hacia la zona segura. Esperando
aproximación...")

        await asyncio.sleep(120)


        print("[EMERGENCIA] Posición alcanzada. Iniciando descenso
vertical de seguridad.")

        await drone_rescate.action.land()
```

ANEXO II Script “navegación.py”

```
import math

from mavsdk.mission import (MissionItem, MissionPlan)


class PilotoAutonomoPaneles:


    def __init__(self, lat_base, lon_base):

        # Guardamos la posición que el mapa detectó automáticamente

        self.lat_base = lat_base

        self.lon_base = lon_base

    def mover_en_metros(self, metros_norte, metros_este):

        """Calcula una nueva coordenada GPS sumando metros exactos a
la base."""

        r_tierra = 6378137.0

        d_lat = metros_norte / r_tierra

        d_lon = metros_este / (r_tierra * math.cos(math.pi *
self.lat_base / 180.0))


        nueva_lat = self.lat_base + (d_lat * 180.0 / math.pi)

        nueva_lon = self.lon_base + (d_lon * 180.0 / math.pi)

        return nueva_lat, nueva_lon


    async def cargar_ruta_paneles(self, drone):

      # Trayectoria en metros relativos: [Metros Norte, Metros este,
Altitud]

      # El Norte y Sur (+/-), El Este y Oeste (+/-)

        ruta_relativa = [

        ]

        waypoints = []

        for punto in ruta_relativa:

            n_lat, n_lon = self.mover_en_metros(punto[0], punto[1])


            wp = MissionItem(
```

```
                n_lat,
                n_lon,
                punto[2],
                20.0,
                True,
                float('nan'),
                float('nan'),
                MissionItem.CameraAction.NONE,
                float('nan'),
                float('nan'),
                float('nan'),
                float('nan'),
                float('nan'),
                MissionItem.VehicleAction.NONE
            )
            waypoints.append(wp)

        print(f"[CENTRAL] Generados {len(waypoints)} waypoints.
Empaquetando...")
        plan = MissionPlan(waypoints)
        await drone.mission.clear_mission()
        await drone.mission.upload_mission(plan)
        print("[CENTRAL] Ruta subida al dron correctamente.")
```

ANEXO III. Diseño Gemelos Digitales en 3D

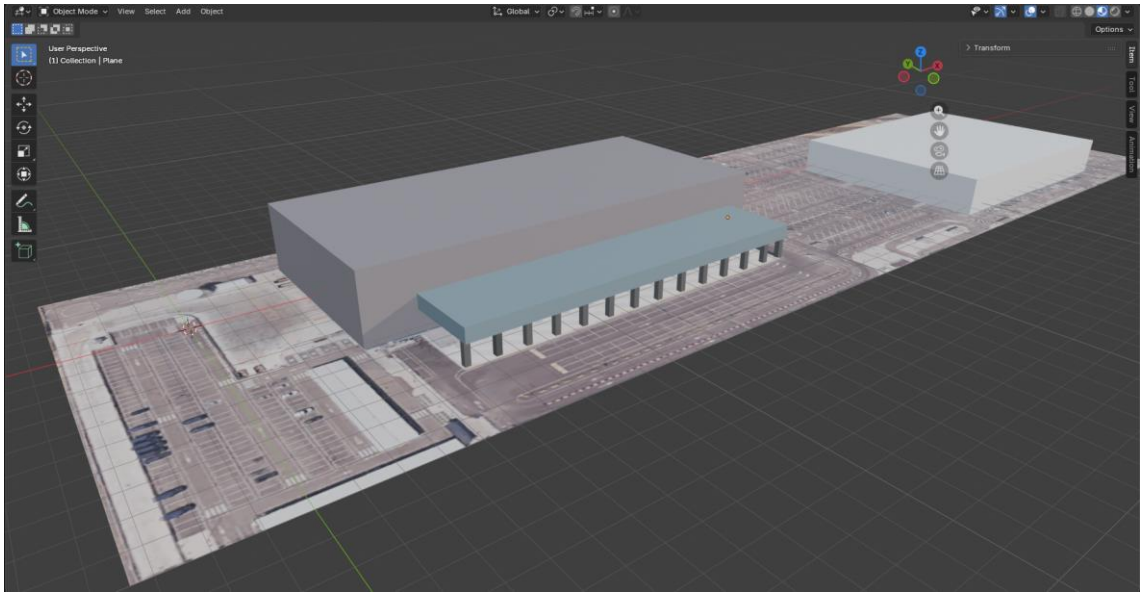


Figura 18. Diseño del Gemelo Digital de la Nave Industrial. (Fuente Propia).

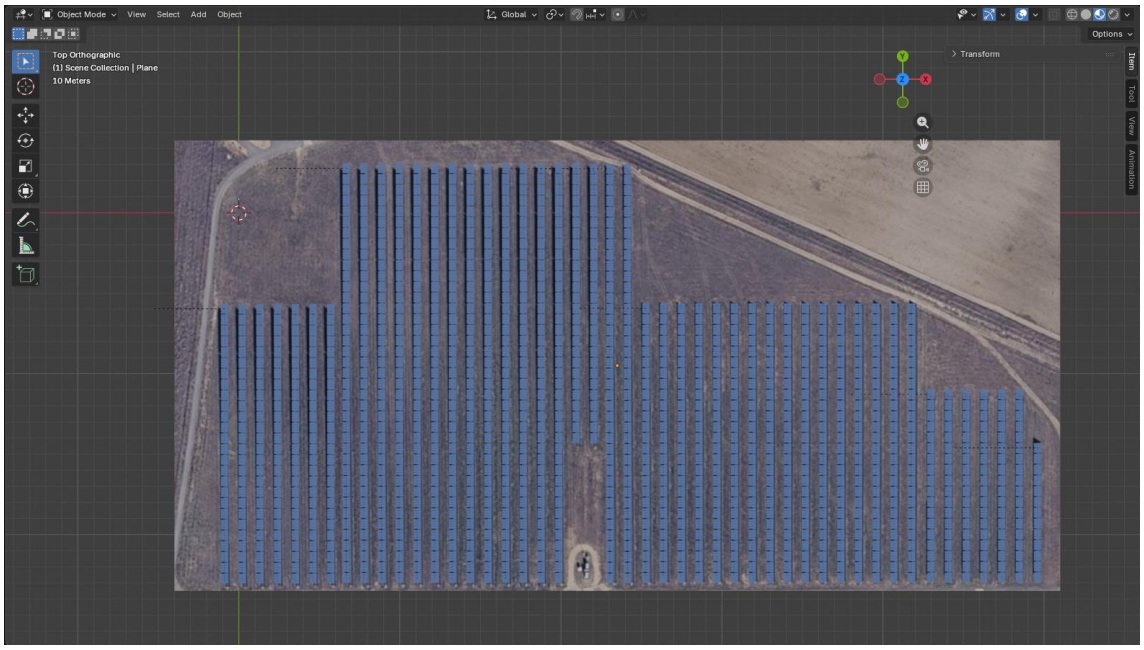


Figura 19. Plano Ortogonal de la planta fotovoltaica en Blender. (Fuente Propia).

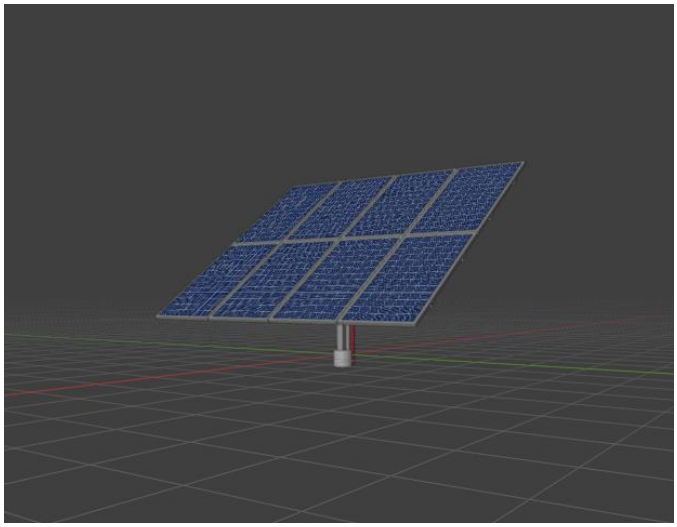


Figura 20. Panel Solar. (Fuente [12]).

BIBLIOGRAFÍA

[1] Jorge Luiz. (2021, 5 de mayo). *Warehouse Gazebo Simulation* [Video]. YouTube. https://www.youtube.com/watch?v=3MY5E4lOR98

[2] Aerostack 5.0. (2022, 13 de enero). *How to Launch Warehouse Inventory application with Gazebo simulator* [Video]. YouTube. https://www.youtube.com/watch?v=9c6axGrjE30

[3] Google. (s.f.). [*Google Maps*]. Recuperado el 2 de julio de 2026, de https://www.google.es/maps

[4] Google. (s.f.). [*Google Earth*]. Recuperado el 2 de julio de 2026, de https://www.google.es/intl/es/earth/index.html

[5] Redes Plus. (2025, 15 de abril). *Como INSTALAR VMWARE Workstation Pro Gratis en Windows 2025* [Video]. YouTube. https://www.youtube.com/watch?v=UbhQsY1WzRk&t=1s

[6] Blender Foundation. (s.f.). *Thanks*. Recuperado el 2 de julio de 2026, de https://www.blender.org/thanks/

[7] infinitedimensions. (Año, 1 de enero). *Ultimate Blender Road Techniques: The Best Tips for Creating Realistics Roads* [Video]. YouTube. https://www.youtube.com/watch?v=25Jl3pR801I

[8] GeoFan. (2026). *Buscador de latitud y longitud | Encontrar coordenadas GPS en un mapa*. https://geofan.org/es/coordinates

[9] Aliane, N. (2024). *A survey of open-source UAV autopilots*. *Electronics*, *13*(23), Artículo 4785. https://doi.org/10.3390/electronics13234785

[10] Aliane, N. (2026). *A survey of Pixhawk/PX4 autopilot and its impact on research and education. Computers, Materials & Continua*, *87*(3), Artículo 6. https://doi.org/10.32604/cmc.2026.078545

[11] Pexels. (s.f.). [Fotos de UAV]. Recuperado el 2 de julio de 2026, de https://www.pexels.com/search/uav/

[12] anggadwiwahyu**.** (2021, 3 de agosto). *3D model Solar Panel* [Modelo 3D]. TurboSquid. https://www.turbosquid.com/FullPreview/1768997

[13] Boletín Oficial del Estado. (2024, 5 de junio). *Real Decreto 517/2024, de 4 de junio, por el que se desarrolla el régimen jurídico para la utilización civil de sistemas de aeronaves no tripuladas (UAS)*. (BOE-A-2024-11388). https://www.boe.es/boe/dias/2024/06/05/pdfs/BOE-A-2024-11388.pdf

[14] Dronecode Foundation. (s.f.). *MAVLink developer guide*. Recuperado el 4 de julio de 2026, de https://mavlink.io/en/

[15] Dronecode Foundation. (s.f.). *PX4 user guide*. PX4 Autopilot. Recuperado el 4 de julio de 2026, de https://docs.px4.io/main/en/

[16] European Union Aviation Safety Agency [EASA]. (s.f.). *Civil drones*. Recuperado el 4 de julio de 2026, de https://www.easa.europa.eu/en/domains/civil-drones

[17] Open-Source Robotics Foundation. (s.f.). *Gazebo docs: Get started*. Gazebo Simulator. Recuperado el 4 de julio de 2026, de https://gazebosim.org/docs/latest/getstarted/

[18] Open-Source Robotics Foundation. (s.f.). *ROS 2 documentation: Jazzy Jalisco*. Robot Operating System (ROS). Recuperado el 4 de julio de 2026, de https://docs.ros.org/en/jazzy/